# *Thesis*

# Efficient and Accurate Non-Metric $k$-NN Search with Applications to Text Matching

Leonid Boytsov

CMU-LTI-18-006

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

**Thesis Committee:**
Dr. Eric Nyberg, Carnegie Mellon University, Chair
Dr. Jamie Callan, Carnegie Mellon University
Dr. Alex Hauptmann, Carnegie Mellon University
Dr. James Allan, University of Massachusetts Amherst
Dr. J. Shane Culpepper, the Royal Melbourne Institute of Technology

*Submitted in partial fulfillment of the requirements*
*for the degree of Doctor of Philosophy.*

# Abstract

In this thesis we advance state-of-the-art of the non-metric $k$-NN search by carrying out an extensive empirical evaluation (both and intrinsic) of generic methods for $k$-NN search. This work contributes to establishing a collection of strong benchmarks for data sets with generic distances.

We start with intrinsic evaluations and demonstrate that non metric $k$-NN search is a practical and reasonably accurate tool for a wide variety of complex distances. However, somewhat surprisingly, achieving good performance does not require distance mapping/proxying via metric learning or distance symmetrization. Existing search methods can often work directly with non-metric and non-symmetric distances. They outperform the filter-and-refine approach relying on the distance symmetrization in the filtering step.

Intrinsic evaluations are complemented with extrinsic evaluations in a realistic text retrieval task. In doing so, we make a step towards replacing/complementing classic term-based retrieval with a generic $k$-NN search algorithm. To this end we use a similarity function that takes into account subtle term associations, which are learned from a parallel monolingual corpus. An *exact* brute-force $k$-NN search using this similarity function is quite slow. We show that an *approximate* search algorithm can be 100-300 times faster at the expense of only a small loss in accuracy (10%). On one data set, a retrieval pipeline using an approximate $k$-NN search is twice as efficient as the C++ baseline while being as accurate as the Lucene-based fusion pipeline. We note, however, that it is necessary to compare our methods against more recent ranking algorithms.

The thesis is concluded with a summary of learned lessons and open research questions (relevant to this work). We also discuss potential challenges facing a retrieval system designer.

# Acknowledgments

This work would have been impossible without encouragement, inspiration, help, and advice of many people. Foremost, I would like to thank my advisor Eric Nyberg for his guidance, encouragement, patience, and assistance. I greatly appreciate his participation in writing a grant proposal to fund my research topic. I also thank my thesis committee: Jamie Callan, James Allan, Alex Hauptmann, and Shane Culpepper for their feedback.

I express deep and sincere gratitude to my family. I am especially thankful to my wife Anna, who made this adventure possible, and to my mother Valentina who encouraged and supported both me and Anna.

I thank my co-authors Bileg Naidan, David Novak, and Yury Malkov each of whom greatly helped me. Bileg sparked my interest in non-metric search methods and laid the foundation of our NMSLIB library [218]. Yury made key improvements to the graph-based search algorithms [197]. David greatly improved performance of pivot-based search algorithms, which allowed us to obtain first strong results for text retrieval [45].

I thank Chris Dyer for the discussion of IBM Model 1; Nikita Avrelin and Alexander Ponomarenko for implementing the first version of SW-graph in NMSLIB; Yubin Kim and Hamed Zamani for the discussion of pseudo-relevance feedback techniques (Hamed also greatly helped with Galago); Chenyan Xiong for the helpful discussion on embeddings and entities; Daniel Lemire for providing the implementation of the SIMD intersection algorithm; Lawrence Cayton for providing the data sets, the bbtree code, and answering our questions; Christian Beecks for answering questions regarding the Signature Quadratic Form Distance; Giuseppe Amato and Eric S. Tellez for help with data sets; Lu Jiang for the helpful discussion of image retrieval algorithms; Vladimir Pestov for the discussion on the curse of dimensionality; Mike Denkowski for the references on BLUE-style metrics; Karina Figueroa Mora for proposing experiments with the metric VP-tree applied directly to non-metric data. I also thank Stacey Young, Jennifer Lucas, and Kelly Widmaier for their help.

This research was primarily supported by the NSF grant #1618159: "Matching and Ranking via Proximity Graphs: Applications to Question Answering and Beyond.". Furthermore, I was supported by the Open Advancement of Question Answering Systems (OAQA) group as well as by the U.S. Army Research Laboratory and the U.S. Army Research Office under contract/grant number W911NF-10-1-0533.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

The focus of this thesis is $k$ nearest neighbor ($k$-NN) search, which is a widely used computer technology with applications in machine learning (ML), data mining, information retrieval (IR), and natural language processing (NLP). There has been a staggering amount of effort invested in designing new and improving existing $k$-NN search algorithms (see e.g., [65, 262, 272, 305]). This effort has been placed disproportionately on techniques for searching in metric spaces, in particular, on search methods for the Euclidean space.

Search methods for non-metric spaces received far less attention. One common approach to non-metric indexing involves mapping data to a low-dimensional Euclidean space. The goal is to find a mapping without large distortion of the original similarity measure [134, 146]. Jacobs et al. [146] review various projection methods and argue that such a coercion is often against the nature of a similarity measure, which can be, e.g., intrinsically non-symmetric. In this thesis, we provide experimental evidence to support this claim (see §§2.3.2.3-2.3.2.4). Alternatively the metric distance can be learned from scratch [28]. In that, Chechik et al. [70] contended that in the task of distance learning enforcing symmetry and metricity is useful only as a means to prevent overfitting to a small training set. However, when training data is abundant, it can be more efficient and more accurate to learn the distance function in an unconstrained bilinear form. Yet, this approach does not necessarily results in a symmetric metric distance [70].

In this thesis we aim to reduce the gap between metric and non-metric indexing by designining and evaluating *generic* non-metric $k$-NN search methods that do not necessarily rely on explicit mapping to a metric space. Generality is an important consideration: We want our methods to be as simple as possible while being applicable to a wide variety of data sets. Thus we can save engineering and intellectual effort of devising custom solutions for specific distances.

We start by proposing a novel modification of the VP-tree [43], which adapts to non-metric data sets by means of a simple algorithm to learn a pruning rule (§2.2.3). In a series of intrinsic evaluations, we show that (1) this algorithm has performance competitive with some of the state-of-the-art methods, (2) it can be improved by adopting a simple distance-transformation approach inspired by the TriGen algorithm [271]. Yet, our modification of VP-tree can be outstripped by proximity-graph (neighborhood-graph) methods (see § 2.3) on high dimensional data.

Good results achieved in intrinsic evaluations give us confidence to carry out extrinsic evaluation of generic $k$-NN search methods on text retrieval tasks (§ 3). It has been long recognized that $k$-NN search shows a promise to make retrieval a *conceptually* simple optimization procedure

[164]. It may be non-trivial to design an effective similarity function and formulate a good query. However, once this problem is solved, the objective is to find data points with highest similarity scores. This approach may permit an efficient separation of labor so that data scientists can focus on development of effective similarity models without worrying about low-level performance issues, while designers of retrieval algorithms and software engineers can focus on development of more efficient and/or scalable search approaches for generic data sets.

Yet, $k$-NN search proved to be a challenging problem due to the *curse of dimensionality*. This emotionally-charged phrase was coined by Richard Bellman to describe a number of undesirable effects caused by an exponential increase in volume related to adding extra dimensions to a vector space [29]. There is empirical and theoretical evidence that in a *general* case this problem can hardly be solved both exactly and efficiently in a high-dimensional setting [34, 65, 239, 268, 311]. For some data sets, e.g., in the case of vectors with randomly generated elements, exact methods degenerate to a brute-force search for just a dozen of dimensions [34, 311].

However, in special cases there exist shortcuts permitting an efficient and accurate *exact* $k$-NN search, even when the similarity function is non-metric. In particular, text retrieval systems answer queries in a pipeline fashion, where the first pipeline stage uses a simple inner-product similarity such as BM25 [254]. Even though this similarity is neither metric nor symmetric, it may be possible to answer inner-product queries exactly and efficiently. Achieving efficiency is possible because (1) term-document matrix is sparse, (2) queries are typically short, and (3) an inner-product similarity function decomposes into a sum of term-specific values (one for each query term) such that computing one summand requires occurrence statistics of only a single query term. [1]

For substantially more sophisticated similarity functions efficient and exact $k$-NN search is rarely possible. For this reason, such functions are applied only to a small number of candidate entries, which are retrieved using a fast-to-compute inner-product similarity. This filter-and-refine approach employed in many text retrieval systems does not carry out an exact $k$-NN search. Yet, the information retrieval community believes it is sufficiently accurate. This belief hinges on the assumption that term-based retrieval using inner-product similarity generates a reasonably complete list of candidate documents. However, this assumption is not fully accurate, in particular, because of a *vocabulary gap*, i.e., a mismatch between query and document terms denoting same concepts. The vocabulary gap is a well-known phenomenon. Furnas et al. [118] showed that, given a random concept, there is less than a 20% chance that two randomly selected humans denote this concept using the same term. Zhao and Callan [328] found that a term mismatch ratio—i.e., a rate at which a query term fails to appear in a relevant document—can be as much as 50%.

Furthermore, according to Furnas et al. [118], focusing only on a few synonyms is not sufficient to effectively bridge the vocabulary gap. Specifically, it was discovered that, after soliciting 15 synonyms describing a single concept from a panel of subject experts, there was still a 20% chance that a new person coined a previously unseen term. To cope with this problem, Furnas et al. [118] proposed a system of *unlimited term aliases*, where potential synonyms would be interactively explored and presented to the user in a *dialog* mode. The order of exploration would depend on a probability that synonyms are associated with original query words.

[1] We thank James Allan for reminding us about properties (1) and (3).

An established *automatic* technique aiming to reduce the vocabulary gap is a *query expansion*. It consists in expanding (and sometimes rewriting) a source query using related terms and/or phrases. For efficiency reasons, traditional query expansion techniques are limited to dozens of expansion terms [57]. Using hundreds or thousands of expansion terms may be infeasible within a framework of term-based inverted files. In contrast, we hypothesize that a system of unlimited term aliases may be easier to implement within a more generic framework of *$k$-nearest neighbor search* ($k$-NN search).

More generally, we conjecture that $k$-NN search can be a replacement for term-based retrieval when term-document similarity matrix is dense. Note, however, that an ultimate objective of such a replacement is not efficiency: We aim to design fast enough algorithms, which have a potential to be more accurate than classic term-based retrieval. More specifically, we hope that *eventually* $k$-NN search with a more accurate similarity function will be able to find document entries that are hard or impossible to obtain using a standard retrieval pipeline where candidate generation is based on a simple inner-product similarity function. We also hope that using the more sophisticated computationally expensive similarity combined with approximate search may allow us to be simultaneously more efficient and effective.

We make a step towards verifying our conjecture by exploring a scenario where queries can be long and a similarity function is complex, e.g., it may take into account subtle word associations mined from a parallel monolingual corpus. Although exact brute-force search using such a similarity is slow, we demonstrate that an *approximate* $k$-NN search algorithm can be reasonably efficient at the expense of only a small loss in accuracy. The retrieval module employing this approximate $k$-NN search algorithm may be simultaneously more effective and efficient than a classic bag-of-terms approach. We believe that this finding opens up new possibilities for designing effective retrieval pipelines. Obtaining this result required a novel approach to computation of IBM Model 1 (see § 3.1.1.2), which is a key ingredient of the similarity function. We argue that a similar trick can speed up other similarity functions as well.

The rest of the thesis unfolds as follows: In the remaining part of this chapter, we formulate research questions and present our research contributions (§ 1.1). Chapter 2 is devoted to intrinsic evaluation of $k$-NN search methods. In § 2.1 we introduce the problem, terminology, review various distance functions, and explain why $k$-NN search is hard. In § 2.2, we discuss related work (§ 2.2.2) and our novel modification of VP-tree (§ 2.2.3). In § 2.3 we present experimental results, which are divided into two sections. In § 2.3.1, we focus on cross-method comparison using a diverse set of data. In § 2.3.2, we experiment with a less diverse but supposedly more challenging data, which employs only non-metric distances. In Chapter 3, we apply $k$-NN search to retrieval of answer-bearing passages. Our experiments include both factoid and non-factoid question-answering (QA) data sets. In Chapter 4, we discuss our findings, hypotheses, and open questions. We conclude the thesis with a brief summary of results (§ 4.2).

## 1.1 Research Questions and Contributions

The main objective of this thesis is to improve our understanding of applicability of generic *approximate* k-NN search methods to complex similarity functions, including non-symmetric and non-metric distances. Achieving this objective requires answering several research questions

and carrying out intrinsic and extrinsic evaluations using diverse data sets. The major research questions include the following:

1. Which generic approaches for $k$-NN search can be used with complex similarities? Can these approaches be fast and practical?
2. Instead of using specialized methods for $k$-NN search, is it possible and feasible to simplify the search problem by distance approximation/proxying? In particular, we want to know if simple pre-filtering using either a learned metric distance or a symmetrized version of initially non-symmetric similarity can result in efficient and accurate solution.
3. Retrieval pipelines commonly rely on a term-based search algorithm to obtain candidate records, which are subsequently re-ranked. Some candidates are missed by this approach, e.g., due to a vocabulary mismatch. Can this issue be mitigated by using a generic $k$-NN retrieval algorithm, where a similarity function can take into account subtle term associations? In other words, can incorporating sophisticated similarity features into early retrieval stages may be useful?

We hope that answering these questions will help us understand when it makes most sense to employ approximate $k$-NN search in a text-retrieval pipeline.

To answer the first research question, we pose several questions of a smaller scope with an objective to evaluate applicability of three classes of search methods:

1. tree-based methods;
2. proximity-graph based methods;
3. permutation-based search methods (pivoting approaches relying on comparison of pivot rankings).

With respect to these classes of search methods, the smaller-scope questions are as follows:

1. Is it possible to learn a non-metric pruning rule for a classic VP-tree?
2. Do proximity-graph based methods require (or benefit from) index- or query-time symmetrization?
3. Permutation-based pivoting search methods (or simply permutation methods) are filter-and-refine approaches which define a simple projection from a generic space to the Euclidean space. How accurate are these permutation-based projections?

In this thesis we advance state-of-the-art of the non-metric $k$-NN search by finding answers for some of the stated research questions and carrying out one of the most extensive empirical evaluations of generic methods for $k$-NN search. In doing so we contribute to establishing a collection of strong benchmarks for generic distances.

The major findings of this work include the following:

1. We demonstrate that non metric $k$-NN search is a practical and reasonably accurate tool for a wide variety of complex distances. However, somewhat surprisingly, achieving good performance does not require distance mapping/proxying via metric learning or distance symmetrization. Existing approaches can work directly with non-metric and non-symmetric distances: This works better compared to the filter-and-refine approach that relies on the distance symmetrization in the filtering step (see § 2.3.2.3-2.3.2.4).
2. In §3 we make a step towards replacing/complementing classic term-based search with

a generic $k$-NN retrieval algorithm, where a similarity function can take into account subtle term associations, in particular, associations learned from a parallel monolingual corpus. While an *exact* brute-force $k$-NN search using this similarity function is slow, an *approximate* search algorithm can be 100-300 times faster at the expense of only a small loss in accuracy (10%). We also provide evidence for our conjecture that incorporation of sophisticated similarity into retrieval can be beneficial. In particular, on one data set, a retrieval pipeline using an approximate $k$-NN search is twice as efficient as the C++ baseline while being as accurate as the Lucene-based fusion pipeline (§3.3). We note, however, that it is necessary to compare our methods against more recent ranking algorithms [51, 94, 114, 186, 198].

3. Despite demonstrating practicality of $k$-NN search in a text retrieval domain, we note that it is not yet clear if $k$-NN search can find many substantially different results compared to a classic term-based retrieval pipeline. In particular, we learn that instead of carrying a $k$-NN search using a more accurate similarity, comparably good results can be achieved by using this similarity to re-rank a sufficiently long list of candidates (generated by a term-based retrieval pipeline).

The technical/algorithmic contributions include the following:

1. In § 2.2.3 we propose a novel modification of the VP-tree and demonstrate that for data of moderate or small dimensionality it is a strong baseline, which can outperform several previously proposed methods while being more generic. It can be further improved by using a light-weight distance transformation in the spirit of the TriGen algorithm [271].
2. We demonstrate that an existing graph-based algorithm SW-graph [196] has excellent performance for challenging data sets with non-symmetric distances, even if good accuracy is not achievable by a filter-and-refine approach with the symmetrized distance (§2.3.2.4). In fact, we observe that in the case of SW-graph this filter-and-refine approach (with the symmetrized distance) is always inferior to using the original distance *directly*. At the same time, performance of SW-graph can be improved by using a *different*, e.g., symmetrized, distance *only* at index time (i.e., to construct a graph).
3. On question-answering data sets, we find that a combination of BM25 and IBM Model 1 scores trained on purely lexical representations is 18-29% in NDCG20 and 20-30% in ERR20 more effective than BM25 alone (see § 3.2.1. However, using linguistic features does not result in substantially better models (§ 3.2.2).
4. We observe that using a large training corpus is crucial to good performance of IBM Model 1. In that, the degree of improvement by using millions of training pairs instead of dozens of thousands is quite remarkable (see Tables 3.9 and 3.10 in § 3.2.1). To our knowledge, this a novel finding with respect to performance of IBM Model 1 in the text-retrieval domain. It allows us to substantially outperform the solution by Surdeanu et al. 2011 in a community question answering task (§ 3.2.1).
5. We propose a novel set of tricks to accelerate computation of IBM Model 1 in two scenarios (§ 3.1.2.1). In particular, to calculate IBM Model 1 scores faster, we build a special small inverted index for each query, which is somewhat similar to a query expansion. This index is not precomputed in advance: It is built when a new query arrives. We also use a similar

approach to accelerate computation of all-pivot distances, i.e., distances from a data point to every pivot. We are not aware of any prior work where these efficiency tricks are used. In addition, we believe that this precomputation approach is applicable to other scenarios, e.g., when the similarity is computed as the cosine distance between word embeddings.

6. We show that permutation-based pivoting search methods with randomly sampled pivots are outperformed by other state-of-the-art methods. The quality of projection pivot-induced projections can vary a lot. In particular, performance is quite poor for sparse high-dimensional data. Motivated by this finding our co-author David Novak showed that use of pivots composed from randomly selected terms allowed to improve performance dramatically.

The thesis is concluded with a summary of learned lessons and open research questions relevant to this work (§ 4.1). We also discuss potential challenges facing a retrieval system designer. In particular, we highlight the following:

1. There is a need for an accurate similarity that can outperform the baseline similarity by a good margin. Otherwise, gains achieved by employing a more sophisticated similarity are invalidated by the inaccuracy of the search procedure.
2. Compared to the exact brute force search, an approximate $k$-NN search may require many fewer computations of the distance/similarity function. However, a typical reduction in the number of distance computations observed in our experiments (sensible for an accurate retrieval) is within two-three orders of magnitude. This is a relatively small improvement that can be easily canceled out by inefficiencies of the distance function. Hence, an efficient $k$-NN search is hardly possible without careful optimization of the distance function computation.
3. Because the $k$-NN recall decreases with $k$, an approximate $k$-NN search is most effective when the underlying similarity generates only a few highly ranked candidates.

# Chapter 2

# Intrinsic Evaluation of $k$-NN Search Methods for Generic Spaces

In this chapter, we carry out multiple intrinsic evaluations, where we study efficiency-effectiveness trade-offs of *generic* methods for $k$-NN search. Note that that generality is an important consideration, because it allows us to save engineering and intellectual effort that would otherwise be spent on tailoring custom solutions for various distance functions and data sets. The focus of this evaluation is on *high-accuracy* retrieval methods (recall close to 0.9), because we suppose that high accuracy in intrinsic evaluations is crucial to achieving good accuracy with respect to extrinsic metrics. In § 3.3, we present experimental evidence supporting this conjecture in the text-retrieval domain.

Another important requirement is efficiency. We believe that achieving high efficiency requires keeping data and indices in memory. Our belief aligns well with a gain in popularity of high-throughput main memory databases (see, e.g. [156]). For large data sets that do not fit into memory of a single server, it can be necessary to split data into multiple shards each of which is indexed and searched separately. In our experiments we simulate a case of indexing and querying a small enough shard, which fits into a memory of one server.

The experimental section § 2.3 is preceeded by the introductory section (§ 2.1), which discusses the basics of $k$-NN search, and by the section where we describe relevant search methods (§ 2.2).

## 2.1 Introduction to $k$-NN Search

### 2.1.1 Problem Formulation

To automate a top-$k$ retrieval task, real-world objects are represented in some abstract, usually vectorial, form. We commonly refer to these real-world object representation as data *points* in some *space*. The space is equipped with with a *distance function* $d(x, y)$, which is used to measure dissimilarity of data points $x$ and $y$. The space may have infinite number of points, but a search algorithm always deals with a finite subset of data. In what follows, we introduce additional concepts related to our work. Notation used in this chapter is summarized in Table 2.1.

The value of $d(x, y)$ is interpreted as a degree of dissimilarity: The larger is $d(x, y)$ the more

| Notation | Explanation |
|---|---|
| $x, y, z$ | data (set) points or vectors (vectors are assumed to be *column* vectors) |
| $q$ | a query point |
| $k$ | the number of neighbors to retrieve |
| $k_c$ | the number of candidate points from which one selects $k$ actual neighbors |
| $\pi$ | a pivot point |
| $A, B, C$ | matrices |
| $x^{\mathrm{T}}$ | a matrix/vector transpose operation |
| $\|\|x\|\|_p$ | a $p$-norm of vector x: $\|\|x\|\|_p \left( \sum_{i=1}^{m} \|x_i\|^p \right)^{1/p}$ |
| $L_p$ | a vector space with distance $d(x, y) = \|\|x - y\|\|_p$ |
| $\|\|x\|\|_2$ | the Euclidean norm of vector $x$, which is the $p$-norm for $p = 2$ |
| $\mathrm{sign}\,(x)$ | the sign of $x$ |
| $L_2$ | a Euclidean space |
| $d(x, y)$ | the value of a distance between points $x$ and $y$ |
| $d_{\text{reverse}}(x, y)$ | the value of the *argument-reversed* distance, i.e., $d(y, x)$ |
| $d_{\text{sym}}(x, y)$ | the value of a *symmetrized* distance between points $x$ and $y$ |
| $\kappa(x, y)$ | the value of a kernel function |
| $B(c, r)$ | a ball, i.e., a set of points within distance $r$ from center $c$ : $B(c, r) = \{x \vert d(x, c) \leq r\}$ |
| $\langle x, y \rangle$ | an inner/dot product between vectors: $\langle x, y \rangle = x^{\mathrm{T}} y = \sum_i x_i \cdot y_i$ |
| $\mathrm{R}_{\text{knn}}@k$ | $k$-NN recall at rank $k$, i.e., an overlap between $k$ true nearest neighbors and $k$ data points returned by a search method |
| $[C]$ | Iverson bracket (equal to one if and only if condition $C$ is true) |

Table 2.1: Notation of Chapter 2

| Property name | Description | Premetric | Semimetric | Metric |
|---|---|---|---|---|
| (1) Non-negativity | $d(x, y) \geq 0$ | ✓ | ✓ | ✓ |
| (2) Zero self-dissimilarity | $d(x, x) = 0$ | ✓ | ✓ | ✓ |
| (3) Identity of indiscernibles | $d(x, y) = 0 \Leftrightarrow x = y$ | ✗ | ✓ | ✓ |
| (4) Symmetry | $d(x, y) = d(y, x)$ | ✗ | ✓ | ✓ |
| (5) Triangle inequality | $d(x, z) \leq d(x, y) + d(y, z)$ | ✗ | ✗ | ✓ |

Table 2.2: Key properties of distance functions

dissimilar points $x$ and $y$ are. Alternatively, the smaller is $d(x, y)$ the more similar the points are. Some distances are non-negative and become zero only when $x$ and $y$ have the highest possible degree of similarity. However, in general, we do not impose any restrictions on the value of the distance function. Specifically, the value of the distance function can be negative and negative distance values indicate higher similarity than positive ones. For example, if $d(x, q) < 0 < d(y, q)$, $x$ is closer (or less dissimilar) to $q$ than $y$.

A *similarity function* (a measure of similarity) is a related concept. However, unlike the distance function, the value of the similarity function is large for similar objects and small for dissimilar one. One can convert a similarity function into a distance function (and vice versa) by applying a monotonically decreasing mapping, e.g., negation.

To define the problem of $k$-NN search, we assume that there are two finite set of points: a query set and a data set, which have an empty intersection. The points in the data set have unique identifiers. Given query $q$ the objective of the nearest neighbor search is to find a data set point *most similar* to $q$ (ties can be resolved arbitrarily e.g., by choosing a data set point with a minimum identifier; see Exposition 2.1 for more details). This point is called the *nearest neighbor* and corresponds to the *smallest* distance between the query and a data set object. If the distance is not symmetric, two types of queries can be considered: *left* and *right* queries. In a *left* query, a data point compared to the query is always the first (i.e., the left) argument of $d(x, y)$.

A natural generalization of the nearest neighbor search is $k$-nearest neighbor ($k$-NN ) search.

**Exposition 2.1: Ties in the definition of the $k$-neighborhood**

In the case of ties, i.e., when the distance to the $k$-th nearest is equal to the distance to the $k+i$-th nearest neighbor for some $i > 1$, there can be multiple ways of defining the true set of nearest neighbors. One definition is to include all $k + i$ elements. However, in the case of real-valued distances, ties are infrequent. Because, in this thesis we do not deal with integer-valued distances, a choice of a tie-resolution method has negligible impact on overall system performance.

Here the objective is to retrieve $k$ closest points instead of merely one. Because the data set is finite, the minimization problem associated with $k$-NN search is well defined. In that, nearest neighbors can be found by a brute-force comparison of the query with every data set point. $k$-NN search belongs to a class of similarity search methods. Unlike exact methods aiming to find an exact copy of the query, a similarity search retrieves data points resembling the query, which are not necessarily exact copies. Also note that our definition of the distance function essentially equates the concepts of top-$k$ retrieval and $k$-nearest neighbor search

Given a distance function, we can define a ball $B(c, r)$ with a given center $c$ and a radius $r$. The ball $B(c, r)$ is defined in a standard way as a set of of points within the distance $r$ from the point $c$. If the distance is not symmetric, one should distinguish between a *left* and a *right* query ball, which are defined as follows:

$$B_{left}(c, r) = \{x | d(x, c) \leq r\}$$

$$B_{right}(c, r) = \{x | d(c, x) \leq r\}$$

To simplify notation, we omit the specification of the ball (left or right) because it is usually clear from context.

*Range search* is another similarity search operation. It consists in finding all data set points within a query ball $B(q, r)$ with a user-specified radius $r$. While range search is an interesting and challenging problem on its own, in our work it is used only as an auxiliary operation. Specifically, for the class of space-partitioning methods discussed in § 2.2.2.1, $k$-NN search is simulated as a range search with a shrinking radius, which is equal to the distance between the query to the $k$-th closest data point discovered by the search procedure.

It is always possible to find $k$ nearest neighbors via a brute-force search, i.e., by computing the distance from the query to every data set point, which, in turn, allows us to retrieve $k$ data points with smallest distances to the query. The brute-force search method is exact in the sense that it does not miss any true nearest neighbors. Yet, it typically has a high computational cost, because the number of distance computation is equal to the number of data set points. In some cases, it is possible to design a more efficient exact search algorithm. However, most frequently, additional gains in efficiency are obtained by carrying out an *approximate* search algorithm, which may miss some of the true nearest neighbors. Exact search algorithms are evaluated in terms of their efficiency only, while approximate search algorithms are evaluated in terms of their efficiency-effectiveness trade-offs. In all cases, performance metric values are averaged over the set of queries.

We use two metrics to assess efficiency: improvement in efficiency (over brute-force search) and reduction in the number of distance computations (again, compared to brute-force search). In the case of a multi-threaded benchmark with $T$ threads, we compare against the multi-threaded variant of brute-force search that carries out $T$ searches in parallel, using $T$ threads.

Improvement in efficiency is a wall-clock speed up over brute-force search. For example, the method with improvement in efficiency 10 is much better than a method with improvement in efficiency two. The fact that we compare against the potentially slow brute-force search does not mean that we use it as baseline that we attempt to outstrip. The wall-clock time of brute-force search is used merely as a convenient reference point, which allows us to characterize efficiency of

| Name | $d(x, y)$ | Premetr. | Semimetr. | Metric |
|---|---|---|---|---|
| $\alpha$-$\beta$ diverg. [243] | $\sum_{i=1}^{m} p_i^{\alpha+1} q_i^{\beta}$ | ✗ | ✗ | ✗ |
| Negative inner product | $-\langle x, y\rangle = -\sum_{i=1}^{m} x_i y_i$ | ✗ | ✓ | ✗ |
| Itakura-Saito distance [143] | $\sum_{i=1}^{m} \left[\frac{x_i}{y_i} - \log \frac{x_i}{y_i} - 1\right]$ | ✓ | ✗ | ✗ |
| Kullback-Leibler diverg. (KL-div.) [171] | $\sum_{i=1}^{m} x_i \log \frac{x_i}{y_i}$ | ✓ | ✗ | ✗ |
| Rényi diverg. [251] | $\frac{1}{\alpha-1} \log \left[\sum_{i=1}^{m} p_i^{\alpha} q_i^{1-\alpha}\right],\ \alpha > 0,\ \alpha \neq 1$ | ✓ | $\alpha = \frac{1}{2}$ | ✗ |
| Jensen-Shannon diverg. (JS-div.) [103] | $\frac{1}{2}\text{KL-div}\left(x, \frac{x+y}{2}\right) + \frac{1}{2}\text{KL-div}\left(y, \frac{x+y}{2}\right)$ | ✓ | ✓ | ✗ |
| Cosine distance | $1 - \frac{\langle x, y\rangle}{\|\|x\|\|_2 \|\|y\|\|_2}$ | ✓ | ✓ | ✗ |
| $L_p$ $(p > 0)$ | $\left[\sum_{i=1}^{m} (x_i - y_i)^p\right]^{1/p}$ | ✓ | ✓ | $p \geq 1$ |
| Angular | $\arccos\left(\frac{\langle x, y\rangle}{\|\|x\|\|_2 \|\|y\|\|_2}\right)$ | ✓ | ✓ | ✓ |
| Euclidean ($L_2$) | $\|\|x - y\|\|_2 = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$ | ✓ | ✓ | ✓ |

Table 2.3: Examples of distance functions

The closer to the bottom row, the more nice properties the distance have.

the method in a data set independent way. For example, the cost of computing $L_2$ between 1000-dimensional vectors is ten times higher than the cost of computing $L_2$ between 100-dimensional vectors. Because of this difference, run times obtained on these different data sets are not directly comparable, yet, comparing improvements in efficiency is meaningful due to normalization by brute-force search time.

Reduction in the number of distance computations is an indirect measure, which tells us how many fewer times a distance function is called during a search compared to brute-force search. For example, if an algorithm answers a query by computing the distance from the query to only 10% of data points, the reduction in the number of distance computations is 10. Reduction in the number of distance computations roughly corresponds to but do not always directly translates into improvement in efficiency There are two main reasons for a potential mismatch: a higher cost of random access (compared to the cost of sequential access, which is used during brute-force search) and bookkeeping costs associated with pruning of the search space. [1] Reduction in the number of distance computations informs us what improvement in efficiency could be if bookkeeping and memory access costs were greatly reduced.

Improvement in efficiency and the reduction in the number of distance computations are *standard* metrics used in assessment of generic *distance-based* $k$-NN search methods: Distance-based search methods can only use values of mutual data point distance, but cannot exploit the structure of our data. In this case, the brute-force search, which needs to compute the distance from the query to every data set point, is a natural reference point, which represents essentially the worst case scenario. In some domains, notably in the case of the inner-product (or cosine-similarity) search for sparse vectors (one common example is text searching using TF$\times$IDF similarities), there exist more efficient reference points. However, such reference points cannot be used as universal benchmarks. Furthermore, in these rare cases when such benchmarks exist, one important question to answer is how much they improve over the naive brute-force search.

There are two approaches to effectiveness evaluation: intrinsic and extrinsic. In an intrinsic evaluation, there is no external information regarding data point relevance. Therefore, we blindly assume that every data point is relevant and measure how accurately the *complete* $k$-neighborhood of the query is preserved by a retrieval method. A popular accuracy measure is recall at $k$, which we denote as $\mathrm{R_{knn}}@k$. $\mathrm{R_{knn}}@k$ is equal to the fraction of true nearest neighbors retrieved by a $k$-NN search. It can be also defined as an overlap between the exact result set and the result set returned by an approximate search algorithm.

In the $k$-NN community, intrinsic evaluation is the most popular evaluation approach. Because there is no need to obtain extrinsic quality judgements from humans—which is slow and expensive—one can carry out a large-scale and, thus, reliable comparison of retrieval methods using a small budget. Although we do not know how well intrinsic quality translates into human-perceived retrieval quality, there seems to be a consensus that the closer is the neighbor to the query, the more important it is to preserve it and rank correctly [310]. It is worth noting that in our work, there is always a refinement step where we compute a distance to the query. Thus, we can miss a nearest neighbor, but we correctly preserve the relative order of retrieved neighbors.[2]

[1] For example, we estimate than the cost of a randomly reading a 100d vector from memory can be higher than the cost to compute the Euclidean distance between two 100d vectors!

[2] Technically, due to non-deterministic nature of floating-point operations, the relative order may sometimes be violated. However, such ranking errors are quite infrequent.

In the case of $\text{R}_{\text{knn}}@k$, errors at all ranks are equally penalized. However, this is not always desirable. Consider, for example, a 10-NN search and two result sets. The first result set contains five closest nearest neighbors (i.e., entries with ranks from one through five). The second result set contains five farther ones (i.e., entries with ranks six through 10). Intuitively, the first result set is much more accurate than the second one, but they are identical in terms of recall (which is 50% in both cases).

To deal with this issue, it has been proposed to impose higher penalties for errors at high ranks. Webber et al. [310] surveys such approaches and proposes a new metric called a *rank-biased overlap* (RBO). Webber et al. suggested to compute the effectiveness as a sum of values of $\text{R}_{\text{knn}}@i$ with decaying weights $w_i$ for $1 \leq i \leq k$. The weights are defined as follows:

$$w_i = (1 - p) \cdot p^{i-1}$$

The choice of the weights is inspired by a simple user behavior model commonly used in information retrieval, in which parameter $0 < p < 1$ represents a persistence of the user when she inspects the ranked result list in a top-down fashion [61, 209]. This parameter is interpreted as a probability that after examining first $i$ items in the result set, the user proceeds to examine the $i + 1$-st item.

One rationale for choosing quickly decaying weights is to make RBO convergent. Convergence guarantees that "the weight of the unseen, conceptually infinite tail of the lists is limited, and does not dominate the weight of the seen, finite prefix" [310]. In addition to basic RBO, Webber et al. propose to compute the minimum, the maximum, and the extrapolated values [310]. Extrapolated RBO is computed under assumption that $\text{R}_{\text{knn}}@k'=\text{R}_{\text{knn}}@k$ for $k' > k$. In that, Webber et al. demonstrate that extrapolated RBO was a better proxy for the mean average precision than Kendall-Tau [159] and the average overlap (an average of all $\text{R}_{\text{knn}}@i$ for $1 \leq i \leq k$). In 3.3.3.2, we test how well RBO fares against a simpler $k$-NN recall.

### 2.1.2 Properties of Distance Functions

As mentioned previously, any function can be interpreted as a distance function. Quite often, however, we use more restricted classes of distances—premetric, semimetric, and metric—that possess additional properties. Below, we describe these classes and provide examples of corresponding distance function (Table 2.3). Key desirable properties of distance functions are listed in Table 2.2. For a more thorough treatment of various distances, the reader is addressed to the book by Deza and Deza [90].

First we note that it can be advantageous to confine ourselves to using a *non-negative* distance function. This property is often augmented with a requirement that all *self-dissimilarities* are zero, which, in turn, corresponds to the reflexivity of the binary relation $\{(x, y) \mid d(x, y) = 0\}$. The class of distances satisfying these properties are called *premetrics*. A stronger version of the reflexivity is an *identity of indiscernibles*, which "forbids" the distance to be zero for distinct data points. Jointly these properties ensure that the distance function has a global minimum of zero, which is achieved on identical elements (and perhaps for some other pairs of points if the identity of indiscernibles does not hold).

Dealing with premetrics is convenient both algorithmically and theoretically, especially when the distance has additional nice properties. However, there are numerous practically interesting

cases when (1) there is no global minimum distance value (2) the minimum of $d(x, y)$—for a fixed $x$—can be achieved when $x \neq y$. One simple example is a vector space model, where the similarity is computed as an inner product between a query and a document vector [19, 199]. The objective of retrieval is to find a data point with the maximum inner product, which corresponds to finding a data point minimizing the negative inner product (see Table 2.3) between query and document vectors.

The value of the (negative) inner product $-\langle x, q\rangle$ is proportional to the Euclidean norm of $x$. Because there is potentially no upper bound on the norm of $x$, it is possible to obtain arbitrarily small values of the distance. Furthermore, for the fixed $x$, $-\langle x, x\rangle$ is not the minimum possible distance value of $-\langle x, y\rangle$, which is another noteworthy anomaly.[3]

The anomalies of the negative inner product would disappear if the space were restricted to data points on a surface of a unit sphere. In this case, the global minimum equal to $-1$ would be achieved only at identical points. In fact, adding one to distance values would produce a symmetric premetric known as the *cosine distance* (see Table 2.3). There is a similar trick to reduce inner product search to the cosine distance search when data points have a limited norm (see Exposition 2.5). Yet, there are other practically important distances where the range of distance values is query specific, does not have a universal global minimum, and queries can be less similar to itself than to other data points. Furthermore, in a generic case, there is no transformation that can eliminate these anomalies and make, e.g., distance ranges obtained for different queries compatible. One example of such distance is the negative value of IBM Model 1 scores [53], which is used extensively in Chapter 3.

Note that most search algorithms are designed for symmetric distances only. According to Schleif and Tiño [266], "Symmetry is in general assumed to be valid because a large number of algorithms become meaningless for asymmetric data." However, we do not think that symmetry is a fundamental property that should be enforced at all possible costs. In particular, it does not make sense when the data is compositional and the distance involves comparison of component parts. For example, if $y$ contains a copy of $x$, we may expect that $x$ matches $y$ perfectly, but not the other way around. This could happen if, in addition to the copy of $x$, $y$ contains other components that do not match $x$ well.

Divergences, or statistical distances, is another class of (generally) non-symmetric distances. A famous example of a statistical distance is KL-divergence [171], which is widely used in machine learning [58, 281, 307] (see Table 2.3). Schleif and Tiño note that divergences "are very popular for spectral data analysis in chemistry, geo-, and medical sciences" [266].

A symmetric premetric satisfying the triangle inequality and identity of indiscernibles is a true metric, which is, perhaps, the most studied class of distances. The triangle inequality is key to designing efficient divide-and-conquer search algorithms in metric spaces. However, a lot of useful distances violate this property. They include statistical distances (such as KL-divergence), the Smith-Waterman score [273], the dynamic time warping distance [259], and the normalized

[3]To see why these simple observations are true, one can represent the distance as the negative product of three terms (two vector norms and the inner product between normalized vectors):

$$-\langle x, y\rangle = -||x||_2||y||_2 \left\langle \frac{x}{||x||_2}, \frac{y}{||y||_2} \right\rangle.$$

**Exposition 2.2: Intrinsic dimensionality**

There are multiple ways to compute the intrinsic dimensionality. One popular measure is an expansion dimension [157]. Imagine that we take a ball and count the number of points inside this ball. Then, we double the radius and count the number of points in a larger ball. If for any possible combination of ball centers and radii, the number of points inside the larger ball is at most $c$ times larger than the number of points in the smaller ball, then the expansion dimension is equal to $\log c$. For example, for points uniformly distributed in a $d$-dimensional Euclidean space, the expansion constant is close to $2^d$ [157]. Interestingly, the expansion dimension is applicable not only to $L_2$, but to a large class of spaces. This allows us to define the dimensionality of data, even if the underlying space has no vectorial structure. Note, however, that—unlike the unambiguously defined representational dimensionality—there are multiple approaches to defining intrinsic dimensionality [4, 54, 166], but there is no single universally accepted definition.

Levenshtein distance. For distances arising in the field of bioinformatics, e.g., in the case of Smith-Waterman score [273], it was argued that " the non-metric part of the data contains valuable information and should not be removed" [266].

Although not every premetric satisfies the triangle inequality, it may still possess other properties allowing us to apply divide-and-conquer algorithms. Consider, for example, a property called a *reverse triangle inequality*, which is an easy-to-prove corollary of the original triangle inequality:

$$|d(z,x) - d(z,y)| \leq d(x,y) \tag{2.1}$$

A natural generalization of Eq. (2.1) is an inequality called $\mu$-defectiveness [1, 108]:[4]

$$|d(z,x) - d(z,y)| \leq \mu d(x,y), \mu > 0 \tag{2.2}$$

The data-dependent parameter $\mu$ is often unknown, but it can be inferred from data (see § 2.2.3).

In a vast majority of cases, the divide-and-conquer search algorithms rely on the triangle inequality or its generalization. Yet, alternative approaches exist as well. For example, recently Hetland [131] proposed to use the Ptolemy's inequality, which connects distances of four points in the following way:

$$d(x,y) \cdot d(z,t) + d(y,z) \cdot d(t,x) \geq d(x,z) \cdot d(y,t)$$

Note that the (reverse) triangle inequality and $\mu$-defectiveness are useful properties, but they provide a rather limited toolset to design search algorithms. The first tool in this toolset is a capability to compute a lower bound from a query to a bounding region. Such lower bounds—allowing us to safely prune unpromising partitions—are key to space-partitioning algorithms (see § 2.2.2.1). The second tool is a capability to compute an upper bound for the distance to the query when a query resides within a certain distance from a *pivoting* point. Pivots are objects that serve as reference points. They are typically randomly selected from the data set or created using a clustering algorithm such as k-means [190].

[4]For non-symmetric distances, there are two variants of $\mu$-defectiveness, which do not follow from one another.

**Exposition 2.3: Bregman divergences**

Bregman divergence is a distance function generated using a strictly convex, differentiable function $f(x)$ of vector argument $x$ using the following formula [47]:

$$D_f(x, y) = f(x) - f(y) - \langle \nabla f(y), x - y \rangle, \tag{2.4}$$

For example, if $f(x) = \langle x, \log x \rangle = \sum_i x_i \log x_i$, the corresponding Bregman divergence is KL-divergence. One can see that the value of a Bregman divergence is equal to the difference between the value of the function $f(x)$ and its first-order approximation.

Indeed, by reversing $\mu$-defectiveness, which generalizes the reverse triangle inequality, we obtain a desired upper bound

$$d(z, x) \leq \mu d(x, y) + d(z, y), \mu > 0. \tag{2.3}$$

Furthermore, as we can see from Eq. (2.2) and (2.3), $\mu$-defectiveness (and the reverse triangle inequality as a special case) provide the basis for an extended version of the well-known *neighbor-of-my-neighbor* property. The basic version of this *informal* property ensures that "the closest neighbor of my closest neighbor is my neighbor as well". The extended version allows to detect far points. It ensures that "if one point is close to a pivot, but another is far away, such points cannot be close neighbors". These properties can be summarized as follows:

- The closest neighbor of my closest neighbor is my neighbor as well;
- If one point is close to a pivot, but another is far away, such points cannot be close neighbors.

Although $\mu$-defectiveness implies a neighbor-of-my-neighbor property, the latter can hold in the absence of $\mu$-defectiveness. For example, it holds when the square root of the distance is $\mu$-defective, but not the original distance. In any case, it may be possible to design an efficient search algorithm based directly on the neighbor-of-my-neighbor property rather than on the underlying properties implying neighbor-of-my-neighbor. This nevertheless requires an appropriate formalization and/or quantification. One way of doing so is to measure overlap of pivot neighborhoods (see the description of Neighborhood APProximation Index in § 2.2.2.4).

Compared to generic metric spaces, vector spaces have a much richer set of capabilities, in particular, in the case of spaces endowed with the $L_p$ norm (see Table 2.3) of which the Euclidean space $L_2$ is, perhaps, the most studied instance. $L_p$ spaces in general and the Euclidean space $L_2$ in particular have multiple useful properties absent in generic metric spaces. Notably, these spaces are endowed with locality-preserving transformations that have strong theoretical guarantees. In particular, we can construct effective hash functions by computing the dot product between data/query vectors and vectors whose elements are sampled independently from a $p$-stable distribution [84]. This, in turn, permits constructing effective hashing schemes (see § 2.2.2.2). Dimensionality reduction via a linear (orthogonal) transformation such as a *random projection* [37, 83] or the principal component analysis is another powerful tool accessible only in $L_2$.

Furthermore, a success of pivoting methods—relying on pivots to reduce the number of direct comparisons between the query and data points—depends on existence of good pivots. K-means clustering [190] is one established method to generate pivots, which, among other fields, is

widely used in computer vision [15, 149, 270]. Developed originally for the Euclidean space, k-means relies on the property that a centroid of a set is a point that minimizes the sum of *squared* distances to set points. Interestingly, a similar property holds in the case of non-metric distances known as Bregman divergences (see Exposition 2.3). This allows one to use k-means in the case Bregman divergences as well [22, 58].[5] K-medoids is a generalization of k-means for generic, i.e., non-vectorial spaces [121, 158]. However, it is a more limited approach, because cluster centers (called *medoids*) can be selected only from the data set. For this reason, unlike k-means in Bregman spaces and $L_2$, where a cluster center truly minimizes a sum (of squared) distances from cluster points to the center, the selection of medoids can be widely sub-optimal. Consider, e.g., points evenly distributed on a sphere. The center of the sphere would be a (nearly) optimal cluster center, but in k-medoids a cluster center could be only a point on the sphere.

We conclude this subsection with a brief discussion of kernelized distances. Kernelized distances are defined by means of a symmetric similarity function $\kappa(x, y)$ (a *kernel* function) by the following equation:

$$d(x, y) = \kappa(x, x) + \kappa(y, y) - 2\kappa(x, y). \tag{2.5}$$

In a general case, kernelized distances are symmetric functions, which satisfy $d(x, x) = 0$, but they are not necessarily premetric and can be negative. However, the most commonly used kernels are positive-semidefinite. This implies that (1) for any sets of data points $\{x_i\}$, the kernel matrix $K_{ij} = \kappa(x_i, x_j)$ is positive-semidefinite, (2) there is a (perhaps, implicit) mapping $\phi(x)$ from the original space to a Hilbert space such that the kernel function can be represented as an inner product in the target space [203]:

$$\kappa(x, y) = \langle \phi(x), \phi(y) \rangle$$

The inner product is a symmetric positive-definite function of two arguments (i.e., $\langle x, x \rangle \geq 0$, $\langle x, x \rangle = 0; \Leftrightarrow x = 0$). It defines a metric distance function, which can be computed using values of the kernel function:

$$d(x, y) = \sqrt{\langle \phi(x) - \phi(y), \phi(x) - \phi(y) \rangle} = \sqrt{\langle \phi(x), \phi(x) \rangle + \langle \phi(y), \phi(y) \rangle - 2\langle \phi(x), \phi(y) \rangle} =$$

$$\sqrt{\kappa(x, x) + \kappa(y, y) - 2\kappa(x, y)}.$$

An indefinite kernel function does not define a mapping to a space with a positive-definite inner product. Yet, it defines an (implicit) mapping to an indefinite inner product space commonly referred to as a *Krĕin* space. The Krĕin space permits an orthogonal decomposition into two Hilbert spaces $\mathcal{H}_+$ and $\mathcal{H}_-$ each of which is endowed with a positive-definite inner product. In that the indefinite inner product in the Krĕin space is equal to the difference between inner products in $\mathcal{H}_+$ and $\mathcal{H}_-$[235, 266].

[5] In the case of Bregman divergences, the centroid minimizes the sum of Bregman distances to other points, not the set of squared distances. Also note that the squared Euclidean distance is a non-metric Bregman divergence itself.

**Exposition 2.4: Iris recognition problem**

Because the texture of the iris is determined by an epigenetic random process [86], a digital descriptor of iris texture is essentially a randomly generated bit vector with 2048 dimensions [85]. Due to the concentration of measure the distribution of the normalized Hamming distances between eye descriptors for any pair of different eyes is narrowly concentrated around 0.5 (including genetically identical eyes from one person) [85]. Because the textures of any two different eyes' irises differ to approximately the same degree, the problem of finding an eye most similar to a given one is pointless. However, the actual problem search problem consists in matching different iris descriptors of the *same* eye. Such descriptors, e.g., obtained by different scanning machines, exhibit a high degree of similarity [85], In this case, the search problem is meaningful. It can also be solved efficiently using near-duplicate detection techniques, which reduce the original problem to exact matching [129].

### 2.1.3 Need for Approximate Search

Finding $k$ nearest neighbors is a conceptually simple optimization problem, which can be solved by a brute-force search. However, carrying out $k$-NN search in a more efficient fashion is a notoriously hard task in a high-dimensional setting. The problem is that high-dimensional spaces have a number of counter-intuitive properties, such as emptiness of the space and a concentration of measure [239, 296]. This and other similar properties cause performance deterioration in many data processing algorithm, which is collectively referred to as the curse of dimensionality.

The concentration of measure deserves special attention. It means that, under a variety of conditions, as dimensionality increases the distribution of distances becomes narrowly concentrated around a non-zero mean value (which can also increase as the dimensionality grows). Furthermore, the distance to the nearest neighbor approaches the distance to farthest neighbor [34, 239]. In this case, there is theoretical [239, 268] and empirical [34, 311] evidence that many proposed *exact* search algorithms, i.e., algorithms reporting all true nearest neighbors, cannot outperform brute-force search. While it is formally established that certain indexing schemes are inefficient on high-dimensional workloads, it seems to be quite hard to prove that *every* conceivable search algorithm is "cursed" [239]. Also note that existing theory focuses on metric spaces, but there is empirical evidence that non-metric spaces are affected as well [43, 58].

Due to the curse of dimensionality, it is unlikely that there exist a general *efficient* solution to the *exact* $k$-NN search problem. However, it is still possible to carry out an efficient $k$-NN search in many practically interesting cases. The theoretical results are valid mostly for vectorial data where both data and query points are sampled randomly from the same distribution. Additionally, it is usually supposed that each dimension is sampled independently. The independence assumption is not realistic, but we do not know how making this assumption affects the accuracy of theoretical predictions. Furthermore, many real data sets are not truly high-dimensional and may, e.g., reside in a low-dimensional manifold [166]. In this case, we say that a data set has a low *intrinsic* dimensionality (see Exposition 2.2 for more details).

Another point of divergence between theory and practice is the assumption about query distribution. Specifically, there are cases where we cannot assume that data set points and queries

**Exposition 2.5: Reduction of inner product search to cosine similarity search**
To reduce inner product search to cosine similarity search, we first normalize each query $q_i$ (by dividing vector elements by the query's Euclidean norm $||q_i||_2$). We further normalize each data point vector by dividing its elements by the largest Euclidean norm of a data point in the data set. Finally, for each data point, we add an additional dimension, which is equal to $\sqrt{1 - ||x_i||_2^2}$, where $||x_i||$ is the norm of the normalized data point. For each query, we add an additional dimension, whose value is zero. It can be seen that the cosine similarity between transformed vectors is equal to the inner product in the original space divided by the product of the Euclidean norm of the query and the maximum Euclidean norm of a data set point. Because for a fixed query, the normalization coefficient is a non-zero constant, the maximum-inner product search using original data produces the same nearest neighbors as the cosine-similarity search using transformed vectors. This neat transformation was proposed by Neyshabur and Srebro [225].

originate from the same data generating process. One interesting example is an iris recognition problem, where data points are basically randomly generated data points located at nearly equal distances from each other while queries are obtained by small perturbations of the data (see Exposition 2.4 for more details).

Text retrieval is another important domain where queries may differ substantially from data points. Data points are long documents, which are often written in a proper natural language. In contrast, queries are often short sentences, which have only a simplified syntactic structure [289]. One common approach to retrieval is to ignore all the structure and treat both queries and documents as bags of words [19, 199]. Consequently, data and query points are represented as sparse vectors of large dimensionality. Even in this case there are substantial differences in the number of non-zero vector elements: Because a typical query has only a few words [282], query vectors are normally much sparser than document vectors.

The sparsity of the term-document matrix complemented by the sparsity of queries allows us to carry out an efficient *exact* $k$-NN search using a *decomposable* term-document similarity function. Decomposability means that the similarity is computed as a sum of term-specific values. In that, the value of of each term depends on the occurrence statistics of this term only. A commonly used type of decomposable similarity functions is a TF×IDF similarity function, which is an inner-product similarity equal to a sum of term-specific values. In turn, each of the term-specific values is equal to a product of term in-query term frequency, in-document frequency (often rescaled to account for a document length), and of an inverse document frequency `IDF`: [19, 199].[6] The negative value of the TF×IDF similarity can be used as a distance function between the document $x$ and the query $y$:

$$d(x, y) = -\sum_{x_i=y_i} \text{TF}(x)_i \cdot \text{TF}(y)_i \cdot \text{IDF}(y)_i \tag{2.6}$$

From Eq. (2.6), we can see that a TF×IDF distance does not have to be symmetric, because a

[6]Note that the inverse document frequency is a bit of a misnomer, because we do not use its raw value, but rather take a logarithm of the inverse frequency. Similarly, a term frequency is not always a true frequency: It often denotes an *unnormalized* number of occurrences.

query and a document generally can have different TF values for the same term. In fact, this is a typical setting: The query TF value is often equal to the number of times the term repeats in the query, whereas the document TF value is typically a number of occurrences in a document possibly normalized by the document length (see § 3.1.1.1 for an example).

When a query-document matrix is sparse and the queries are short, the inner-product similarity can be computed efficiently, partly, because some many documents do not have any query words and, thus, can be safely ignored during query processing. However, the search problem is harder if we the queries have similar statistical properties to documents, in particular, with respect to the number of unique terms. In this case, the query-document matrix can be considerably more dense and the exact maximum product search can be much less efficient. In addition, if both queries and documents can be modeled as objects sampled from the same high-dimensional random variable, nearest neighbors tend to be quite far from the query and the distance to a nearest neighbor tend to be close to the median distance from the query to documents. In other words, it can be hard to distinguish nearest neighbors from farther data points.

The simplicity of the TF×IDF similarity permits various optimizations for top-$k$ retrieval (see, e.g., [6, 93, 278]). A common optimization approach is to compute an upper bound for a document score: If the bound is lower than the score of the smallest top-$k$ score, one can discard the document without further evaluation. Obtaining the upper bound is possible, because we can (1) isolate contributions of specific terms to the TF×IDF similarity and (2) upper bound the values of these contributions using their local or global estimates.

To further underscore the simplicity of the TF×IDF similarity function in Eq. 2.6, we note that, although TF×IDF is not necessarily a symmetric measure, as we can see from Eq. 2.6, the respective top-$k$ retrieval problem is a maximum inner product search (which is a search using a symmetric similarity). The maximum inner product search can, in turn, be reduced to a cosine similarity search [225] (see Exposition 2.5 for details). Recasting the problem as the cosine similarity search permits application of efficient methods for $k$-NN search, e.g., application of the Locality-Sensitive Hashing (LSH) methods [5, 225]. Finally, if necessary, the cosine similarity search can be further reduced to a search using the angular distance, which is a *metric* function. [7]

To sum up, efficient $k$-NN search is often possible when one of the following is true:

- Data is intrinsically low-dimensional;
- Queries have substantially different properties from data points;
- The underlying structure of the problem is simple and permits an efficient computation shortcut (e.g., data points are sparse vectors and the distance is an element-wise decomposable function).

Much effort has been devoted to search algorithm in metric spaces, in particular, in $L_2$. Search methods for non-metric spaces received far less attention, partly because non-metric searching is more challenging. Due to diversity of properties, non-metric spaces lack *common* and *easily identifiable* structural properties such as the triangle inequality. There is, therefore, little hope that *fully* generic exact search methods can be devised. In fact, aside from decomposable distances on sparse data sets, we know only one broad class of non-metric distances, namely, Bregman

[7] A *monotonic* distance transformation of the cosine similarity—namely taking the inverse cosine of the cosine similarity value—produces a metric angular distance. Due to the monotonicity of such a transformation the $k$-NN search using the angular distance find the same set of neighbors as the search using the cosine similarity.

divergences (see Exposition 2.3), where exact non-metric search methods exist [58, 227, 327]. Note that these exact methods still suffer from the curse of dimensionality [43, 58].

Chen and Lian [71] claim to have proposed an *exact* search method for generic non-metric distance, based on the concept of local constant embedding. Chen and Lian use the idea of Roth et al. [255] to "fix" the triangle inequality by adding a data-dependent constant $C(x, y)$ to the distance value of $d(x, y)$ when $x \neq y$. Using an expensive algorithm—cubic in the number of data points or cluster centers—it may be, indeed, possible to "fix" the triangle inequality for all data point triples. However, it does not seem to be possible for previously unseen queries (see Exposition 2.6 for more details). In addition, their algorithm seems to be too expensive to be practical for large data sets.

In the absence of known structural properties, it may still be possible to carry out $k$-NN search efficiently by answering queries *approximately*, i.e., by finding only some of the nearest neighbors. Furthermore, this can be done in a generic fashion (see § 2.3): There are approximate search methods that work well for the Euclidean distance, for KL-divergence (which belongs to the class of Bregman divergences), and for the cosine similarity on sparse data sets [197, 220, 245]. In § 3 we also show that a generic approximate method works well for a hard-to-tackle and computationally expensive textual similarity, which is not readily decomposable.

To summarize, resorting to approximate searching allows us to achieve two important objectives: (1) weakening the curse of dimensionality, (2) designing sufficiently generic $k$-NN search methods. Because of these advantageous properties, we focus specifically on approximate methods for $k$-NN search.

## 2.2 Search Methods

Similarity searching in general and $k$-NN search in particular is an extensively studied area, and the current survey cannot do full justice to the vast body of related work. Instead, we focus on most relevant generic approaches, especially on approximate search methods for high-dimensional vector spaces as well as on generic non-metric search methods. For a more detailed discussion the reader is addressed to more comprehensive surveys and books [65, 262, 271, 305, 326].

Also note that—while generic vector space and metric-space methods have a broad applicability—there are also specialized methods for specific domains such as symbolic sequences [41, 120, 167, 222, 223], which are not covered here either. In addition, this survey excludes $k$-NN search methods designed specifically for near-duplicate detection such as a shingling approach of Broder [50] or hashing approaches used in iris recognition [129] which reduce similarity searching to exact matching. Specialized near-duplicate detection algorithms work well only for high similarity thresholds, i.e., small distances, and typically cannot be used for general-purpose $k$-NN retrieval (see Exposition 2.4 for an example of a relevant application).

As mentioned in § 1, one common approach to non-metric indexing involves mapping data to an easy-to-tackle (e.g., low-dimensional Euclidean) space with a hope that such mapping does not introduce a large distortion of the original similarity measure [134, 146]. Thus, it may seem to be appealing to divide methods into two classes: mapping and direct methods. However, this is problematic for several reasons. First, there is no well-defined boundary between mapping and direct methods. For example, proximity graphs (§ 2.2.2.3) might be indisputably direct methods,

**Exposition 2.6: Notes on Local Constant Embedding**

Chen and Lian [71] propose to modify the distance function by adding a "local" constant to the distance function. Their solution is designed for premetric distances, which are non-negative and symmetric. Consider a triple $x$, $y$, and $z$ that violates the triangle inequality:

$$d(x,y) + d(y,z) < d(x,z) \tag{2.7}$$

By setting $d'(x,y) = d(x,y) + C$ and choosing the constant $C$ to be sufficiently large, it is possible to "fix" the triangle inequality for this specific triple as follows:

$$d'(x,y) + d'(y,z) - d'(x,z) =$$

$$= C + d(x,y) + d(y,z) - d(x,z) > 0, \text{ if } C > d(x,z) - d(x,y) - d(y,z)$$

It is possible to come up with a set of local constants C (depending only on $x$ and $z$) to fix the triangle inequality for all distance triples in the data set. However, for an *arbitrary* non-metric distance, these local constants do not necessarily generalize to previously unseen objects.

Chen and Lian [71] claim that generalization is possible if the distribution of queries follows the distribution of data, but they do not try define this constraint formally. If they talk about a probabilistic interpretation, there can be only probabilistic guarantees. Deterministic constraints are not impossible, but they would certainly limit the class of non-metric distances for which the algorithm would be exact. In any case, we do not see how local constant embedding can define a truly exact search method for a generic non-metric space.

Consider again Eq. 2.7 and assume that data points $x$ and $y$ belong to the data set while $z$ is a previously unseen point. For an arbitrary premetric distance, there are virtually no restrictions on how we assign distance values to a pair of different data points $x$ and $y$ (as long as $d(x,y) = d(y,x) > 0$). Thus, we can manually design a non-metric distance, where $z$ would satisfy Eq. 2.7 and, consequently, violate the triangle inequality. For example, we can choose the distance function that satisfies $d(y,z) = 1$ and $d(x,z) = 2 + d(x,y)$.

while pivoting approaches (§ 2.2.2.4) rely on mapping internally. Likewise, many LSH (§ 2.2.2.2) methods involve a mapping from the original space to the space of binary strings equipped with the Hamming distance. Yet, both hashing and pivoting methods are direct methods in the sense that they operate on the original data rather than on transformed one.

Furthermore, use of data mapping or distance transformation introduces an additional approximation, which complicates experimental evaluation. In many cases, we believe, replacing an expensive distance function with a cheaper proxy negatively affects quality ([12, 230]). For example, Norouzi et al. [230] experiment with replacing the Euclidean distance between 128-dimensional SIFT descriptors [191] with the Hamming distance between respective lossy 128-bit representations (of SIFT descriptions). For a one-million subset of SIFT descriptors, ensuring a 90% recall for the closest neighbor requires retrieving about 200 nearest neighbors in the space of binary codes. It is even harder to get accurate results for the set of one billion SIFT descriptors. If candidate records are selected among 1000 nearest neighbors in the space of binary codes, there is only a 75% chance to find a Euclidean nearest-neighbor. A similar outcome is observed in our own experiments with mapping based on random projections. Random projections are quite accurate, yet, according to Figure 2.10b, if we project original sparse vectors (having about 150 non-zero dimensions on average) to 128-dimensional dense vectors, ensuring that 10-NN search has 90% recall requires retrieving about 0.3% of data points. Because the data set in Figure 2.10b contains one million records, this is equivalent to to carrying out a 3000-NN search!

From these examples it can be seen that an accurate $k$-NN search via mapping may require carrying a $k'$-NN search in the target space with $k' \gg k$. However, judging by experience and results published in the literature, the larger is $k$ in the $k$-NN search, the harder is the search problem (in terms of efficiency and/or accuracy). Furthermore, nearly every approach to mapping and distance learning produces a symmetric distance function. In the case of *non*-symmetric distance, an implicit or explicit symmetrization associated with such mapping can be a source of additional inaccuracy, which can be quite substantial (see experimental results in § 2.3.2.4).

Although mapping may lead to quality deterioration, it is not universally true. For example, Norouzi et al. [229] showed that the $k$-NN classifier based on the Euclidean distance between vectorial image descriptors can be outperformed by the $k$-NN classifier based on the Hamming distance between respective lossy binary codes. Quite remarkably, mapping to a space of binary codes improves quality in this case. However, this may happen because the distance in the original space is not sufficiently effective. In particular, the Euclidean distance may produce suboptimal results because it does not account for the different scale of features [169]. If this is the case, it should have been possible to learn a more accurate distance in the original space (e.g., using one of the methods discussed in § 2.2.1). However, we do not know if the lossy binary representation would have outperformed such learned distance.

Due to the above-discussed difficulties, it is expedient to adopt a viewpoint that data mapping, distance learning and transformation are always applied as a separate step. We briefly review some common transformation approaches in § 2.2.1. Furthermore, after possibly transforming data, we apply a method that operates directly on data. Such methods, which we survey in § 2.2.2, exploit properties of the distance function, which hold for a broad class of spaces (see § 2.1.2 for a discussion of properties). For example, in the case of space-partitioning methods (§ 2.2.2.1) the search algorithm is a best-first traversal of space partitions. The traversal can rely on pruning using a lower-bound for the distance from the query to a neighbor in a region. In metric spaces,

computing such lower bounds is based on the reverse triangle inequality. In a more generic case, one can use $\mu$-defectiveness [1, 108].

The question of whether data and/or distance transformations are beneficial (e.g., can lead to improved accuracy in downstream applications) is mostly out of the scope of this thesis. However, our experiments in § 2.3.2.3 demonstrate that learning an accurate proxy distance for a non-metric similarity can be quite hard, especially when dimensionality is high.

### 2.2.1 Data Mapping, Distance Learning and Transformation

Approaches to data mapping, distance learning and transformation can be broadly classified into three categories:

- Distance and representation learning;
- Dimensionality reduction;
- Pivoting.

Metric learning methods (see [168] for a thorough survey) is a vast field, where most methods are tailored to achieve good performance in classification and learning-to-rank tasks by training the distance function from scratch. In this setting, the distance function is learned in such a fashion that the distance between data points from the same class tends to be small, while the distance between data points from different classes tends to be large. A common approach is to require that the latter distance is strictly larger than the former. In the case of the learning-to-rank task, we need to ensure that relevant data points are ranked higher than non-relevant ones.

One popular approach of doing so is to learn a distance in the form:

$$d(x, y) = \sqrt{(x - y)^{\mathrm{T}} A (x - y)}, \tag{2.8}$$

where $A$ is a symmetric, positive-semidefinite matrix. This method is known as as "Mahalanobis metric learning" although as pointed out by Kulis the learned metric is not necessarily a Mahalanobis distance [168].

Furthermore, Kulis notes that this problem can be seen as learning a global *linear* transformation of the data [168]. Indeed, because $A$ is a positive semi-definite matrix it can be factorized as $A = G^{\mathrm{T}}G$ using Cholesky decomposition [74]. Then, the distance function in Eq. (2.8) can be re-written as:

$$d(x, y) = \sqrt{(x - y)^{\mathrm{T}} A (x - y)} = \sqrt{(x - y)^{\mathrm{T}} G^{\mathrm{T}} G (x - y)} = \sqrt{||G(x - y)||_2} \tag{2.9}$$

Also note that computing the Mahalanobis distance at query time using Eq. 2.8 can be impractical, because the cost is quadratic in the number of vector dimensions. In contrast, if we precompute values $Gx_i$ for all data set points at index time, computing the distance between a query and a data point is much cheaper. Ignoring a query transformation time—which is negligibly small for a realistically large data set—the cost is only linear in the number of vector dimensions.

A popular approach to learning a *non*-linear transformation involves multi-layer neural networks [75, 260]. Another common method to introduce non-linearity is through kernelization (see [168], §3.1) Note that kernelization can be seen as a form of implicit data transformation, because a kernel defines a mapping to (possibly infinite-dimensional) space with an inner product.

Perhaps, the most common objective in Mahalanobis distance learning is to ensure that distances between points in one class are smaller than distances between points from different classes. Furthermore, we want these distances to be separated by a large margin [168]. This results in a convex optimization problem with a hinge loss [312], which is quite similar to support vector machines. Other standard machine learning techniques were employed as well. In particular, Xiong et al. [317] train random forests using feature sets composed of the scalar $||x - y||_2$ and the vector $(x + y)/2$, where $x$ and $y$ are a pair points from a training set. Athitsos et al. [12] create multiple one-dimensional projections using FastMap [107], each of which is a weak classifier. They combine weak classifiers via AdaBoost [263] with a goal to preserve the ranking induced by the original distance. In § 2.3.2.3, we experiment with learning a Mahalanobis *metric* distance in the form specified by Eq. (2.8) and (2.9), as well as with learning a *non*-metric distance with the help of random forests [317].

Another class of metric learning methods uses a set of hash functions in the form:

$$h_i(x) = \text{sign}\left(\langle w_i, x\rangle + b\right) \tag{2.10}$$

These hash functions define a mapping from $L_2$ (or cosine) to compact binary codes, which are compared using the Hamming distance. The mapping in Eq. (2.10) is inspired by locality-sensitive hashing (LSH) methods (§ 2.2.2.2) for $L_2$, where vectors $w_i$ are created by sampling elements from a Gaussian distribution (elements are sampled independently) and $b$ is sampled from a uniform distribution [84]. However, unlike LSH, vectors $w_i$ are learned from data. The learning objective is to minimize a classification/re-ranking error of a $k$-NN based classification/retrieval module, which relies on the Hamming distance between compact binary codes (see, e.g., [228, 229]). Vectors $w_i$ in Eq. (2.10) can also be learned to minimize the reconstruction error, i.e., the objective is to ensure that the Hamming distance between binary codes (multiplied by a constant) is an accurate estimate for the original $L_2$ distance (see, e.g., [169]).

Instead of learning a distance from scratch, it may be beneficial to "adjust" the original one. For example, Skopal [271] proposed a method called *TriGen* which "metrizes" a *symmetric* distance function using a monotonic concave transformation (we consider TriGen to be a special form of metric learning). In § 2.3.3, we discuss this method in more details. Due to the monotonicity of such a transformation, the $k$-NN search using the modified distance produces the same result as the $k$-NN search using the original distance. However, it is possible to learn such a transformation that the modified distance obeys the triangle inequality with a given degree of approximation. This, in turn, allows us to apply a space-partitioning algorithm designed for metric spaces (§ 2.2.2.1).

Note that many important algorithms cannot not benefit from the mapping created via TriGen. First, some of the most efficient LSH algorithms require the structure of the vector space, which cannot be obtained via TriGen. Second, most proximity-graph based algorithms (§ 2.2.2.3) rely only on point rankings rather than on raw distance values. For example, given points $x$, $y$, $z$ they may need to verify if $d(x, y) < d(x, z)$. Because, monotonic distance transformations do not change rankings, a monotonic mapping does not affect effectiveness of a proximity-graph based method.

Instead of dealing directly with original data one may resort to mapping data to a lower-dimensional space using a *dimensionality reduction* algorithm. The most straightforward way to use a lower-dimensional mapping is to carry out a $k$-NN search in the lower dimensional space

using a larger value of $k$. This step provides a list of candidates for subsequent verification (using the distance in the original space). Efficient searching in the lower-dimensional space can be done by using any of the existing technique. Additionally, dimensionality reduction may be "embedded" into the indexing and search algorithms as a tool that helps organizing the search space. This is an approach employed in, e.g., LSH methods (see § 2.2.2.2) and random projection trees [83].

Principal component analysis (PCA) is an early approach to dimensionality reduction in the Euclidean space [116]. PCA finds several directions of greatest variance and projects data onto the corresponding subspace. While it is possible to compute PCA for large data sets [128], it is still an expensive procedure. Random projections is a less expensive approach which nevertheless preserves inner products and distance values with high probability and good accuracy [37]. Unlike PCA, random projections projection directions are chosen randomly. It is not hard to see that the respective dimensionality reduction procedure is equivalent to multiplying a (column) vector of data points by a randomly generated matrix with unit norm rows. The classic approach to matrix creation consists in sampling elements from the standard normal distribution with subsequent orthonormalization of rows, but simpler approaches work as well [82].

Both PCA and random projections are linear transforms. A number of non-linear methods were also proposed, in particular, based on neural network autoencoders [75, 133, 256]. Most dimensionality reduction approaches are designed only for vector spaces: There exist also purely distance-based generalizations, but one should not expect them to preserve the original distance.

Faloutsos and Lin proposed one of the first distance-based projection algorithms FastMap [107]. FastMap projects data points on a line passing through two randomly selected data set points [107]. FastMap can be seen as a variant of random projections. Therefore, it tends to preserve distances well.[8] Faloutsos and Lin make a clever observation that computing the projection does not require knowledge of vector coordinates. One has to know only three pairwise distances [107]. Thus FastMap can be used in non-vector spaces including non-metric ones, albeit without any expectations with respect to preservation of distances.

Another simple approach to data transformation consists in representing data points by respective pivot rankings. To this end, a set of reference points, i.e., pivots, is created in advance (e.g., by sampling pivots from the data set). Then, for each data point, we rank pivots in the order of increasing distance from the point. Pivot rankings are integer-valued vectors, which can be compared using, e.g., the Euclidean distance. The idea gave rise to a family of the so-called permutation approaches, we survey in § 2.2.2.4. In addition to complete rankings, one can use shortened rankings or represent the data point by an *unordered* set of its closest pivots. Note that there are no guarantees of distance preservation. However, empirically, the order of nearest neighbors in the transformed space may be preserved to a good degree (see § 2.3.1.4), which is sometimes sufficient for reasonably effective $k$-NN search.

### 2.2.2 Related Work

To highlight relationships among different search methods as well as to make it easier to understand algorithmic details, we first briefly review key properties and building blocks of the search methods.

[8]Note, however, that we are not aware of any theoretical justification. Unlike classic random projections where the direction is selected fully randomly, a direction in FastMap is defined by existing data points. Because of this dependence on specific data distribution, an ability of FastMap to preserve the distance is hard to analyze analytically.

To this end, we arrange methods along three dimensions. As shown in Figure 2.1, we classify methods by the type of access to underlying data, as well as by the indexing and search strategy. Note that these classification dimensions are not independent. In particular, a choice of a search strategy is largely determined by an indexing approach. After briefly discussing the underlying principles of search methods, we review several classes of algorithms in more detail.

With respect to access type, we distinguish between *data structure aware* methods, *distance-based* methods, and *hybrid distance based* methods. Hybrid distance-based methods may exploit data set structure to, e.g., select better reference points, but still rely crucially on a "data blind" distance-based algorithm.

The most common type of data structure aware methods are vector space methods. Apparently the first index-based search method for vector spaces codenamed KD-tree was proposed by Jon Louis Bentley [30]. In KD-tree, the data set is recursively divided by hyperplanes orthogonal to one of the dimension axes. The indexing algorithm selects a dimension according to some rule (e.g., randomly) and finds an orthogonal hyperplane that divides the data set into two subsets of nearly equal sizes.[9] The process continues recursively till the number of points in a partition becomes smaller than a threshold (a *bucket size*). The resulting hierarchy of data points is represented by a binary tree with leaves keeping sufficiently small subsets of data points called buckets.

In the case of KD-tree the underlying data structure is directly exposed to the indexing and the search algorithms, which exploit the vectorial structure of the data. One may argue that access to the underlying data structure is a necessary condition for devising the most efficient and accurate search methods. However, algorithms relying on specific properties of data may have limited applicability. In particular, KD-tree is not applicable to metric spaces that lack a vector space structure.

A metric-space generalization of the KD-tree also employs a decision surface to recursively divide the space into nearly equal partitions [234, 293, 323]. Unlike KD-tree, it splits the data set using hyperspheres rather than hyperplanes. The center of a hypersphere—a *pivot*—is usually a randomly selected data set point. The radius of the hypersphere is a median of distance values between data set points and the pivot. Thus, similar, to the case of KD-tree, the partition boundary divides the data set into two subsets of nearly equal size. The resulting data structure is a binary tree called the vantage-point tree (VP-tree), the ball tree, or the metric tree [234, 293, 323].

VP-tree is a *distance-based* method. Distance-based methods do not have a direct access to the data structure and are, therefore, more generic. Specifically, distance-based methods treat data points as unstructured objects, together with a black-box distance function. An indexing and/or a search algorithm of a distance-based method exploits only values of mutual object distances. A purely distance-based method can compute distances only between data set points as well as between data set points and queries. This restriction may significantly affect performance. For example, if the data set has easily identifiable clusters of small radii, it is easy to verify if queries and data points belong to the same cluster by computing distances to cluster centers. However, cluster centers are not guaranteed to be present in the data set (consider, again, an example of points evenly distributed on a sphere).

[9]This may not be possible in a degenerate case, when all (or most) data points have equal values along the selected dimension. In such a case, the algorithm may chose a more appropriate dimension to split the data.

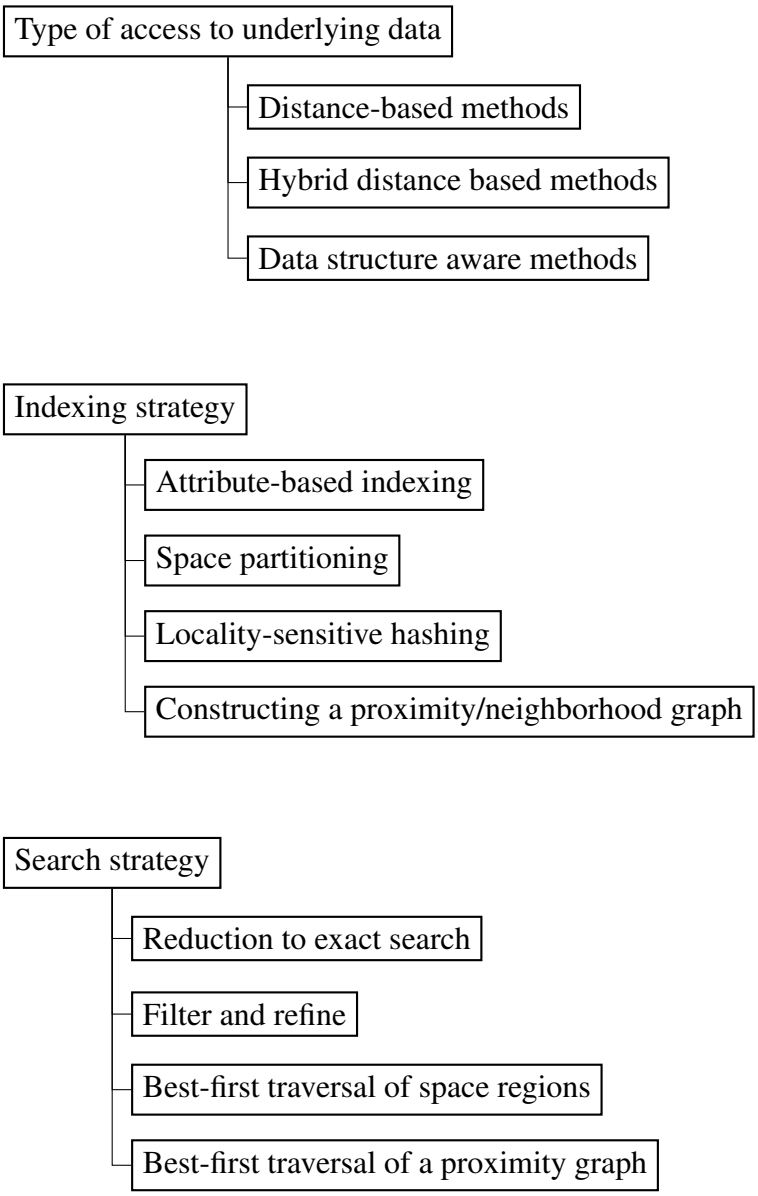



Figure 2.1: Key properties and building blocks of search Methods

For this reason, a variety of *hybrid distance-based* approaches were proposed. Hybrid distance-based approaches combine the flexibility of distance-based methods with knowledge of data set structure possessed by, e.g., vector spaced methods. One possible use of knowledge is to construct reference points, i.e., pivots that are not present in the data set. For example, in image search engines, it is common to create reference points using k-means [15, 149, 215, 270]. In § 3.1.2.2, we describe a simple method to construct reference points for sparse high-dimensional vector spaces.

At a high-level, indexing strategies can be divided into the following classes:

- Attribute-based indexing;
- Space partitioning;
- Locality-sensitive hashing (LSH);
- Constructing a proximity/neighborhood graph.

Attribute-based indexing requires access to underlying structure of data and relies on compositionality of data. Compositionality means that data points can be represented by a set of key attributes, which are indicative of data point proximity. For example, in the case of sparse vectors, a set of attributes can include indices of non-zero vector elements. Attributes would typically be indexed using a simple data structure for exact search, e.g., an inverted file.

Space partitioning indexing strategy (see § 2.2.2.1) consists in dividing the space into possibly overlapping bounding regions. Usually we use a hierarchy of partitions by dividing the space recursively. In addition, rather than building a single hierarchy, it can be beneficial to build several hierarchies, e.g., using random initialization parameters.

Hashing involves grouping data points with equal hash codes. The hashing is supposed to be *locality-sensitive*, i.e., close data points tend to have the same hash codes, while distant points are likely to have different ones (see § 2.2.2.2). Yet, grouping by hash values does not define nice bounding regions as data points with equal hash values are typically scattered all over the space. Because a probability that two close points have the same hash value (we call this event a *collision*) is small, multiple hash tables are built using *different* hash functions. For good performance, hash functions should be selected in such a way that given a pair of close points a collision with respect to one hash function is (approximately) independent of collisions with respect to other hash functions.

Building a *proximity graph* (also known as a *neighborhood graph*) is, perhaps, the most unusual indexing strategy, which creates a data set graph, where data points are nodes and edges connect close data points. Proximity graphs are also quite unique in the search strategy, which consists in a best-first traversal of the neighborhood graph. Somewhat confusingly, this strategy is sometimes called hill climbing, whereas in truth the graph traversal is a descent. The search is directed towards nodes with small distances to the query in the sense that nodes with smaller distances are explored first.

Space-partitioning algorithms also rely on best-first traversal of a graph, but this traversal is quite different in nature. In a proximity graph, the graph consists of individual data points, whereas any two sufficiently close points can be connected. In contrast, in a space-partitioning method each node in a graph represents a region that typically covers multiple points. Furthermore, nodes/regions are typically organized into a tree (or a forest), where a parent region contains all the points from child regions. Because of this, the search algorithm may prune a region by

estimating a lower bound from the query to a nearest neighbor inside this region.

Another common search strategy consists in reducing approximate search to exact one. For example, LSH (see § 2.2.2.2) places data points with same hash codes into the same bucket. Because close data points tend to have equal hash values, these points can be discovered by checking buckets with the same hash code as the query. Attribute-based indexing may also allow to reduce approximate searching to exact one. If close data points tend to share at least some common attributes, nearest-neighbor queries can be answered by retrieving data points with common attributes and comparing them directly against the query. An efficient lookup by attributes can be supported by creating an inverted file of point attributes.

Both LSH methods and attribute-based retrieval can be seen as *filter-and-refine* methods, which rely on some simple approach to generate a list of potential matches, called *candidates*. These candidates are then compared directly to the query using the original distance function. In the case of LSH and attribute-based retrieval, candidates are generated via exact search with a help of a simple index. A viable alternative to exact indexing is a brute-force search using a cheap approximation of the distance function.

A vector-approximation file (VA-file) [311] is one famous example of a simple filter-and-refine method for vector spaces, where candidates are generated using brute-force search. VA-file uses a simple form of quantization. For each vector dimension, the range of values is divided into a few regions, whose quantity is a power of 2, which, in turn, allows us to obtain a compact representation of each dimension in a form a small bit vector. The concatenation of these bit vectors defines a bounding rectangle containing the original data point.

The search algorithm iterates over coarse vector representations and compares them to the coarse representation of the query. Each comparison produces a lower bound for the distance between the query and a data point. If this bound is smaller than the distance to the $k$-th closest points encountered so far, then the vector is compared directly to the query by computing the distance. Comparing coarse representations of vectors permits establishing an upper and a lower bound for a variety of decomposable distance functions, including the Euclidean distance [311] and KL-divergence [59, Chapter 7],[327].

In the following subsections, we provide a more detailed review of search methods, which we divide into the following categories:

- Space-partitioning methods (§ 2.2.2.1);
- Locality-sensitive hashing (LSH) methods (§ 2.2.2.2);
- Proximity graph (neighborhood graph) methods (§ 2.2.2.3);
- Permutation methods (§ 2.2.2.4).

Note that permutations methods can be seen as filter-and-refine methods—sometimes combined with space partitioning. However, they have specifics that deserves to be discussed separately.

#### 2.2.2.1 Space-partitioning Methods

Space-partitioning search methods belong to a broad class of divide-and-conquer approaches. KD-tree [30] may be the first formally published divide-and-conquer search method for vector spaces, but divide-and-conquer search algorithms for real-world geometric problems clearly predate the computer era. A simple example is routing of correspondence where a mail is first

relayed to a destination country, then to a local post office near a destination city or village, and, finally, to the recipient, who lives in the vicinity of the local post office.

A space-partitioning approach creates divisions of the search using finite or infinite solid objects, which we call *bounding regions*. It is common to repeat the process recursively, which results in division of the data set into into progressively smaller subsets of data. The recursion stops when the size of a subset becomes smaller than a threshold (a bucket size). Bucket data points are stored using a simple data structure such as a list or an array. The resulting hierarchy of data points is best represented as a tree, whose leaves are buckets. A flat hierarchy is a special case (see, e.g., the list of clusters method [64]), when we partition the data set once, without re-dividing partitions recursively.

Space-partitioning algorithms differ mostly in types of bounding regions. Vector spaces permit great flexibility of choice. For example, KD-trees employ hyperplanes to divide the space into two half-spaces. In the classic KD-tree due to Bentley [30] hyperplanes are orthogonal to one of the coordinate axes. In a KD-tree variant called a *random projection tree*, the direction of the hyperplane is chosen randomly. R-trees [126] and its variants rely on hyper-rectangles.

In the case of metric-space methods, there are two major partitioning approaches. In the first approach, bounding regions are Voronoi cells [299] induced by a set of (e.g., randomly selected) pivots. A Voronoi cell is a region surrounding a pivot and containing points for which this pivot is the closest. A Geometric Near-neighbor Access Tree (GNAT) [48] is an example of the search method that uses recursive Voronoi partitioning. This approach was dubbed as a *compact partitioning* by Chávez et al. [65], because points in the same partition tend to be located close to each other.

There are two main approaches to selecting pivots in compact partitioning methods: random selection and carrying out a cluster analysis. Clustering methods differ mostly in the choice of the clustering algorithm. In vector spaces, a popular approach is k-means [15, 149, 215]. In spaces that lack vector structure, a reasonable alternative is a k-medoids algorithm due to Kaufman and Rousseeuw [158]. In particular, Goh et al. [121] employed a more efficient variant of k-medoids called CLARANS [226]. Leibe et al. [177] experimented with a variant of an agglomerative clustering. However, agglomerative clustering is hardly practical for large collections, because indexing time is quadratic in the number of data points. Likewise, both k-means and k-medoids clustering are expensive and simpler approaches were proposed. For example, Chávez and Gonzalo Navarro [64] explored a family of simple clustering algorithms similar to canopy clustering [202], which use a greedy assignment. Greedy approaches are cheap, but not necessarily accurate.

It is noteworthy that in image retrieval—in addition to organizing the search space—clustering is also widely used for data compression. In the classic approach—known as vector quantization—a data point is basically replaced by the closest cluster center. Thus, instead of memorizing the original data set, we only keep a relatively small set of clusters—called a *codebook*—and an identifier of the closest point from the codebook (which requires many fewer bits compared to the floating-point vector representation). Constructing codebooks using k-means is expensive: Tong et al. propose a method to compose a codebook by sampling images' keypoints [286].

As a side note, vector quantization via k-means dates back to 1957, but it was not formally published by its author Stuart P. Lloyd until 1982 [190]. A number of variants and extensions of this algorithm are used in image retrieval: See a recent survey for more details [329].

The second partitioning approach divides the space with hypersphere centered at (e.g., ran-

domly selected) pivots. For example, in each node of VP-tree [234, 293, 323] there is a single hypersphere whose radius is a median distance to a hypersphere center (the center is usually a randomly selected data point). In a binary multi-vantage point tree, there are two pivots in each node that divide the space into four parts using spherical cuts [46].

A search algorithm of an *exact* space-partitioning method proceeds recursively starting from the root of the tree. At each step the crucial step is to verify if a partition contains a data point within a given maximum allowed distance $r$ to the query. Because we know the shape and location of the bounding object, we can compute a lower bound or an estimate thereof. If the lower bound is larger than the maximum allowed distance, the partition cannot contain a point of interest. In this case the partition is *pruned*. Otherwise, we need to examine the partition recursively. In the case of compact space-partitioning methods, an additional pruning rule—based on the properties of the Voronoi diagram—can be used. Because, an answer belongs to the partition with the closest center, we can discard a region $A$ for which the lower bound to the partition center is higher than the upper bound to the center of another region $B$.

In the case of range search, the maximum allowed distance is specified by the user. In the case of $k$-NN search, there is no maximum distance specified beforehand and the search process is simulated via a priority range search with a shrinking radius, which is equal to the distance between the query to the $k$-th closest data point discovered by the search procedure. An initial radius value is set to infinity. This priority search process is a best-first recursive traversal (with pruning), where partitions with smaller lower bounds are visited first [9]. Whenever the search process arrives at a leaf (or the query is compared with pivots), we compute actual distances to data points and decrease the radius of the search if the distance from the query to the $k$-th closest data point becomes smaller than the current radius value.

Clearly, an ability to compute the lower bound from the query to a point inside a bounding region efficiently and accurately is essential to performance of the search algorithm. Doing so is straightforward for hyper-rectangular bounding regions in the form $\{m_i \leq x_i \leq M_i\}$ and the Euclidean distance. If bounding regions are hyper-rectangular, this may also be possible for a variety of decomposable distance functions including some Bregman divergences [327],[59, Chapter 7].

In the case of metric-space methods, bounding regions are either Voronoi cells or space areas produced by spherical cuts. Such regions can be quite complex. However, lower bounds can be obtained using the triangle inequality. Assume, for example, that we have a cluster with the center $c_i$ and the radius $R_i$. All data points belonging to the cluster are located inside the cluster ball $B(c_i, R_i)$. If the cluster contains a data point within distance $r$ from the query $q$, then this ball should intersect with the query ball $B(q, r)$ (recall that the $k$-NN search is simulated via the range query with a shrinking radius). According to the triangle inequality, this is only possible if $d(q, c_i) - r \leq R_i$. In contrast, if $d(q, c_i) - r > R_i$, the query balls does not intersect the cluster so that the latter can be safely ignored.

There is a great deal of metric-space search methods, which rely on checking if the query ball intersects with one of the bounding balls covering a partition. Extending this check to non-metric spaces is highly non-trivial because, as we noted previously, non-metric spaces lack common and easily identifiable structural properties such as the triangle inequality. Note, however, that there has been great progress in recent years in devising exact indexing algorithms for a class of non-metric distances called Bregman divergences (see Exposition 2.3). In particular, Cayton [58]

proposed to divide a Bregman-divergence space via a hierarchy of Bregman balls (a Bregman ball is simply a ball when the distance belongs to the class of Bregman divergences). To this end, the data set is recursively divided into two subsets using k-means with two centers. Each subset is represented by a Bregman ball that covers every cluster point. At query time, a clever algorithm is used to verify if the query ball intersects with Bregman balls constructed during indexing. This algorithm estimates the distance from the query to a Bregman using a binary search on the line between the ball center and the query.

Using properties of Bregman divergences, Zhang et al. [327] obtained a generic formula to compute lower and upper bounds from the query to a bounding rectangle. They also proposed improved bounds for *decomposable* Bregman divergences, which can be found analytically for KL-divergence and the Itakura-Saito distance (see Table 2.3). For other decomposable divergences, though, obtaining the bounds may require a binary search. Another limitation of Zhang el al. [327]'s approach is that it is designed for left queries only. Supporting right queries may be possible, but this requires obtaining new derivations of upper lower bounds, which involves a non-trivial analytical effort. Likewise Cayton [58]'s algorithm works for left queries only. Cayton notices that right queries using $d(x, y)$ can be answered using the divergence generated from a convex conjugate $f^*(y)$ and data points replaced with $\nabla f(x)$, where $f$ is the convex function generating the original divergence $d(x, y)$ (See Eq. 2.4) and the convex conjugate $f(x)$ is defined as $f^*(y) = \sup_x\{\langle x, y\rangle - f(x)\}$. It is not always possible to find the convex conjugate analytically, which likely limits the applicability of such a reduction.

Exact space-partitioning methods are inefficient for high-dimensional data sets [34, 58, 311]. Resorting to approximate searching permits to trade off some accuracy for improved speed. There are two straightforward approaches to turn an exact space-partitioning method into an approximate one: early termination and aggressive pruning. It is not clear who first introduced early termination. However, it seems to be a folklore approach that has been known for a while. In particular, early termination is employed in the venerable ANN library [213]. The same seems to be true for aggressive pruning. For some early mentions of using it, please, see the paper by Arya et al. [11] and Yianilos [324].

In the case of early termination, the search algorithm simply stops after visiting a specified number of leaves (a method's parameter). A powerful modification of this approach engages a forest of space-partitioning trees rather than a single tree. In particular, Silpa-Anan and Richard I. Hartley [269] proposed to build multiple KD-trees and search them using a single priority queue. Aggressive pruning means that we heuristically modify the pruning rule so that tree branches can be discarded even when the lower bound from the query to a bounding region does not exceed the distance to the $k$-th closest point encountered so far. One common approach is to multiply the lower bound estimate by $1 + \epsilon$ for some $\epsilon > 0$. This modifications makes pruning unsafe (i.e., we may miss some true neighbors). Despite being unsafe, these pruning algorithm still has a (somewhat weaker) quality guarantee. Specifically, one can be sure that the distance to a found data point is at most $1 + \epsilon$ times larger than the distance to a true nearest neighbor.

In the domain of metric-space methods, Chávez and Navarro [63] propose a variant of the aggressive pruning, which they call *stretching of the triangle inequality*. In § 2.2.3, we describe a novel modification of this approach that combines aggressive pruning with adaptation to a non-metric distance function. While Chávez and Navarro [63] suggest to "stretch" the triangle inequality, Skopal [271] propose to stretch the distance itself. The indexing algorithm—called

**Exposition 2.7: Locality sensitive functions for the Hamming distance**

In the case of the Hamming distance and bit vectors a locality-sensitive function can be generated as follows. First, one selects $l < m$ integer values $k_i$ from the set $\{1, 2, \ldots, m\}$, where $m$ is the dimensionality of data. Assume that $k_i$ are already sorted. Then the hash function takes an $m$-dimensional bit vector $x = x_1 x_2 \ldots x_m$ and produces an $l$-dimensional bit vector $x_{k_1} x_{k_2} \ldots x_{k_l}$. Now imagine that vectors $x$ and $y$ differ in 10 bits and the hash function is obtained by sampling one tenth of all bits (i.e., $l = m/10$). Then, the expected difference between two hash functions values would be one (under a common assumptions that hash functions are selected randomly, while data points are fixed [301]).

*TriGen*—finds an appropriate monotonic transformation of the distance such that the transformed distance obey the triangle inequality with a given degree of approximation. Then, a standard metric-space indexing algorithm can be applied. Due to the monotonicity of the transformation the $k$-NN search using the modified distance produces the same result as the $k$-NN search using the original distance. In § 2.3.3, we discuss this method in more details.

#### 2.2.2.2 Locality-Sensitive Hashing Methods

A well-known approximate $k$-NN search method is the locality-sensitive hashing (LSH). It was pioneered by Broder [49], Indyk and Motwani [142], Kushilevitz et al. [172], with additional important contributions coming from Moses Charikar (see, e.g., [62]). A recent survey on the topic was written by Wang et al. [305]. The underlying idea is to construct a family of functions that tend to have the same hash values for close points and different hash values for distant points. It is a probabilistic method in which the probability of having the same hash value is a monotonically decreasing function of the distance between two points (that we compare). A hash function that possesses this property is called *locality sensitive*.

A basic LSH-based search algorithm creates a hash table during indexing time. At query time, it computes the hash value of the query and retrieves data points with the same hash values. These data points are candidate entries, which are further compared directly against the query. A probability to retrieve a nearest neighbor using a single hash function is usually low. Consequently, many hash tables are to be created during indexing. Turns out that achieving high recall requires too many tables and, consequently, impractically large indices.

To overcome this problem, Lv et al. proposed a multi-probe version of the LSH, which can query multiple buckets of the same hash table [192]. To this end, the search algorithm finds "neighboring" hash values that are likely to be produced by near data points. Consider, for example, a data set of bit vectors with the Hamming distance (The Hamming distance between two bit vectors is equal to the number of mismatching elements: $d(x, y) = \sum_{i=1}^{m} [x_i \neq y_i]$). Bit sampling is a basic approach to construct an LSH family for such data [142]. A bit-sampling function is defined by which bits of the original bit vector it retains. The remaining bits are discarded. If the number of sampled bits is much smaller than the number of bits in original vectors, than the Hamming distance between hash values is usually much smaller than the distance between original bit strings (see Exposition 2.7 for formal details). This observation suggests a multi-probing

scheme where neighboring hash values are the ones that differ in a small number of bits from the hash value of the query.

Designing good locality-sensitive functions proved to be a challenging task. Families of LSH functions are devised for the Hamming distance [142], $L_p$ metrics [84] (see Table 2.3), the cosine similarity [5, 62], as well as for the kernelized similarity functions [170]. The latter approach is designed for symmetric positive definite kernels and, consequently, works only with symmetric similarity functions. Mu and Yan propose to embed data into a Krěin space equipped with an indefinite inner product [214]. Their approach does not require an explicit kernelization and works with similarity functions represented by indefinite similarity kernels. However, the similarity should still be symmetric. For an arbitrary similarity functions, we know only one data-driven approach (based on sampling) [13]. Yet, neither this approach or the approach of Mu and Yan [214] are well tested and/or widely adopted yet.

LSH is a well regarded and award winning technique: In 2012, Andrei Broder, Moses Charikar, and Piotr Indyk received *Paris Kanellakis Theory And Practice Award* for "...their groundbreaking work on Locality-Sensitive Hashing that has had great impact in many fields of computer science including computer vision, databases, information retrieval, machine learning, and signal processing."[10]

Note that LSH methods are quite efficient. For example, Datar et al. [84] showed that an LSH algorithm was 40 times faster than an approximate variant of KD-tree from the ANN library [213]. However, carefully tuned and engineered tree-based algorithms can outperform at least some LSH methods (or match their performance) on a variety of realistic data sets. For example, Muja and Lowe demonstrated that LSH can be outstripped by a hierarchical k-means tree or by forest of random projection trees [215]. In turn, we demonstrated that a multi-probe LSH due to Dong et al. [96] may be outperformed even by a single VP-tree [43, 220].

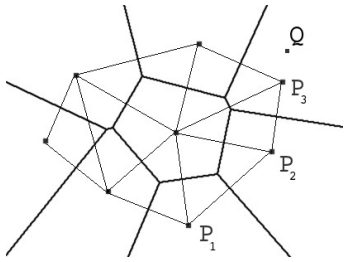



Figure 2.2: An example of the Voronoi diagram/partitioning and its respective Delaunay graph. Thick lines define the boundaries of Voronoi regions, while thin lines denote edges of the Delaunay graph.

Older LSH approaches for vector spaces construct hash functions using random hyperplanes. However, as it is recently demonstrated by Andoni et al. [5], a novel family of hash functions, namely a cross-polytope LSH, can be several times more efficient than hyperplane-based methods. Andoni et al.'s work concentrates on the case of Euclidean distance on the unit sphere, which corresponds to the angular distance and cosine similarity.

We are not aware of any comprehensive comparison of this latest method against other approaches. Yet, in one public evaluation, the cross-polytope LSH is compared against tree based methods on two data sets.[11] As of May 2016, the cross-polytope LSH is faster than both our

[10] See, e.g., `http://awards.acm.org/award_winners/indyk_2720472.cfm`

[11] See `https://github.com/erikbern/ann-benchmarks` for the description of the benchmark

VP-tree and FLANN [215]. However, when compared against an optimized package Annoy[12], which relies on a forest of hierarchical k-means trees ($k = 2$), the cross-polytope is up to $2\times$ faster on the Glove data set and slightly slower on the SIFT data set.

#### 2.2.2.3 Proximity/Neighborhood Graph Methods

A *proximity graph*—also known as a *neighborhood graph*—is a data structure in which data points are associated with graph nodes and edges connect sufficiently close nodes. These graphs have a long history and were rediscovered multiple times by different research groups. Toussaint published a pioneering paper where he introduced proximity graphs on the plane [287]. Arya and Mount published apparently the first study where proximity graphs were applied to the problem of $k$-NN search. The graph of Toussaint—dubbed a *relative neighborhood* graph—is undirected. A pair of points are neighbors, if there is no other point in the data set that has a smaller distance to both points than the distance between points in the pair.

There are multiple ways to define neighbor nodes, but most of the recently published papers employ $k$-NN graphs. In the *exact* $k$-NN graph, each point is connected to its $k$ true nearest neighbors. In an *approximate* $k$-NN graph, each point is connected to only some of its $k$ nearest neighbors. An exact or approximate $k$-NN graph is often defined as a directed graph [98, 127], where the edges go from a vertex to its neighbors. Yet, there are variants where the $k$-NN graph is undirected [196].

Why are these graphs useful for searching? One simple explanation is that similarity is transitive to a certain degree: the closest neighbor of the closest neighbor is a close neighbor as well. A more formal explanation involves properties of the Voronoi diagram [299], induced by data set points, and the corresponding Delaunay graph, which the $k$-NN graph approximates. Recall that the Voronoi diagram is division of space where each data point $X$ is surrounded by region called a Voronoi cell. For any point in the cell, $X$ is the nearest neighbor (see Figure 2.2).

A graph representation of the neighbor structure for the Voronoi diagram is called a Delaunay graph. In this graph the points are connected only if they have bordering Voronoi regions. Given the Delaunay graph, finding the nearest neighbor can be achieved via a greedy algorithm. This algorithm starts at an arbitrary node and recursively transitions to the neighbor point (by following the graph edge) that is closest to the query [262]. The algorithm stops when the current point $X$ is closer to the query than any of the $X$'s neighbors.

Consider an example in Figure 2.2, where the nearest neighbor search with the query $Q$ starts at $P_1$. The point $P_1$ has three neighbors among which $P_2$ is the closest to $Q$. So, the greedy search moves from $P_1$ to $P_2$. After transitioning to $P_2$, the algorithm again chooses the next neighboring point closest to $Q$, i.e., $P_3$. The search terminates at $P_3$, because $P_3$ is closer to $Q$ than any of the $P_3$'s neighbors.

While theoretically appealing, exact Delaunay graphs are rarely used in practice, because there is no algorithm to construct these graphs in generic spaces and/or high dimensions. In particular, in generic metric spaces, the only usable superset of the Delaunay graph is a complete graph [221]. The complete graph has prohibitive storage cost $O(n^2)$, where $n$ is the number of data set points. In addition, each transition may require up to $n$ distance computations to find the neighbor point

[12] `https://github.com/spotify/annoy`

closest to the query. As a result, most practical methods approximate the Delaunay graph by an exact or approximate $k$-NN graph.

The $k$-NN graph can be computed by brute-force computation of the distance between all pairs of data points [242, 267, 304]. The computational cost is $O(n^2)$ and, thus, this approach is feasible only for small data sets. In metric spaces, $k$ closest neighbors can be efficiently found using an exact retrieval approach. Due to the curse of dimensionality, this works well only for intrinsically low-dimensional data sets [237].

For large data sets, various approximations are used. First, one can use agglomerative approaches where the data set is divided into subsets and the exact algorithm is used to construct the graph for each subset. Then, these graphs are are combined via a sequence of merge operation [298, 302]. Second, it is possible to build the graph incrementally while using the partially constructed graph to obtain nearest neighbors [7, 138, 196]. Trad et al. [288] proposed to obtain nearest neighbors using a novel LSH method based on the Random Maximum Margin Hashing. One interesting algorithm is NN-descent.[98] It starts by picking a random approximation for the direct and reverse nearest neighbors. Then, it iteratively improves the approximation by re-examining neighbors of objects' neighbors. The graph created by NN-descent is directed. However, both the author of NN-descent Wei Dong (See the code of kgraph[95]) as well as Li et al. [184] discovered that adding reverse links improved performance. There are also several domain-specific algorithms, which we do not mention here, because they are not sufficiently generic.

A retrieval algorithm is a greedy search procedure starting from seed nodes and exploring neighbor points recursively. Several heuristics related to the search procedure were proposed, in particular, for selection of seed nodes and the exploration strategy. Most importantly, a search can start from either a randomly selected node [127, 196], or from a node found by a low accuracy method, e.g., by a KD-tree [10]. The termination condition can be defined in terms of either the immediate [127] or extended neighborhood [196, 267].

Wang et al. proposed an interesting hybrid distance-based approach in which the neighborhood graph is expanded with additional nexus points [303]. The underlying idea is that the search can benefit by connecting data points via shorter routes. However, such routes may need to pass through areas of the space which do not contain data set points in their vicinity. Thus, if we used only data set points as graph nodes, there would be no nodes in such areas. The proposed workaround is to generate additional nexus points via a Cartesian product quantization, which is based on sub-space k-means clustering (see [303] for details).

Some performance is lost because we construct an approximate graph rather than an exact one. However, even exact neighborhood graph can be suboptimal. In particular, using larger neighborhoods helps increasing a probability of a successful traversal. However, it entails longer retrieval times due to processing more neighbors. At the same time, not all neighbors are equally useful. For example, it does not make much sense to keep neighbors that are close to each other [130, 184]. To reduce redundancy various pruning methods were proposed [130, 184, 197].

In addition to selection of neighbors, the structure of the graph can be sometimes improved. First, many approaches build an undirected graph. However, it was observed that graph symmetrization (i.e., addition of reverse links) can substantially improve performance (see, e.g., [184]). Second, it may be beneficial to expand the neighborhood links with a hierarchical structure [136, 137, 197].

```
1: procedure UPDATETOPK(res, k, x, dist)                    ▷ res is max priority queue
2:                                                          ▷ passed by ref/pointer
3:     res.push( (dist, x) )                                ▷ This permits an obvious optimization,
4:     if res.size() > k then                               ▷ which we omit to simplify presentation.
5:         topK.pop()
6:     end if
7: end procedure
8:
9: procedure KNNSEARCH(data, q, k, ef)
10:    res ← ()                                             ▷ Empty max priority queue
11:    candQueue ← ()                                       ▷ Empty min priority queue
12:    closestDistQueue ← ()                                ▷ Empty max priority queue
13:    visited ← ()                                         ▷ Empty set (can have different implementations)
14:    curr ← 0
15:    visited.add(curr)
16:    currDist ← d(data[curr], q)
17:    candQueue.push( (curr, currdist) )
18:    closestDistQueue.push(currDist)
19:    updateTopK(res, k, data[curr], currDist)
20:    while not candQueue.isEmpty() do
21:        (curr, currDist) ← candQueue.pop()
22:        if currDist > closestDistQueue.top() then
23:            break                                        ▷ If the closest query element is farther than
24:                                                         ▷ the ef-th query neighbor, terminate search.
25:        end if
26:        for neighb in curr.neighbors do
27:            if not neighb ∈ visited then
28:                visited.add(neighb)
29:                neighbDist ← d(data[neighb], q)
30:                candQueue.push( (neighb, neighbDist) )
31:                closestDistQueue.push(neighbDist)
32:                if closestDistQueue.size() > ef then
33:                    closestDistQueue.pop()
34:                end if
35:                updateTopK(res, k, data[neighb], neighbDist)
36:            end if
37:        end for
38:    end while
39:    return res
40: end procedure
```

Figure 2.3: A $k$-NN search algorithm using SW-graph.

In this work, we use an efficient proximity-graph method called a navigable *Small World* graph (SW-graph) [196]. The graph constructed by this method is undirected. Both indexing and retrieval re-use essentially the same search algorithm, which consists in a series of greedy sub-searches. A sub-search starts at some, e.g., random node and proceeds to expanding the set of traversed nodes in a best-first fashion by following neighboring links. The sub-search search terminates when the candidate queue is exhausted. Specifically, the candidate queue is expanded only with points that are closer to the query than the $k'$*-th* closest point already discovered by the sub-search ($k'$ is a search parameter). When we stop finding such points, the queue dwindles and eventually becomes empty. We also terminate the sub-search when the queue contains only points farther than the $k'$*-th* closest point already discovered by the sub-search. The algorithm is outlined in Figure 2.3. Note that this search algorithm is only approximate and does not necessarily return all true nearest neighbors.

Indexing is a bottom-up procedure that relies on this best-first greedy search algorithm. We insert points one by one. For each data point, we find `NN` closest points using an already constructed index (`NN` is a parameter). Then, we create an *undirected* edge between a new graph node (representing a new point) and nodes that represent `NN` closest points found by the greedy search (these are `NN` closest points collected from all sub-searches). Each sub-search starts from some, e.g., random node and proceeds expanding the candidate queue with points that are closer than the `efConstruction`-*th* closest point. Similarly, the search procedure executes one or more sub-searches that start from some node. The queue is expanded only with points that are closer than the `efSearch`-*th* closest point. `efConstruction` and `efSearch` are parameters that define the value of the above-mentioned parameter $k'$ at index- and query-time, respectively. The number of sub-searches is defined by the parameter `initIndexAttempts` (at index time) and `initSearchAttempts` (at query time).

Empirically, it was shown that this method often creates a navigable small world graph, where most nodes are separated by only a few edges (roughly logarithmic in terms of the overall number of objects) [195]. A simpler and less efficient variant of this algorithm was presented at ICTA 2011 and SISAP 2012 [195, 244]. An improved variant appeared as an Information Systems publication [196]. In the latter paper, however, the values of `efSearch` and `efConstruction` are set equal to `NN`. The idea of using values of `efSearch` and `efConstruction` potentially (much) larger than `NN` was proposed by Malkov and Yashunin [197].

The indexing algorithm is rather expensive and we accelerate it by running parallel searches in multiple threads. The number of threads is defined by the parameter `indexThreadQty`. By default, this parameter is equal to the number of virtual cores. The graph updates are synchronized: If a thread needs to add edges to a node or obtain the list of node edges, it first locks a node-specific mutex. Because different threads rarely update and/or access the same node simultaneously, such synchronization creates little contention and, consequently, this parallelization approach is efficient. It is also necessary to synchronize updates for the list of graph nodes, but this operation takes little time compared to finding `NN` neighboring points. Because the search algorithm does not need to visit the same node twice, we need a data structure for membership queries. To this end, we adopt an approach of Wei Dong [95], who uses a bitmask of the fixed size.

#### 2.2.2.4 Permutation Methods

**Core Principles** Permutation methods are *filter-and-refine* methods belonging to the class of pivoting searching techniques. *Pivots* (henceforth denoted as $\pi_i$) are reference points randomly selected during indexing. To create an index, we compute the distance from every data point $x$ to every pivot $\pi_i$. We then memorize either the original distances or some distance statistics in the hope that these statistics can be useful during searching. At search time, we compute distances from the query to pivots and prune data points using, e.g., the triangle inequality [262, 326] or its generalization for non-metric spaces [108].

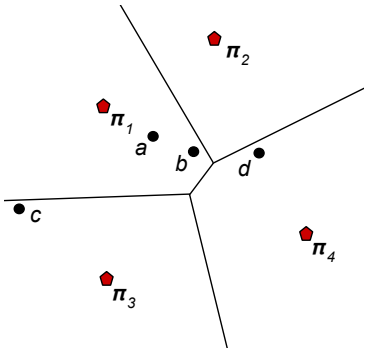



Figure 2.4: Voronoi diagram produced by four pivots $\pi_i$. The data points are $a$, $b$, $c$, and $d$. The distance is $L_2$.

Alternatively, rather than relying on distance values directly, we can use precomputed statistics to produce estimates for distances between the query and data points. In particular, in the case of permutation methods, we assess similarity of objects based on their relative distances to pivots. To this end, for each data point $x$, we arrange pivots $\pi_i$ in the order of increasing distances from $x$. The ties can be resolved, e.g., by selecting a pivot with the smallest index. Such a *permutation* (i.e., ranking) of pivots is essentially a vector whose $i$-th element keeps an ordinal position of the $i$-th pivot in the set of pivots sorted by their distances from $x$. We say that point $x$ *induces* the permutation.

Consider the Voronoi diagram in Figure 2.4 produced by pivots $\pi_1$, $\pi_2$, $\pi_3$, and $\pi_4$. Each pivot $\pi_i$ is associated with its own cell containing points that are *closer* to $\pi_i$ than to any other pivot $\pi_j, i \neq j$. The neighboring cells of two pivots are separated by a segment of the line *equidistant* to these pivots. Each of the data points $a$, $b$, $c$, and $d$ "sits" in the cell of its closest pivot.

For the data point $a$, points $\pi_1$, $\pi_2$, $\pi_3$, and $\pi_4$ are respectively the first, the second, the third, and the forth closest pivots. Therefore, the point $a$ induces the permutation $(1, 2, 3, 4)$. For the data point $b$, which is the nearest neighbor of $a$, two closest pivots are also $\pi_1$ and $\pi_2$. However, $\pi_4$ is closer than $\pi_3$. Therefore, the permutation induced by $b$ is $(1, 2, 4, 3)$. Likewise, the permutations induced by $c$ and $d$ are $(2, 3, 1, 4)$ and $(3, 2, 4, 1)$, respectively.

The *underpinning assumption* of permutation methods is that most nearest neighbors can be found by retrieving a small fraction of data points whose pivot rankings, i.e., the induced permutations, are similar to the pivot ranking of the query. Two most popular choices to compare the rankings $x$ and $y$ are: Spearman's rho distance (equal to the squared $L_2$) and the Footrule distance (equal to $L_1$) [67, 91]. More formally, SpearmanRho$(x, y) = \sum_i (x_i - y_i)^2$ and Footrule$(x, y) = \sum_i |x_i - y_i|$. According to Chávez et al. [67] Spearman's rho is more effective than the Footrule distance. This has been also confirmed by our own experiments.

Converting the vector of distances to pivots into a permutation entails information loss, but this loss is not necessarily detrimental. In particular, our preliminary experiments have showed that using permutations instead of vectors of original distances results in slightly better retrieval performance. The information about relative positions of the pivots can be further coarsened by binarization: All elements smaller than a threshold $b$ become zeros and elements at least as large as $b$ become ones [283]. The similarity of binarized permutations is computed via the Hamming distance.

In the example of Figure 2.4, the values of the Footrule distance between the permutation of $a$ and permutations of $b$, $c$, and $d$ are equal to 2, 4, and 6, respectively. Note that the Footrule distance on permutations correctly "predicts" the closest neighbor of $a$. Yet, the ordering of points based on the Footrule distance is not perfect: the Footrule distance from the permutation of $a$ to the permutation of its second nearest neighbor $d$ is larger than the Footrule distance to the permutation of the third nearest neighbor $c$.

Given the threshold $b = 3$, the binarized permutations induced by $a$, $b$, $c$, and $d$ are equal to $(0, 0, 1, 1)$, $(0, 0, 1, 1)$, $(0, 1, 0, 1)$, and $(1, 0, 1, 0)$, respectively. In this example, the binarized permutation of $a$ and its nearest neighbor $b$ are equal, i.e., the distance between respective permutations is zero. When we compare $a$ to $c$ and $d$, the Hamming distance does not discriminate between $c$ and $d$ as their binary permutations are both at distance two from the binary permutation of $a$.

Permutation-based searching belongs to a class of *filter-and-refine* methods, where objects are mapped to data points in a low-dimensional space (usually $L_1$ or $L_2$). Given a permutation of a query, we carry out a nearest neighbor search in the space of permutations. Retrieved entries represent a (hopefully) small list of candidate data points that are compared directly to the query using the distance in the *original* space. The permutation methods differ in ways of producing candidate records, i.e., in the way of carrying out the filtering step. In the next sections we describe these methods in detail.

Permutation methods are similar to the rank-aggregation method OMEDRANK due to Fagin et al. [106]. In OMEDRANK there is a small set of voting pivots, each of which ranks data points based on a somewhat imperfect notion of the distance from points to the query (e.g., computed via a random projection). While each individual ranking is imperfect, a more accurate ranking can be achieved by rank aggregation. Thus, unlike permutation methods, OMEDRANK uses pivots to rank data points and aims to find an *unknown* permutation of *data points* that reconciles differences in data point rankings in the best possible way. When such a consolidating ranking is found, the most highly ranked objects from this *aggregate* ranking can be used as answers to a nearest-neighbor query. Finding the aggregate ranking is an NP-complete problem that Fagin et al. [106] solve only heuristically. In contrast, permutation methods use data points to rank pivots and solve a much simpler problem of finding *already known and computed* permutations of *pivots* that are the best matches for the query permutation.

**Brute-force Searching of Permutations** In this approach, the filtering stage is implemented as a brute-force comparison of the query permutation against the permutations of the data with subsequent selection of the $\gamma$ entries that are $\gamma$-nearest objects in the space of permutations. A number of candidate entries $\gamma$ is a parameter of the search algorithm that is often understood as a fraction (or percentage) of the total number of points. Because the distance in the space of permutations is not a perfect proxy for the original distance, to answer a $k$-NN-query with high accuracy, the number of candidate records has to be much larger than $k$ (see § 2.3.1.4).

A straightforward implementation of brute-force searching relies on a priority queue. Chávez et al. [67] proposed to use incremental sorting as a more efficient alternative. In our experiments with the $L_2$ distance, the latter approach is twice as fast as the approach relying on a standard C++ implementation of a priority queue.

The cost of the filtering stage can be reduced by using binarized permutations [283]. Binarized permutations can be stored compactly as bit arrays. Computing the Hamming distance between bit arrays can be done efficiently by XOR-ing corresponding computer words and counting the number of non-zero bits of the result. For bit-counting, one can use a special instruction available on many modern CPUs. [13]

Brute-force searching in the permutation space, unfortunately, is not very efficient, especially if the distance can be easily computed: If the distance is "cheap" (e.g., $L_2$) and the index is stored in main memory, brute-force search in the space of permutations is not much faster than brute-force search in the original space.

**Indexing of Permutations** To reduce the cost of the filtering stage of permutation-based searching, three types of indices were proposed: the Permutation Prefix Index (PP-Index) [105], existing methods for metric spaces [112], and the Metric Inverted File (MI-file) [2].

Permutations are integer vectors whose values are between one and the total number of pivots $m$ (in our implementation, this value of $m$ is defined by parameter `numPivot`). We can view these vectors as sequences of symbols over a finite alphabet and index these sequences using a prefix tree. This approach is implemented in the PP-index. At query time, the method aims to retrieve $\gamma$ candidates by finding permutations that share a prefix of a given length with the permutation of the query object. This operation can be carried out efficiently via the prefix tree constructed at index time. If the search generates fewer candidates than a specified threshold $\gamma$, the procedure is recursively repeated using a shorter prefix. For example, the permutations of points $a$, $b$, $c$, and $d$ in Figure 2.4 can be seen as strings `1234`, `1243`, `2314`, and `3241`. The permutation of points $a$ and $b$, which are nearest neighbors, share a two-character prefix with $a$. In contrast, permutations of points $c$ and $d$, which are more distant from $a$ than $b$, have no common prefix with $a$.

To achieve good recall, it may be necessary to use short prefixes. However, longer prefixes are more selective than shorter ones (i.e., they generate fewer candidate records) and are, therefore, preferred for efficiency reasons. In practice, a good trade-off between recall and efficiency is typically achieved only by building several copies of the PP-index (using different subsets of pivots) [3].

Figueroa and Fredriksson experimented with indexing permutations using well-known data structures for metric spaces [112]. Indeed, the most commonly used permutation distance: Spearman's rho, is a monotonic transformation, namely squaring, of the Euclidean distance. Thus, it should be possible to find $\gamma$ nearest neighbors by indexing permutations, e.g., in a VP-tree [234, 293, 323].

Amato and Savino proposed to index permutation using an inverted file [2]. They called their method the MI-file. To build the MI-file, they first select $m$ pivots and compute their permutations/rankings induced by data points. For each data point, $m_i \leq m$ closest pivots are indexed in the inverted file (in our implementation the value of $m_i$ is defined by the parameter `numPivotIndex`). Each posting is a pair $(pos(\pi_i, x), x)$, where $x$ is the identifier of the data point and $pos(\pi_i, x)$ is a position of the pivot in the permutation induced by $x$. Postings of the same pivot are sorted by pivot's positions.

[13] In C++, this instruction is provided via the intrinsic function `__builtin_popcount`.

Consider the example of Figure 2.4 and imagine that we index two closest pivots (i.e., $m_i = 2$). The point $a$ induces the permutation $(1, 2, 3, 4)$. Two closest pivots $\pi_1$ and $\pi_2$ generate postings $(1, a)$ and $(2, a)$. The point $b$ induces the permutation $(1, 2, 4, 3)$. Again, $\pi_1$ and $\pi_2$ are two pivots closest to $b$. The respective postings are $(1, b)$ and $(2, b)$. The permutation of $c$ is $(2, 3, 1, 4)$. Two closest pivots are $\pi_1$ and $\pi_3$. The respective postings are $(2, c)$ and $(1, c)$. The permutation of $d$ is $(3, 2, 4, 1)$. Two closest pivots are $\pi_2$ and $\pi_4$ with corresponding postings $(2, d)$ and $(1, d)$.

At query time, we select $m_s \leq m_i$ pivots closest to the query $q$ and retrieve respective posting lists (in our implementation the value of $m_s$ is defined by the parameter `numPivotSearch`). If $m_s = m_i = m$, it is possible to compute the exact Footrule distance (or Spearman's rho) between the query permutation and the permutation induced by data points. One possible search algorithm keeps an accumulator (initially set to zero) for every data point. Posting lists are read one by one: For every encountered posting $(pos(\pi_i, x), x)$ we increase the accumulator of $x$ by the value $|pos(\pi_i, x) - pos(\pi_i, q)|$. If the goal is to compute Spearman's rho, the accumulator is increased by $|pos(\pi_i, x) - pos(\pi_i, q)|^2$.

If $m_s < m$, by construction of the posting lists, using the inverted index, it is possible to obtain rankings of only $m_s < m$ pivots. For the remaining, $m - m_s$ pivots we pessimistically assume that their rankings are all equal to $m$ (the maximum possible value). Unlike the case $m_i = m_s = m$, all accumulators are initially set to $m_s \cdot m$. Whenever we encounter a posting posting $(pos(\pi_i, x), x)$ we subtract $m - |pos(\pi_i, x) - pos(\pi_i, q)|$ from the accumulator of $x$.

Consider again the example of Figure 2.4. Let $m_i = m_s = 2$ and $a$ be the query point. Initially, the accumulators of $b$, $c$, and $d$ contain values $4 \cdot 2 = 8$. Because $m_s = 2$, we read posting lists only of the two closest pivots for the query point $a$, i.e., $\pi_1$ and $\pi_2$. The posting lists of $\pi_1$ is comprised of $(1, a)$, $(1, b)$, and $(2, c)$. On reading them (and ignoring postings related to the query $a$), accumulators $b$ and $c$ are decreased by $4 - |1 - 1| = 4$ and $4 - |1 - 2| = 3$, respectively. The posting lists of $\pi_2$ are $(2, a)$, $(2, b)$, and $(2, d)$. On reading them, we subtract $4 - |2 - 2| = 4$ from each of the accumulators $b$ and $d$. In the end, the accumulators $b$, $c$, $d$ are equal to 0, 5, and 4. Unlike the case when we compute the Footrule distance between complete permutation, the Footrule distance on truncated permutations correctly predicts the order of three nearest neighbors of $a$.

Using fewer pivots at retrieval time allows us to reduce the number of processed posting lists. Another optimization consists in keeping posting lists sorted by pivots position $pos(\pi_i, x)$ and retrieving only the entries satisfying the following restriction on the maximum position difference: $|pos(\pi_i, x) - pos(\pi_i, q)| \leq D$, where $D$ is a method parameter. Because posting list entries are sorted by pivot positions, the first and the last entry satisfying the maximum position difference requirement can be efficiently found via the binary search.

Tellez et al. [284] proposed a modification of the MI-file which they called a Neighborhood APProximation index (NAPP). In the case of NAPP, there also exist a large set of $m$ pivots of which only $m_i < m$ pivots (closest to inducing data points) are indexed. Recall that $m_i$ is defined by the parameter `numPivotIndex`. Unlike the MI-file, however, posting lists contain only object identifiers, but no positions of pivots in permutations.

This simplification has two consequences. First, the size of the inverted index is smaller: for each document the index has only $m_i$ data point identifiers (but no pivot positions). Second, it is not possible to compute an estimate for the Footrule distance by reading only posting lists. Therefore, instead of an estimate for the Footrule distance, the number of closest *common* pivots

is used to sort candidate objects. In addition, the candidate objects sharing with the query fewer than $m_s$ closest pivots are discarded ($m_s$ is defined by the parameter `numPivotSearch`). For example, points $a$ and $b$ in Figure 2.4 share the same common pivot $\pi_1$. At the same time $a$ does not share any closest pivot with points $d$ and $c$. Therefore, if we use $a$ as a query, the point $b$ will be considered to be the best candidate point.

Chávez et al. [68] proposed a single framework that unifies several approaches including PP-index, MI-file, and NAPP. Similar to the PP-index, permutations are viewed as strings over a finite alphabet. However, these strings are indexed using a special sequence index that efficiently supports rank and select operations. These operations can be used to simulate various index traversal modes, including, e.g., retrieval of all strings whose $i$-th symbol is equal to a given one.

### 2.2.3 A Novel Modification of VP-tree

#### 2.2.3.1 Motivation

Despite recent progress on search methods for non-metric Bregman divergences [58, 227, 327], an overwhelming majority of proposed space-partitioning search methods can work only with metric distances. This should not be surprising, because extending exact space-partitioning methods to non-metric distances requires a non-trivial intellectual effort. Only few exact non-metric search methods are proposed. Moreover, when exact algorithms exist, they are not necessarily efficient for high-dimensional data [34, 58, 311]. As a result, one has to resort to approximate searching.

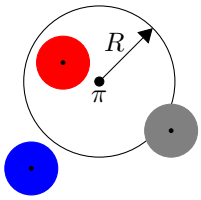



Figure 2.5: Three types of query balls in VP-tree.

The black circle (centered at the pivot $\pi$) is the sphere that divides the space.

If approximate searching is unavoidable, designing an exact pruning algorithm analytically does not make much sense. Instead, we can spare an effort and *learn* an empirical approximation of the pruning rule from data. In particular, we propose to employ a simple parametric model whose parameters are tuned to maximize efficiency at a given recall. This can be seen as a simple form of machine learning. Note that there are a number of approaches, where machine learning is used to estimate optimal parameters of classic search methods (see, e.g., [60, 169]). Yet, this work is the first successful attempt to employ VP-tree with the learned pruning algorithm in non-metric spaces. In the remainder of this subsection, we describe our approach in detail.

Note that Nielsen et al. [227] also use VP-tree for non-metric data sets. However, their method has two limitations: (1) it works only with Bregman divergences, (2) it can never prune the outer partition induced by a pivot. Specifically, Nielsen et al. [227] propose an improved algorithm to verify if one Bregman ball (e.g., the query ball) intersects another ball (e.g., the one that covers data points). However, VP-tree (see a more detailed explanation below) divides the space into two partitions, where only the inner partition is covered by a ball. The outer partition is the ball's complement, which is not a ball. Yet, unlike the classic pruning algorithm of VP-tree, the pruning algorithm of Nielsen et al. [227] can check if the query ball intersects with the inner partition, but not with the outer one. Thus, it can never prune the outer partition even when it is safe to do so.

#### 2.2.3.2 Main Idea

Let us review the indexing and searching algorithms of the classic VP-tree method in detail. VP-tree is a hierarchical space-partitioning method, which divides the space using hyperspheres [234, 293, 323]. The output of the indexing algorithm is a hierarchical partitioning of the data set. This partitioning is represented by a binary tree.

The indexing algorithm is a *recursive* procedure that operates on a subset of data—which we call an *active subset*—and on a partially built tree. When indexing starts, the tree contains only the root and the active subset contains all data points. At each step of recursion, the indexing algorithm checks if the number of active data points is below a certain threshold called the bucket size. If this is the case, the active data points are simply stored as a single bucket. Otherwise, the algorithm divides the active subset into two nearly equal parts, each of which is further processed recursively.

Division of the active subset starts with selecting a pivot $\pi$ (e.g., randomly) and computing the distance from $\pi$ to every other data point in the active subset. Assume that $R$ is the median distance. Then, the active subset is divided into two subsets by the hypersphere with radius $R$ and center $\pi$. Two subtrees are created. Points inside the pivot-centered hypersphere are placed into the left subtree. Points outside the pivot-centered hypersphere are placed into the right subtree. Points on the separating hypersphere may be placed arbitrarily. Because $R$ is the median distance, each of the subtrees contains approximately half of active data points.

As we explained previously, in the case of space-partitioning methods, $k$-NN search is simulated via the range search with a shrinking radius. The search algorithm is a best-first traversal procedure that starts from the root of the tree and proceeds recursively. It updates the search radius $r$ as it encounters new close data points. Let us consider one step of recursion. If the search algorithm reaches a leaf of the tree, i.e., a bucket, all bucket elements are compared against the query. In other words, elements in the buckets are searched via brute-force search.

If the algorithm reaches an internal node $X$, there are exactly two subtrees representing two spaces partitions. The query belongs to exactly one partition. This is the "best" partition and the search algorithm always explores this partition recursively before deciding whether to explore the other partition. While exploring the best partition, we may encounter new close data points (pivots or bucket points) and further shrink the search radius. On completing the sub-recursion and returning to node $X$, we make a decision about skipping the other partition.

This algorithm is outlined in Figure 2.6. An essential part of this process is a decision function `isPruningPossible`, which identifies situations when pruning is possible without sacrificing too much in accuracy. Let us review the decision process. Recall that each internal node keeps pivot $\pi$ and radius $R$, which define the division of the space into two subspaces. Although there are many ways to place a query ball, all locations can be divided into three categories, which are illustrated by Figure 2.5. The red query ball "sits" inside the inner partition. Note that it does not intersect with the outer partition. For this reason, the outer partition cannot have sufficiently close data points, i.e., points with radius $r$ from the query. Hence, this partition can be safely pruned. The blue query ball is located in the outer partition. Likewise, it does not intersect the other, inner, partition. Thus, this inner partition can be safely pruned. Finally, the gray query ball intersects both partitions. In this situation, sufficiently close points may be located in both partitions and no safe pruning is possible.

```
1: procedure UPDATETOPK(res, k, x, dist)                    ▷ res is max priority queue
2:                                                          ▷ passed by ref/pointer
3:     res.push( (dist, x) )                                ▷ This permits an obvious optimization,
4:     if res.size() > k then                               ▷ which we omit to simplify presentation.
5:         topK.pop()
6:     end if
7: end procedure
8:
9: procedure KNNSEARCH(curr, q, k, r, res)                  ▷ Passed by ref/pointer
10:    if curr.isBucket() == True then
11:        for e in curr.bucket do
12:            updateTopK(res, k, e, d(e, q))               ▷ Update the result set
13:            r ← min(r, d(e, q)))                         ▷ Shrink the radius
14:        end for
15:    else
16:        updateTopK(res, k, π, d(π, q))                   ▷ Update the result set
17:        r ← min(r, d(π, q)))                             ▷ Shrink the radius
18:        if isPruningPossible(π, R, q, r) == True then
19:            if d(π, q) < R then          ▷ The query ball center is in the inner partition
20:                KNNSearch(curr.left, q, r)
21:            else                         ▷ The query ball center is in the outer partition
22:                KNNSearch(curr.right, q, r)
23:            end if
24:        else                                  ▷ The query ball intersects both partitions
25:                                  ▷ and pruning can cause a substantial accuracy loss
26:            KNNSearch(curr.left, q, r)
27:            KNNSearch(curr.right, q, r)
28:        end if
29:    end if
30: end procedure
```

Figure 2.6: A $k$-NN search algorithm using VP-tree.

Clearly, the pruning algorithm can be seen as a ternary or binary classification problem, whose outcome is prediction of the query ball type. In the ternary classification variant, the classifier would be able to distinguish among three types of the ball. However, for practical purposes, there is no need to distinguish between two cases where the query ball is located completely inside one partition without touching the boundary. In both of these cases, we can prune the other partition. Most importantly, we need to detect a situation when the query ball intersects both partitions or not, because this is the case when pruning is not possible. Thus, the actual classification problem of interest is binary.

Should we decide to train a classifier, we need to do the following: (1) collect training data and (2) define features. Collecting training data is a relatively straightforward *sampling process*. It can be done, e.g., as follows. First we select several pivots randomly. For each pivot, we compute the distance to all data points. Next, we compute the median and obtain two data set partitions (recall that the inner partition corresponds to distances smaller than the median). We then select a bunch of data points randomly. Finally, we consider a sequence of increasing radii each of which defines the query ball for the sampled points.

A combination of the query ball center and its radius represent one *training sample*. For each training point, we check which partitions contain data points inside the respective query ball (this verification is carried out using brute-force search). If the query ball contains only data points from the partition where the query ball center is located, we label the training sample with zero. It represents the case when the other partition can be pruned. If the query ball contains at least one data point from the other partition, we label the training sample with one. This is the case when pruning is not possible.

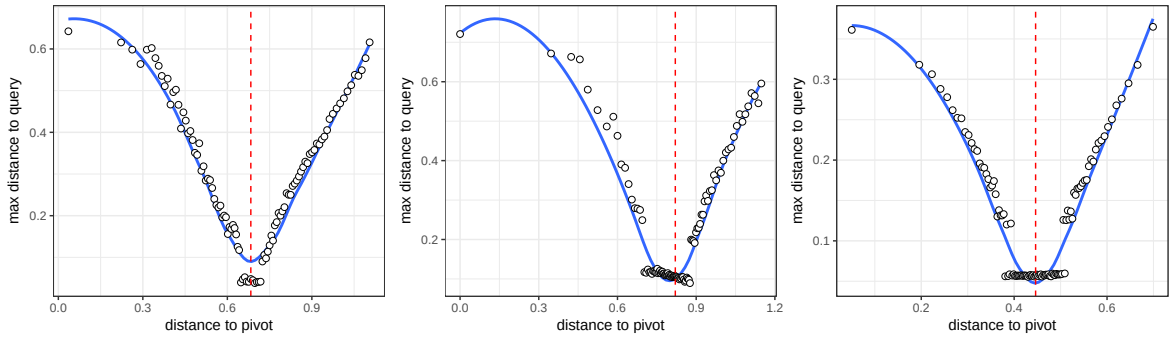
One obvious feature is the query radius. Defining other features is trickier. In vector spaces, one might resort to using all vector coordinates. However, such a solution would require too much training data. It would also generalize poorly in high dimensions due to potential sampling biases and "emptiness" of a high-dimensional space. Consider, for example, a unit-side hypercube. If we divide each side into two halves, such as division induces a division of the cube into $2^d$ cells of equal shape and volume. For a moderate value of $d = 50$, the number of cells exceeds a quadrillion. It should now be clear that for any reasonably-sized training set a vast majority of these cells would not contain a single training point. In other words, our training data would be incredibly sparse. The situation is not hopeless when a data set points are located close to a low-dimensional manifold. Yet, collecting a comprehensive training set representing all manifold "corners" is a complicated task.

For these reasons, we propose a much simpler feature set. In addition to requiring less training data, such a representation permits implementing a fast pruning classifier, which is essential to the overall efficiency of the search method. Specifically, in addition to the query radius $r$, we use only one feature, whose value is the distance $d(\pi, q)$ from the query to the pivot. To see why this is a useful suggestion, let us consider the classic pruning rule employed in VP-tree [234, 293, 323].

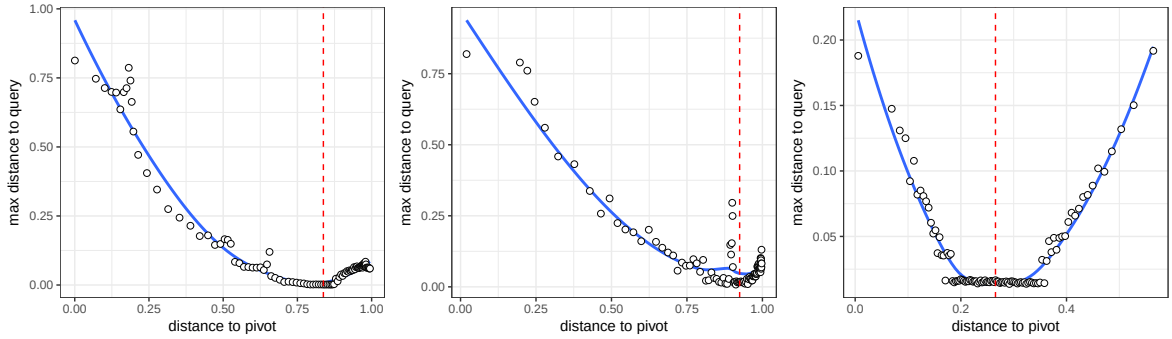
Using the triangle inequality, one can easily obtain that the search algorithm has to visit:

- *only* the left subtree if $d(\pi, q) < R - r$;
- *only* the right subtree if $d(\pi, q) > R + r$;
- both subtrees if $R - r \leq d(\pi, q) \leq R + r$.

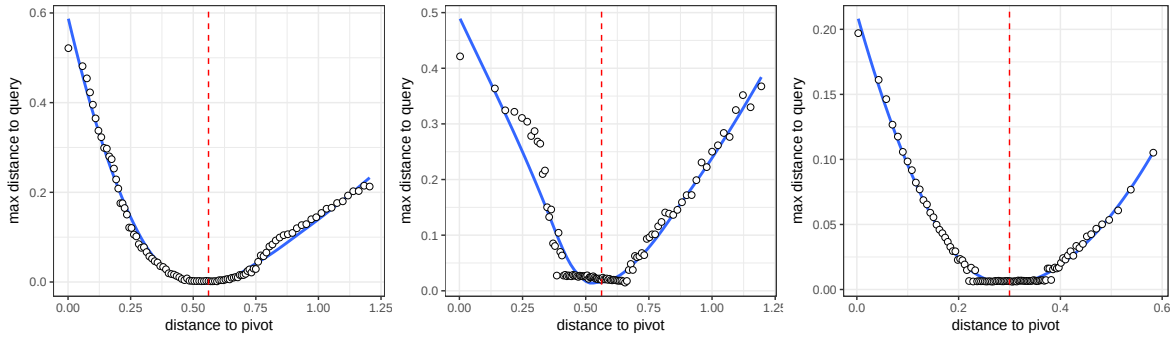
Let us rewrite these rules using notation $D_{\pi,R}(x) = |R - x|$. It is easy to see that the search



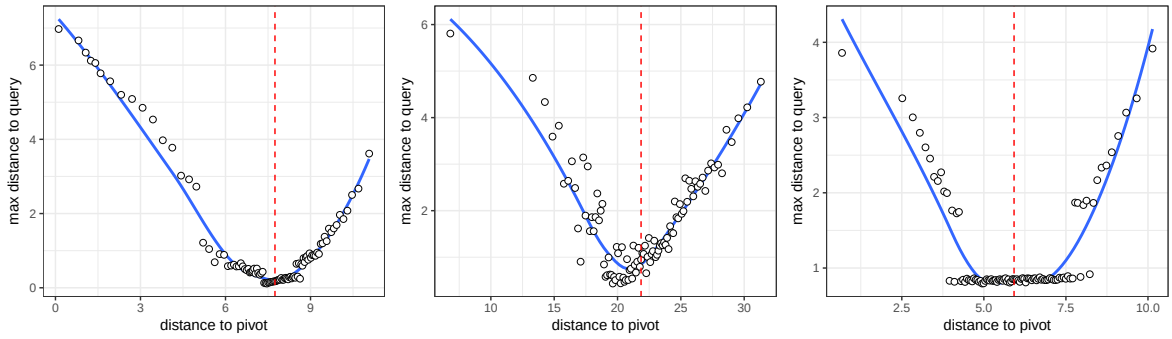

(a) RCV-8 ($L_2$) (b) RCV-32 ($L_2$) (c) RandHist-8 ($L_2$)




(d) RCV-8 (Cosine dist.) (e) RCV-32 (Cosine dist.) (f) RandHist-8 (Cosine dist.)



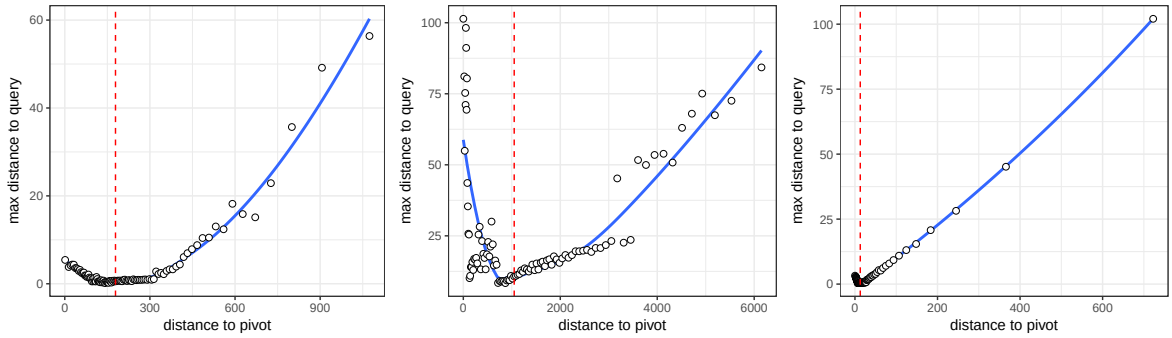

(g) RCV-8 ($L_2^2$) (h) RCV-32 ($L_2^2$) (i) RandHist-8 ($L_2^2$)



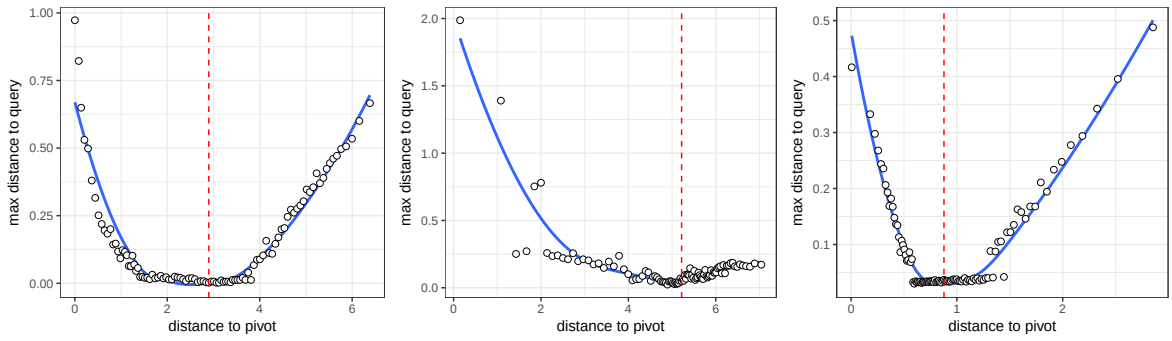

(j) RCV-8 ($L_{0.5}$) (k) RCV-32 ($L_{0.5}$) (l) RandHist-8 ($L_{0.5}$)

Figure 2.7: The empirically obtained pruning decision function $D_{\pi,R}(x)$ (part I).

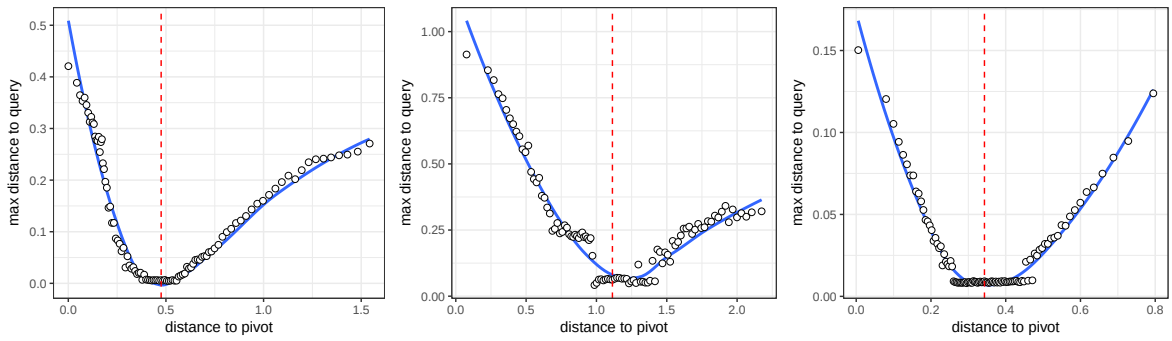
Each point is a value of the function learned by sampling (see Section 2 of the supplemental materials in [43]). Blue curves are fit to these points. The red dashed line denotes a median distance $R$ from data set points to the pivot $\pi$.



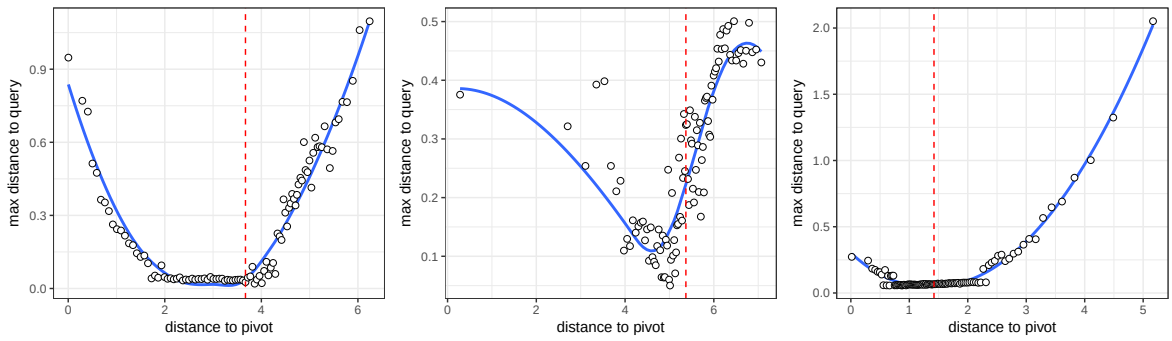

(a) RCV-8 (Itakura-Saito) (b) RCV-32 (Itakura-Saito) (c) RandHist-8 (Itakura-Saito)




(d) RCV-8 (KL-div.) (e) RCV-32 (KL-div.) (f) RandHist-8 (KL-div.)




(g) RCV-8 (Rényi div. $\alpha = 0.25$) (h) RCV-32 (Rényi div. $\alpha = 0.25$) (i) RandHist-8 (Rényi div. $\alpha = 0.25$)




(j) RCV-8 (Rényi div. $\alpha = 2$) (k) RCV-32 (Rényi div. $\alpha = 2$) (l) RandHist-8 (Rényi div. $\alpha = 2$)

Figure 2.8: The empirically obtained pruning decision function $D_{\pi,R}(x)$ (part II).

Each point is a value of the function learned by sampling (see Section 2 of the supplemental materials in [43]). Blue curves are fit to these points. The red dashed line denotes a median distance $R$ from data set points to the pivot $\pi$.

algorithm has to visit both partitions if and only if $r \geq D_{\pi,R}(d(\pi,q))$. If $r < D_{\pi,R}(d(\pi,q))$, we need to visit only one partition, which contains the query point. The other partition can be safely pruned.

Thus, in the case of a metric space, the pruning algorithm employs only two features: $r$ and $d(\pi,q)$. Furthermore, the pruning decision is made by comparing one feature, namely the query radius $r$, with the value of the function $D_{\pi,R}(x)$, whose only argument is the distance from the query to the pivot $d(\pi,q)$. We hypothesize that this basic pruning rule can be used for non-metric data with only a modest modification. Specifically, a non-metric search algorithm have to rely on a different function $D_{\pi,R}(x)$. This function can be hard to compute analytically. Yet, it may be learned from data.

#### 2.2.3.3 Learning $D_{\pi,R}(x)$ from Data

Our initial approach to learn $D_{\pi,R}(x)$ employed a stratified sampling procedure (see § 2 of the supplemental materials of our publication [43]). Unfortunately, we have discovered that the sampling method is expensive and not very accurate. For this reason, we have also implemented a simple parametric approximation whose parameters are selected to optimize efficiency at a given value of recall. The simple parametric approach is about twice as efficient as sampling on nearly every data set that we have tried. Hence, only the parametric approach will be discussed in detail. One possible reason for this performance gap is that the parametric approach directly optimizes the target metric.

To choose the right parametric representation, we examine the functions $D_{\pi,R}(x)$ learned by the sampling algorithm. To this end we use the following data sets and distances:

- *RCV-d*, $d \in \{8, 32\}$: $d$-dimensional data sets each containing $0.5 \cdot 10^6$ topic histograms. The distance is KL-divergence. These data sets were created by Cayton [58]from the RCV1 corpus of newswire stories [180] using latent Dirichlet allocation (LDA) [38].
- *RandHist-8* $0.5 \cdot 10^6$ 8-dimensional histograms sampled uniformly from a simplex.

Plots of functions $D_{\pi,R}(x)$ learned from data are shown in Figures 2.7 and 2.8. Small dots in these plots represent function values obtained by sampling. Blue curves are fit to these dots. For the Euclidean data (Panels 2.7a-2.7c in Figure 2.7), $D_{\pi,R}(x)$ resembles a piecewise linear function approximately equal to $|R - x|$. For the KL-divergence data (Panels 2.8d-2.8f) in Figure 2.8, $D_{\pi,R}(x)$ looks like either a U-shape or a hockey-stick curve. These observations motivated the use of a piecewise polynomial decision function, which is formally defined as:

$$D_{\pi,R}(x) = \begin{cases} \alpha_{left}|x - R|^{\beta_{left}}, & \text{if } x \leq R \\ \alpha_{right}|x - R|^{\beta_{right}}, & \text{if } x \geq R \end{cases}, \tag{2.11}$$

where $\beta_i$ are positive integers. First of all, preliminary experiments have shown that using different $\beta_i$ does not make our pruning function sufficiently more accurate. However, it makes it the optimization problem harder, due to additional parameters. For this reason, we use only a single value of $\beta = \beta_1 = \beta_2$.

Second, we find that in many cases, a polynomial approximation is not better than a piecewise linear one, especially when dimensionality is high. This is not very surprising: Due to the concentration of measure, for most data points the distance to the pivot $\pi$ is close to the median

distance $R$ (which corresponds to the boundary between two VP-tree partitions). If we explore the shape of $D_{\pi,R}(x)$ in Panels 2.7a, 2.8e, 2.8h (and several others) around the median (denoted by a dashed red line), we can see that a piecewise linear shape approximation is quite reasonable. To sum up, the parametric approximation that we use most frequently is defined as:

$$D_{\pi,R}(x) = \begin{cases} \alpha_{left}|x - R|, & \text{if } x \leq R \\ \alpha_{right}|x - R|, & \text{if } x \geq R \end{cases} \tag{2.12}$$

This is similar to stretching of the triangle inequality proposed by Chávez and Navarro [63]. There are two crucial differences, however. First, we utilize different values of $\alpha_i$, i.e., $\alpha_{left} \neq \alpha_{right}$, while Chávez and Navarro used $\alpha_{left} = \alpha_{right}$. Second, we devise a simple training procedure to obtain values of $\alpha_i$ that maximize efficiency at a given recall value.

Note that lower values of $\alpha_{left}$ and $\alpha_{right}$ correspond to less aggressive pruning: We can prevent nearly all pruning from happening by making $\alpha_i$ to be very small. In this case, the recall will be close to 100%, but efficiency would be comparable to that of the brute-force search. Likewise, we can prune nearly everything by making $\alpha_i$ very large. Yet, the recall would be close to zero. Between these two extreme cases, there are useful trade-offs that we can explore.

To find these trade-offs we further assume that both recall and efficiency of the search algorithm are approximately monotonic functions of parameters $\alpha_i$. If we keep, e.g., $\alpha_{left}$ fixed and increase $\alpha_{right}$, efficiency mostly improves (or remains the same), but recall mostly deteriorates (or remains the same).

In the case of range search, monotonicity holds strictly and not approximately. To see why this is true, let us revisit the pruning rule, e.g., for the outer partition. From Eq. 2.11 we obtain, that for the query ball center in the inner partition, the outer partition can be pruned only if the following holds:

$$r \leq \alpha_{left}|x - R|^{\beta_{left}} \tag{2.13}$$

If we increase $\alpha_{left}$, the right hand side value of this inequality will increase in all nodes with $x < R$. Hence the set of nodes where pruning happens will expand monotonically. In other words, in addition to the sub-trees that are already pruned, we may prune some additional nodes. The latter means that efficiency improves, but the subset of found data points shrinks monotonically (by monotonically decreasing sequence of sets we understand the sequence where set $i$ is a subset of set $j$ if $j > i$). Thus, recall either decreases or remains the same. Similarity, decreasing the value of $\alpha_{right}$ leads to a monotonic expansion of the set of visited VP-tree nodes (for a constant $\alpha_{left}$).

In the case of $k$-NN search, the radius of the query shrinks as we encounter new points. If we increase $\alpha_{left}$ and prune some additional sub-trees, the query radius $r$ may shrink by a smaller value than previously (before the increase in $\alpha_{left}$). For some nodes, the increase in the query radius may even counterbalance a *small* increase in $\alpha_{left}$ in Eq. 2.13. A more substantial increase in $\alpha_{left}$ will help prune such nodes as well. However, we cannot expect that the set of pruned nodes will expand strictly monotonically.

**Main optimization procedure** An observation about (approximate) monotonicity inspired a simple optimization algorithm, which can be seen as a variant of an adaptive grid search. A

**Exposition 2.8: Benefits of a multiplicative step**

Imagine that the multiplicative step $\rho = 2$ and initial values $\alpha_{left} = \alpha_{right} = 1$. After one grid search is completed, we find that a better "efficiency point" is achieved with $\alpha_{left} = 0.5$ and $\alpha_{right} = 32$. If the grid search were additive, we might need to carry out a lot of iterations to arrive at this new optimal point. For example, if the grid step were 0.5, this would require at least $[(32 - 0.5)/0.5]^2] = 63^2 \approx 4000$ $k$-NN searches! In contrast, the multiplicative grid search may require only $(\log_2(32/0.5))^2 = 6^2 = 36$ $k$-NN searches. This saving comes at a price of using a much coarser step near the new optimal point. Specifically, we carry out a $k$-NN search for $\alpha_{left} = 0.5, \alpha_{right} = 16$ and $\alpha_{left} = 0.5, \alpha_{right} = 32$. Yet, there is no grid search for $\alpha_{left} = 0.5, 16 < \alpha_{right} < 32$. However, we would compensate for this deficiency by running another grid search with new initial values $\alpha_{left} = 0.5$ and $\alpha_{right} = 32$ and a smaller value of the multiplicative step. As we explained previously, this new search will make a more careful examination of values in the neighborhood of the new initial parameter values $\alpha_{left} = 0.5$ and $\alpha_{right} = 32$.

slightly simplified variant of this procedure is briefly outlined in our publication [43]. Here we present a more detailed description.

The optimization procedure involves creation of an index for a subset of data (both training points and training queries can be sampled from the original data set). Then, it would iteratively and recursively carry out a two-dimensional grid search with a "multiplicative" step $\rho$ and some initial parameter values $\alpha'_{left}$ and $\alpha'_{right}$. For each pair of parameter values, we carry out a $k$-NN search and measure reduction in the number of distance computations as well as the value of recall. If recall value is above a specified threshold, we compare the reduction in the number of distance computations with the best reduction seen so far. If the current reduction in the number of computations is better (i.e., its value is higher) than the previously observed one, we update the values of optimal parameters and memorize the new best reduction in the number of distance computations as well as the corresponding recall value.

If obtained recall values are considerably larger than a specified threshold, the procedure repeats the grid search using larger values of $\alpha'_{left}$ and $\alpha'_{right}$ (increasing the value of these parameters usually leads to decrease in recall and improvement in efficiency). Similarly, if the recall is not sufficient, the values of $\alpha'_{left}$ and $\alpha'_{right}$ are decreased and the grid search is repeated. If the values of recall encountered during the grid search are close to the desired recall, the grid search is restarted using a smaller grid step and new initial parameter values for $\alpha_{left}$ and $\alpha_{right}$. These parameter values correspond to the optimal parameter values obtained during a previous grid search.

Parameter values explored during a single grid search are given by the following equation:

$$\begin{array}{l} \alpha_{left} = \alpha'_{left}\rho^i, \ -m \leq i \leq m \\ \alpha_{right} = \alpha'_{right}\rho^j, \ -m \leq j \leq m \end{array}, \tag{2.14}$$

where $\rho$ and $m$ are hyperparameters. The multiplicative procedure in Eq. 2.14 aims to carry out a careful examination of the neighborhood of $(\alpha'_{left}, \alpha'_{right})$ as well as probe a few parameter combinations that are far from initial parameter values (see Exposition 2.8 for a more detailed

explanation). The ability to consider vastly different combinations of parameters is important, because, in many cases, best performance is achieved for small $\alpha_{left}$ and large $\alpha_{right}$ or vice versa.

The above described procedure assumes a constant value of $\beta$. In practice, this means that the optimization procedure is restarted using a few integer values of $\beta$, whose range is selected by the user. Also note that the user can chose to restart procedure several times using randomly selected initial values. This is necessary, because there is no guarantee of finding the global optimum.

**Compensating for the smaller size of the training set** Note that we obtain optimal parameter for a training set that is different from the main collection. Most importantly, the training set has a smaller size. Because the decision function is not exact, there is a nonzero probability that it makes a pruning mistake when the search algorithm arrives at a certain node. As a result of such a mistake, the search algorithm would prune a node containing a true answer. The larger is the data set, the more nodes VP-tree has and, consequently, the higher is a probability of mistake. For this reason, if we keep increasing the size of the data set (by adding new data points) while keeping parameter values constant, the recall would keep decreasing more or less monotonically. Consequently, if the training procedure uses a smaller set, it should also optimize for a higher recall value than it is required for the main collection.

To estimate the adjusted recall value, we make a simplifying assumption that probabilities of *not* making a mistake at any node *all* have the same value, which we denote as $p$. Second, we assume that mistakes happen independently. Consider a path from the root to some of the nearest neighbors. This theoretical setup is not novel (see, e.g., [20]).

Because, VP-tree is a balanced binary tree, it's length is approximately equal to $\log_2 n/B$, where $n$ is a number of data points and $B$ is the size of the bucket. Recall now that the search procedure does not make a pruning mistake in each node along this path with probability $p$. Therefore, the probability of overall success is equal to:

$$p_{\text{success}} = p^{\log_2 n/B} \tag{2.15}$$

$$p_{\text{success}} = 2^{\log_2 p^{\log_2 n/B}} = 2^{(\log_2 p)(\log_2 n/B)}. \tag{2.16}$$

Because the value of recall is a maximum likelihood estimate for the overall probability of success, we can use the value of recall and Eq. 2.16 to estimate the probability $p$ (of not making a mistake in a node) as follows:

$$\text{recall} \approx p_{\text{success}}$$

$$\text{recall} \approx 2^{(\log_2 p)(\log_2 n/B)}$$

$$\log_2 \text{recall} \approx (\log_2 p)(\log_2 n/B)$$

$$p \approx 2^{(\log_2 \text{recall})/\left(\log_2 \frac{n}{B}\right)} \tag{2.17}$$

Finally, by plugging the estimate given by Eq. 2.17 into Eq. 2.15, we can estimate the recall for a different data set size $n'$ as follows:

$$\left[2^{(\log_2 \text{recall})/\left(\log_2 \frac{n}{B}\right)}\right]^{\log_2 \frac{n'}{B}} = 2^{(\log_2 \text{recall})\left(\log_2 \frac{n'}{B}/\log_2 \frac{n}{B}\right)} = \left[2^{\log_2 \text{recall}}\right]^{\left(\log_2 \frac{n'}{B}/\log_2 \frac{n}{B}\right)} =$$

$$= \text{recall}^{\log_2 \frac{n'}{B} / \log_2 \frac{n}{B}} \tag{2.18}$$

In sum, to obtain a model that has a given value of recall for the data set of size $n$, we can train a model on data set of the size $n' < n$, as long as we use a larger desired recall value given by Eq. 2.18 during training.

Estimating model parameters is more accurate when VP-tree built on training data has similar depth to VP-tree built for the complete data set. For this reason, using larger training subsets (i.e., larger $n'$) permits a more accurate estimation of model parameters. Yet, it is also more expensive. To get extra training efficiency with little loss in accuracy, we propose to use a smaller bucket size during training (compared to the bucket size used to index the complete data set). Instead of indexing $n'$ records using bucket size $B > 1$, we propose to index $n'/B$ using the bucket size one. Note that we would still use the value of adjusted recall given by Eq. 2.18 computed as if the number of training data set size were $n'$ and the bucket size were $B$.

## 2.3 Experiments

In this section we carry out a series of intrinsic evaluations. The presentation is divided into two sections, where we pursue different objectives. In § 2.3.1, we focus on cross-method comparison using a diverse set of data. In § 2.3.2, we experiment with a less diverse but supposedly more challenging data, which relies only on non-metric distances. In particular, we assess an effectiveness of the approach, where the original hard-to-tackle distance is replaced with an "easier" proxy distance, obtained via symmetrization or distance learning. In § 2.3.2, we use only a limited set of search approaches, which include the brute-force search, SW-graph (§ 2.2.2.3), and two approaches relying on the classic VP-tree with a modified pruning rule.

### 2.3.1 Cross-Method Comparison with Diverse Data

This intrinsic evaluation carried in this section aims to compare three types of generic search methods: a VP-tree whose pruning rule can be modified to adapt to non-metric distances (§ 2.2.3), proximity/neighborhood graphs (§ 2.2.2.3), and permutation methods (§ 2.2.2.4). The underlying idea of permutation methods is that if we rank a set of reference points–called *pivots*–with respect to distances from a given point, the pivot rankings produced by two near points should be similar. Basically, this means that mapping from data points to respective pivot rankings defines a projection from the original space to, e.g., $L_2$. Furthermore, it was argued that for some data sets (see, e.g., [3, 66, 67]), this projection preserves the ordering in the original space quite well. However, nearly all experiments were restricted to metric data sets.

In contrast, we attempt to carry out a more comprehensive evaluation that involves a diverse set of metric and non-metric data sets, which come from image and text domains. We employ both sparse high-dimensional vector data and dense vector data of moderate dimensionality. In addition, we include non-vectorial data and experiment with both symmetric and non-symmetric distance functions. Because we aim for generality, we test each search method on both textual and image data. In earlier evaluations [43, 245] we mostly reuse data sets created by other researchers, in particular data sets created by Cayton for [58]. For evaluation described in this section, however,

**Exposition 2.9: Relationship between neighborhoods in the original and projected spaces**
Consider a 10-NN search in a data set containing one million data points. Let us assume that achieving 90% recall requires retrieving 0.1% of candidate data set points (which are closest to the query in the projected space). Thus, ensuring 90% level of recall of a 10-NN search in the original spaces requires carrying out a 1000-NN search in the projected space (as well as comparing 1000 candidate records with the query by computing the distance in the original space). We can therefore conclude that, on average, data points from an original 10-neighborhood become spread over a larger 1000-neighborhood in the projected space. The smaller is the degree of neighborhood "expansion" in the projected space, the better is a projected algorithm.

most of the data sets are created by us, in particular, to ensure that they are sufficiently large (every data sets contains at least one million data points).

Furthermore, we create two data sets with an expensive distance function to check if permutation methods would be useful in this case. Permutation-based search methods use filter-and-refine approach, where the filtering step can be costly. Restricting our attention to only cheap distances (as we did in earlier experiments [43, 245]) biases evaluation unfairly.

In previous evaluations of permutation methods (see e.g. [2, 3, 67, 69, 105]) projection/mapping quality is evaluated only indirectly. Specifically, it is measured what *fraction* of data points (closest to the query in the projected space) is necessary to retrieve to ensure a desired level of recall. This is a useful efficiency metric whose value is a reciprocal of the reduction in the number of distance computations. Moreover, it tells us how well the projection algorithm preserves neighborhood structure (see Exposition 2.9 for a clarifying example). However, this approach gives us only an "average" picture, which hides a lot of details. For this reason, we also visualize the relationship between the original distance values and corresponding values in the projected space. In particular, this permits a more accurate qualitative comparison against random projections, which is the classic projection approach in $L_2$.

In what follows, we describe the data sets, the implemented methods, and experimental setup in more detail. We then present experiments where we (1) evaluate the quality of permutation-based projections, (2) carry out an efficiency tests for a variety of methods. The section will be concluded with a discussion of experimental results and lessons learned from this evaluation.

#### 2.3.1.1 Data Sets and Distance Functions

We employ three image data sets: CoPhIR, SIFT, ImageNet, and several data sets created from textual data (DNA and Wikipedia). The smallest data set (DNA) has one million entries, while the largest one (CoPhIR) contains five million high-dimensional dense vectors. All data sets derived from Wikipedia are generated using the topic modelling library Gensim [250]. Data set properties (dimensionality, number of records, etc) are summarized in Table 2.4. Below, we describe our data sets in more detail. The data set can be downloaded using the following script: `https://github.com/searchivarius/nmslib/blob/master/data/get_data_vldb2015.sh`.

**CoPhIR** is a five million subset of 282-dimensional MPEG7 descriptors downloaded from

| Name | Distance function | # of rec. | Brute-force search (sec) | In-memory size | Dimens. | Source |
|---|---|---|---|---|---|---|
| | | | **Metric Data** | | | |
| CoPhIR | $L_2$ | $5 \cdot 10^6$ | 0.6 | 5.4GB | 282 | MPEG7 descriptors [39] |
| SIFT | $L_2$ | $5 \cdot 10^6$ | 0.3 | 2.4GB | 128 | SIFT descriptors [150] |
| ImageNet | SQFD[27] | $1 \cdot 10^6$ | 4.1 | 0.6 GB | N/A | Signatures generated from ImageNet LSVRC-2014 [257] |
| | | | **Non-Metric Data** | | | |
| Wiki-sparse | Cosine sim. | $4 \cdot 10^6$ | 1.9 | 3.8GB | $10^5$ | Wikipedia TF×IDF vectors generated via Gensim [250] |
| Wiki-8 | KL-div/JS-div | $2 \cdot 10^6$ | 0.045/0.28 | 0.13GB | 8 | 8-topic LDA [38] histograms generated from Wikipedia via Gensim [250] |
| Wiki-128 | KL-div/JS-div | $2 \cdot 10^6$ | 0.22/4 | 2.1GB | 128 | 128-topic LDA [38] histograms generated from Wikipedia via Gensim [250] |
| DNA | Normalized Levenshtein | $1 \cdot 10^6$ | 3.5 | 0.03GB | N/A | Sampled from the Human Genome[a] with sequence length $\mathcal{N}(32, 4)$ |

Table 2.4: Summary of Data Sets

[a] `http://hgdownload.cse.ucsc.edu/goldenPath/hg38/bigZips/`

the website of the Institute of the National Research Council of Italy[39].

**SIFT** is a five million subset of 128-dimensional SIFT descriptors [191] (from the learning subset) downloaded from a TEXMEX collection website[150].[14]

In experiments involving CoPhIR and SIFT, we use $L_2$ to compare unmodified, i.e., raw visual descriptors. We have implemented an optimized procedure to compute $L_2$ that relies on Single Instruction Multiple Data (SIMD) operations available on Intel-compatible CPUs. Using this implementation, it is possible to carry out about 30 million $L_2$ computations per second using SIFT vectors or about 10 million $L_2$ computations using CoPhIR vectors.

**ImageNet** collection comprises one million signatures (one signature per image) extracted from LSVRC-2014 data set of high resolution images [257].[15] One approach to represent an image is to extract multiple local features (e.g., SIFT descriptors [191]) and cluster them all using k-means. Thus, we can construct a *global* codebook, where local features are represented by distances to cluster centers. The global codebook is created to facilitate efficient $k$-NN search using a simple distance $L_2$ [15, 149]. We adopted an alternative signature-generation proposed by Beecks et al. [26, 27]. In this approach a separate, i.e., *local*, codebook (called a signature) is constructed for each image. To compare images, we compare their local codebooks using a distance function called the Signature Quadratic Form Distance (SQFD). SQFD is shown to be effective in an image retrieval task [26, 27]. Yet, it is nearly two orders of magnitude slower compared to $L_2$. We hope that experimenting with this expensive distance would help us understand strengths and weaknesses of various search methods.

Extraction of signatures involves the following steps: For each image, we first select $10^4$ pixels randomly and map them to 7-dimensional features (one feature per pixel): three color, two position, and two texture dimensions (contrast and coarseness). The features are clustered by the standard $k$-means algorithm with 20 clusters. Then, each cluster is represented by an 8-dimensional vector. The 8-dimensional vector is obtained by expanding a 7-dimensional cluster center with a cluster weight (the number of cluster points divided by $10^4$).

As noted above, images are compared using a distance function called the Signature Quadratic Form Distance (SQFD). This distance is computed as a quadratic form, where the matrix is re-computed for each pair of images using a heuristic similarity function applied to cluster representatives. SQFD is a distance *metric* defined over vectors from an *infinite*-dimensional space such that each vector has only finite number of non-zero elements. For further details, please, see the thesis of Beecks [26].

**Wiki-sparse** is a set of about four million sparse TF×IDF vectors. These vectors are created from Wikipedia text using Gensim [250]. Gensim retains only $10^5$ most frequent terms. Thus, dimensionality of TF×IDF vectors is $10^5$. However, these vectors are quite sparse: On average, there are only 150 non-zero elements. Here we use a cosine similarity, which is a symmetric non-metric distance (see Table 2.3). This collection is useful in checking how well various methods perform a on text retrieval task using a TF×IDF similarity.

Computation of the cosine similarity between sparse vectors relies on an efficient procedure to obtain an intersection of non-zero element indices. To this end, we use an all-against-all SIMD comparison instruction as was suggested by Schlegel et al. [265]. This distance function is

[14] `http://corpus-texmex.irisa.fr/`

[15] LSVRC-2014 contains 1.2 million, but we used only million.

| | VP-tree | | NAPP | | LSH | | Brute-force filt. | | $k$-NN graph | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Metric Data** | | | | | | | | | | |
| CoPhIR | 5.4 GB | (0.5m) | 6 GB | (6.8m) | 13.5 GB | (23.4m) | | | 7 GB | (52.1m) |
| SIFT | 2.4 GB | (0.4m) | 3.1 GB | (5m) | 10.6 GB | (18.4m) | | | 4.4 GB | (52.2m) |
| ImageNet | 1.2 GB | (4.4m) | 0.91 GB | (33m) | | | 12.2 GB | (32.3m) | 1.1 GB | (127.6m) |
| **Non-Metric Data** | | | | | | | | | | |
| Wiki-sparse | | | 4.4 GB | (7.9m) | | | | | 5.9 GB | (231.2m) |
| Wiki-8 (KL-div) | 0.35 GB | (0.1m) | 0.67 GB | (1.7m) | | | | | 962 MB | (11.3m) |
| Wiki-128 (KL-div) | 2.1 GB | (0.2m) | 2.5 GB | (3.1m) | | | | | 2.9 GB | (14.3m) |
| Wiki-8 (JS-div) | 0.35 GB | (0.1m) | 0.67 GB | (3.6m) | | | | | 2.4 GB | (89m) |
| Wiki-128 (JS-div) | 2.1 GB | (1.2m) | 2.5 GB | (36.6m) | | | | | 2.8 GB | (36.1m) |
| DNA | 0.13 GB | (0.9m) | 0.32 GB | (15.9m) | | | 61 MB | (15.6m) | 1.1 GB | (88m) |

**Note:** The indexing algorithms of NAPP and $k$-NN graphs uses four threads. Indexing time is in minutes.

In all but two cases (DNA and Wiki-8 with JS-divergence), we build the $k$-NN graph using the Small World algorithm [196].

In the case of DNA or Wiki-8 with JS-divergence, we build the $k$-NN graph using the NN-descent algorithm [98].

Table 2.5: Index Size and Creation Time for Various Data Sets

relatively fast being only about 5$\times$ slower compared to $L_2$.

**Wiki-D** consists of dense vectors of topic histograms with $D$ topics. These data sets are similar to collections of Cayton [58] and are created using the latent Dirichlet allocation (LDA)[38]. To create these sets, we randomly split Wikipedia into nearly equal two parts. We then train a model on one part of the collection and apply the model to the second part (using Gensim [250]). Zero values are replaced by small numbers ($10^{-5}$) to avoid division by zero in the distance calculations. We have generated data sets using LDA for $D = 8$ and $D = 128$ topics.

Because each topic histogram defines a discrete probability distribution, Wiki-D data sets are especially useful for testing statistical distances. In this section we present results for two statistical distances: KL-divergence and JS-divergence (see Table 2.3), which are non-metric distances. Note that the KL-divergence is not even symmetric.

Our implementation of the KL-divergence relies on the precomputation of logarithms at index time. Therefore, during retrieval it is as fast as $L_2$. In the case of JS-divergence, it is not possible to do so, because computation of JS-divergence requires computation of logarithms for previously unseen values (see also NMSLIB manual for technical details [219]). Thus, computation of JS-divergence is about 10-20$\times$ slower compared to $L_2$.

**DNA** is a collection of DNA sequences sampled from the Human Genome [16]. Starting locations are selected randomly and uniformly (however, lines containing the symbol `N` were excluded). The length of the sequence is sampled from $\mathcal{N}(32, 4)$. The distance function is the *normalized* Levenshtein distance. This non-metric distance is equal to the minimum number of basic edit operations[17]—necessary to convert one sequence into another—divided by the maximum of the sequence lengths.

[16] `http://hgdownload.cse.ucsc.edu/goldenPath/hg38/bigZips/`

[17] insertions, deletions, substitutions

#### 2.3.1.2 Tested Methods

Table 2.5 lists all implemented methods and provides information on index creation time and size.

**Multiprobe-LSH (MPLSH)** is implemented in the library LSHKit [18]. It is designed to work only for $L_2$. Some parameters are selected automatically using the cost model proposed by Dong et al. [97]. However, the size of the hash table $H$, the number of hash tables $L$, and the number of probes $T$ need to be chosen manually. We have previously found that (1) $L = 50$ and $T = 10$ provide a near optimal performance and (2) performance is not affected much by small changes in $L$ and $T$ [43]. This time, we have re-confirmed this observation by running a small-scale grid search in the vicinity of $L = 50$ and $T = 50$ (for $H$ equal to the number of points plus one). MPLSH generates a list of candidates that are directly compared against the query. This comparison involves the optimized SIMD implementation of $L_2$.

**VP-tree** is a classic space decomposition tree that recursively divides the space with respect to a randomly chosen pivot $\pi$[234, 293, 323]. Here, we use our modification of VP-tree that can work in non-metric spaces. In this modification, the pruning rule based on the triangle inequality is replaced by a parametric decision function learned from data as described in § 2.2.3. The parametric decision function is given by Eq. 2.11. Specifically, we used $\beta = 2$ for the KL-divergence and $\beta = 1$ for every other distance function. The size of the bucket is 50.

**$k$-NN graph** (a proximity graph) is a data structure in which data points are associated with graph nodes and sufficiently close nodes are connected by edges. A search algorithm relies on a concept "the closest neighbor of my closest neighbor is my neighbor as well." Constructing an *exact* $k$-NN graph is hardly feasible for a large high-dimensional data set, because, in the worst case, the number of distance computations is $O(n^2)$, where $n$ in the number of data points. An *approximate* $k$-NN graph can be constructed more efficiently.

In this work, we employ two different graph construction algorithms: the NN-descent proposed by Dong et al. [98] and the search-based insertion algorithm SW-graph proposed by Malkov et al. [196]. These algorithms are briefly outlined in § 2.2.2.3. Note that NN-descent creates a directed graph, while SW-graph creates an undirected one. We have incorporated an *open-source* implementation of NN-descent publicly available online.[19]. However, it comes without a search algorithm. For this reason, we create a $k$-NN graph using this open-source implementation of NN-descent, but the search process relies on the algorithm due to Malkov et al. [196], which is available in NMSLIB [42, 219]. [20] Both graph construction algorithms are computationally expensive and are, therefore, constructed in a multi-threaded mode (four threads).

For each data set, we use the best-performing graph-construction algorithm. To this end, we tune search methods on a subset of data. This process involves manual selection of parameters. In the case of SW-graph, we experiment with selecting the parameter `NN`, which indirectly influences the number of neighbors in a graph, as well as parameters `initSearchAttempts`, `initIndexAttempts` (see § 2.2.2.3). In this evaluation, we use the older implementation of

[18]Downloaded from `http://lshkit.sourceforge.net/`

[19]`https://code.google.com/p/nndes/`

[20]After results presented in this chapter were published, the author of NN-descent Wei Dong open-sourced an improved implementation of NN-descent (`https://github.com/aaalgo/kgraph`), which did have an implementation of the search algorithm. He also modified the indexing algorithm to create an undirected graph (apparently by adding reverse edges). In July 2016, this improved implementation of NN-descent was found to be comparable to SW-graph (check the following link).

**Exposition 2.10: Index size of NAPP**

As noted in § 2.2.2.4, the inverted index used in NAPP contains `numPivotIndex` data point identifiers per each data point. In our experiments, the value of this parameter ranges from 20 to 32. Without compression, each identifier can be encoded using four bytes. Thus, each data point accounts for 80-128 bytes in the inverted index. At the same time, for all data sets except DNA and Wiki-8, one data point uses at least 512 bytes (128 four-byte float values).

SW-graph in which the values of parameters `efSearch` and `efConstruction` are hardcoded to be equal to `NN`. In the case of NN-descent, we tune query-time parameters `efSearch` and `initSearchAttempts` as well as two additional index-time parameters: the number of neighbors in a graph and the decay coefficient. The latter parameter defines convergence speed.

**Brute-force filtering** is a simple approach where we exhaustively compare the permutation of the query against the permutation of every data point. We then use incremental sorting to select $\gamma$ permutations closest to the query permutation. These $\gamma$ entries represent candidate records compared directly against the query using the original distance.

As noted in § 2.2.2.4, the cost of the filtering stage is high. Thus, we use this algorithm only for computationally intensive distances: SQFD and the Normalized Levenshtein distance. Originally, both in the case of SQFD and Normalized Levenshtein distance, good performance is achieved with permutations of the size 128. However, Levenshtein distance is applied to DNA sequences, which are strings whose average length is only 32. Therefore, using uncompressed permutations of the size 128 is not space efficient (128 32-bit integers use 512 bytes). Fortunately, we can achieve the same performance using bit-packed binary permutations with 256 elements, each of which requires only 32 bytes.

The optimal permutation size is found by a small-scale grid search (again using a subset of data). Several values of $\gamma$ (understood as a fraction of the total number of points) have been manually selected to achieve recall in the range 0.85-0.9.

**NAPP** is a neighborhood approximation index described in § 2.2.2.4 [284]. The implementations of Chávez et al. [69] and Tellez et al. [283] is quite a bit complicated: They use a sophisticated compression algorithm and an adaptive algorithm to merge postings. However, for most data sets and distances, compression is not very useful, because the inverted index is already small compared to the memory footprint of data (see Exposition 2.10). In addition to removing compression, we employed a simpler algorithm, namely, *ScanCount*, to merge posting lists [182]

**Exposition 2.11: ScanCount algorithm**

For each entry in the database, there is a counter. When we read a posting list entry corresponding to the object $i$, we increment counter $i$. When all posting lists are read, we scan the list of counters and check if their values are above a given threshold (`numPivotSearch`). Before each search counters are zeroed using the function `memset` from a standard C library. To improve cache utilization and overall performance, we split the inverted index into small chunks, which are processed one after another.

(see Exposition 2.11).

To ensure that our modification has not had a negative effect on performance, we have compared our NAPP implementation to that of Chávez et al. [69] using the same $L_1$ data set: $10^6$ normalized CoPhIR descriptors. Due to differences in implementation and hardware used for testing (in particular, Chávez et al. [69] use C# and we use C++), a direct comparison of retrieval times would not be fair. Instead, we compare improvement in efficiency over brute-force search. Specifically, at 95% recall, Chávez et al. [69] achieve a 14× speed up, while we achieve a 15× speed up.

Tuning NAPP involves selection of three parameters `numPivot` (the total number of pivots), `numPivotIndex` (the number of indexed pivots), and `numPivotSearch` (the minimum number of indexed pivots that need to be shared between the query and a candidate data set point). By carrying out a small-scale grid search, we have found that increasing `numPivot` improves both recall and decreases retrieval time, yet, improvement is small beyond `numPivot`$= 500$. At the same time, computation of one permutation entails computation of `numPivot` distances to pivots. Thus, larger values of `numPivot` incur higher indexing cost. Values of `numPivot` between 500 and 2000 represent reasonable trade-offs. Because the indexing algorithm is computationally expensive, it is executed in a multi-threaded mode (four threads).

The fact that recall improves as we add more pivots is not surprising. Pivoting methods are effective only if comparing a query and an answer with the same pivot provides a meaningful information regarding proximity of the query and the data point. This happens only if at least one of the points is close to the pivot. Therefore, accuracy of NAPP crucially depends on existence of pivots that are close to both the query and its nearest neighbors. In that, the larger is the number of pivots, the higher is the probability that such pivots would exist. For the same reason, increasing the number of indexed pivots (i.e., `numPivotIndex`) improves recall. However, this increase makes NAPP less efficient: The larger is `numPivotIndex`, the more posting lists are to be processed at query time. We find that for our data sets good efficiency-effectiveness trade-offs are achieved for `numPivotIndex` in the range from 20 to 32.

Smaller values of `numPivotSearch` result in higher recall values, because decreasing `numPivotSearch` monotonically expands a set of candidate points. However, the increase in the number of candidate records negatively affects performance. Thus, for cheap distances, e.g. $L_2$, we manually select the smallest `numPivotSearch` that allows one to achieve a desired recall (using a subset of data). For more expensive distances, we have an additional filtering step (as proposed by Tellez et al. [283]), which involves sorting by the number of commonly indexed pivots.

Our initial assessment has showed that NAPP is more efficient than the PP-index and at least as efficient as MI-file. These findings agree with results of Chávez et al. [69]. Additionally we have compared NAPP against our own implementation of Fagin et al.'s OMEDRANK algorithm [106]. We have found NAPP to be substantially more efficient. We also experimented with indexing permutations using VP-tree. Yet, this approach is outperformed by either VP-tree that indexes data points directly or by NAPP.

#### 2.3.1.3 Experimental Setup

Experiments are carried out on an Linux Intel Xeon server (3.60 GHz, 32GB memory) in a single threaded mode using NMSLIB [42, 219] as an evaluation toolkit. The code is written in C++ and compiled using GNU C++ 4.8 with the `-Ofast` optimization flag. Additionally, we use the flag `-march=native` to enable SIMD extensions. All indices are loaded into memory. Specifically, each method is tested separately by a program that loads a data set, creates an index, and repeats evaluation for each unique set of *query-time* parameters A query-time parameter is a parameter that can be changed without rebuilding the index. For example, parameter `numPivotSearch` in NAPP and parameter `efSearch` in SW-graph are query-time parameters.

We evaluate performance of a 10-NN search using five trials. In each trial the data set is randomly split into two parts. The larger part is indexed. Data points from the smaller part are used as queries. The total number of queries is 5000 for cheap distances, and 100 for expensive ones (SQFD and normalized Levenshtein distance). To simplify our presentation, in the case of non-symmetric KL-divergence, we report results only for left queries. Results for right queries are similar.

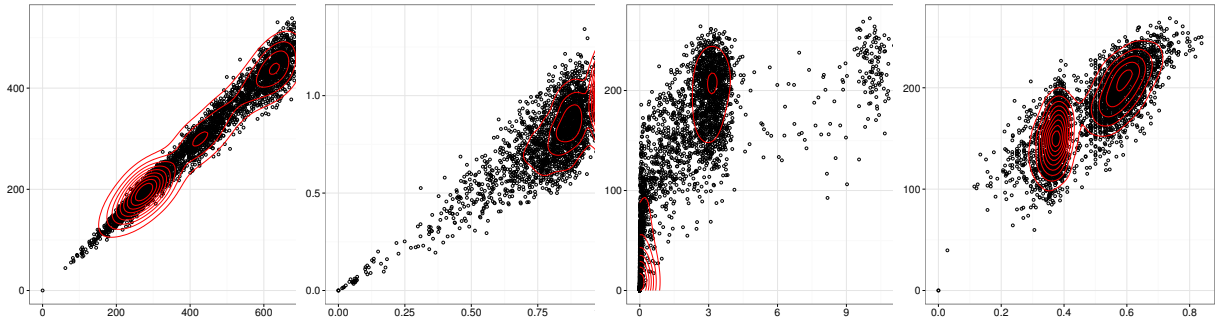
For each split, we evaluate retrieval performance by measuring the average retrieval time, the improvement in efficiency (compared to a single-thread brute-force search), recall, index creation time, and memory consumption. These performance metrics are aggregated over five splits. Note that comparing against the potentially slow brute-force search does not mean that we use brute-force search as baseline that we attempt to outstrip.

Because we are interested in high-accuracy (near 0.9 recall) methods, we have tuned parameters of the methods so that their recall falls in the range 0.85-0.95. Tuning is carried out using large subsets of data and randomly sampled subsets of queries, which are different from the queries used in final efficiency tests. Due to using different subsets of query and data points, it is impossible to ensure that the range of recall values for the complete set is strictly 0.85-0.95. Thus, the domains of recall values for various methods in Figure 2.11 do not overlap perfectly. Furthermore, because it is quite expensive to carry out experiments for all reasonable parameter values, we use manual tuning, which resembles coordinate ascent. Specifically, we select a parameter of interest and try to modify it by keeping all other parameter values constant.

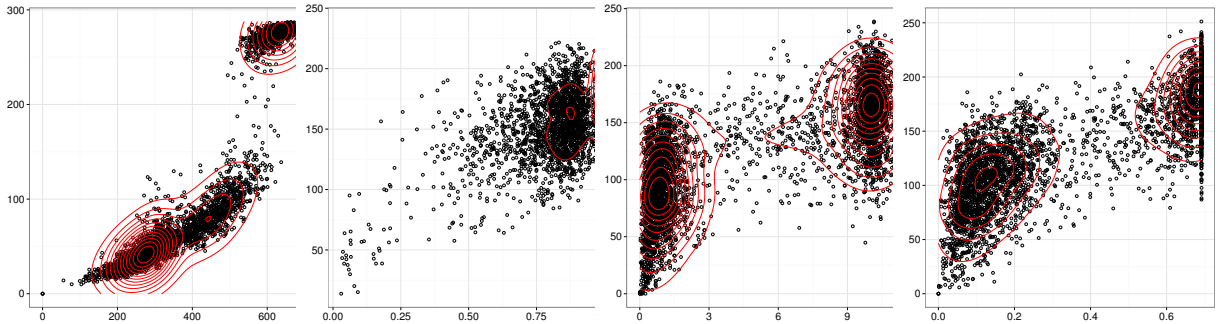
In most cases, such experiments reveal unambiguous regularities. For example, in SW-graph, increasing the value of query-time parameter `efSearch` always improves recall, but slows the search down. Similarly, decreasing `numPivotSearch` in NAPP leads to increase in recall at the expense of retrieval speed. In the case of brute-force filtering of permutations, increasing the size of the permutation has a positive effect on accuracy, but negative on recall. Tuning is less obvious for MPLSH, in particular, because some of the parameters are selected automatically by the MPLSH implementation. Hence, as noted in § 2.3.1.2, optimal parameters are found using a small-scale grid search. A more detailed description of method parameters is given in respective subsections of § 2.3.1.2.

#### 2.3.1.4 Quality of Permutation-Based Projections

Recall that permutation methods are filter-and-refine approaches that map data from the original space to $L_2$ or $L_1$. Their accuracy depends on the quality of this mapping, which we assess in



(a) SIFT ($L_2$) **rnd-proj** (b) Wiki-sparse (cos.) **rnd-proj** (c) Wiki-8 (KL-div) **perm** (d) DNA (norm. Leven.) **perm**



(e) SIFT ($L_2$) **perm** (f) Wiki-sparse **perm** (g) Wiki-128 (KL-div) **perm** (h) Wiki-128 (JS-div) **perm**

Figure 2.9: Distance values in the projected space (on the y-axis) plotted against original distance values (on the x-axis).

Plots 2.9a and 2.9b use random projections. The remaining plots rely on permutations. Dimensionality of the target space is 64. All plots except Plot 2.9b represent projections to $L_2$. In Plot 2.9b the target distance function is the cosine similarity. Distances are computed for pairs of points sampled at random. Sampling is conducted from two strata: a complete subset and a set of points that are 100 nearest neighbors of randomly selected points. All data sets have one million entries.

this subsection. To this end, we explore (1) the relationship between the original distance values and corresponding values in the projected space, (2) the relationship between the recall and the fraction of candidate records scanned in response to a query.

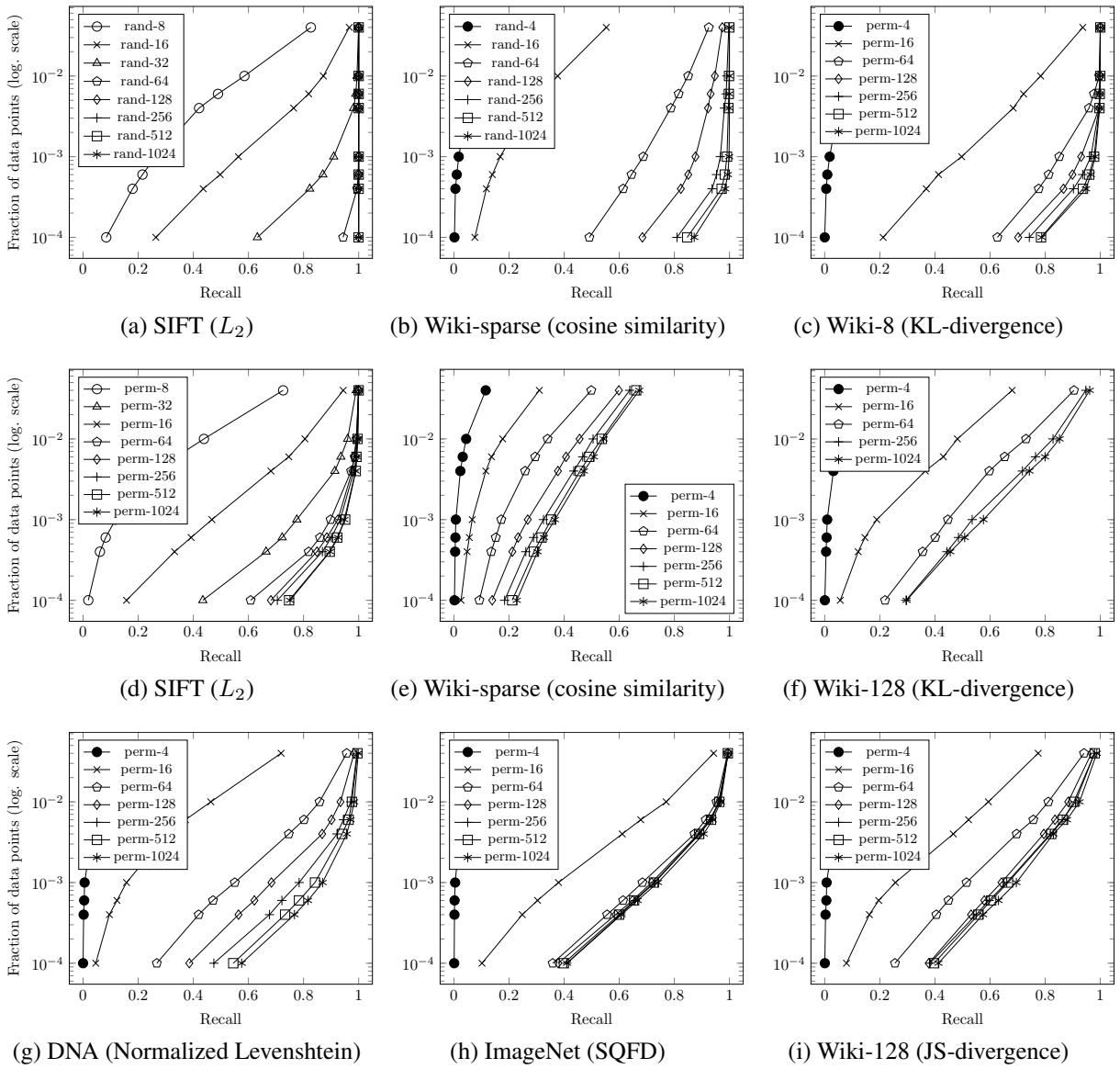
Figure 2.9 shows distance values in the original space (on the x-axis) vs. values in the projected space (on the y-axis) for eight combinations of data sets and distance functions. The plots are obtained from subsets of data containing one million data points each. Points are randomly sampled from two strata. One stratum includes the complete subset of one million points. The other stratum is a set of points that are 100 nearest neighbors of randomly selected points. Of the presented panels, 2.9a and 2.9b correspond to the classic random projections. The remaining panels show permutation-based projections.

Classic random projections are known to preserve inner products and values of the Euclidean distance [37]. Indeed, the relationship between the distance in the original and the projected space appears to be approximately linear in Panels 2.9a and 2.9b. Therefore, it preserves the relative distance-based order of points with respect to a query. For example, there is a high likelihood for the nearest neighbor in the original space to remain the nearest neighbor in the projected space. In principle, any monotonic relationship—not necessarily linear–will suffice [271]. If the monotonic relationship holds at least approximately, the projection typically distinguishes between points close to the query and points that are far away.

For example, the projection in Panel 2.9e appears to be quite good, which is not entirely surprising, because the original space is Euclidean. The projections in Panels 2.9h and 2.9d are also reasonable, but not as good as one in Panel 2.9e. The quality of projections in Panels 2.9f and 2.9c is somewhat uncertain. The projection in Panel 2.9g–which represents the non-symmetric and non-metric distance–is obviously poor. Specifically, there are two clusters: one is close to the query (in the original distance) and the other is far away. However, in the projected space these clusters largely overlap.

Figure 2.10 contains nine panels that plot recall (on x-axis) against a fraction of candidate records necessary to retrieve to ensure this recall level (on y-axis). In each plot, there are several lines that represent projections of different dimensionality. Good projections (e.g., random projections in Panels 2.10a and 2.10b) correspond to curves, where recall approaches one even for a small fraction of candidate records retrieved. "Goodness" depends on the dimensionality in the target spaces. However, good projection curves quickly reach high recall values even in relatively low dimensions.

The worst projection according to Figure 2.9 is in Panel 2.9g. It corresponds to the Wiki-128 data set with distance measured by KL-divergence. Panel 2.10f in Figure 2.10, corresponding to this combination of the distance and the data set, also confirms the low quality of the projection. For example, given a permutation of dimensionality 1024, scanning 1% of the candidate permutations achieves roughly a 0.9 recall. An even worse projection example is in Panel 2.10e. In this case, regardless of the dimensionality, scanning 1% of the candidate permutations achieves recall below 0.6.



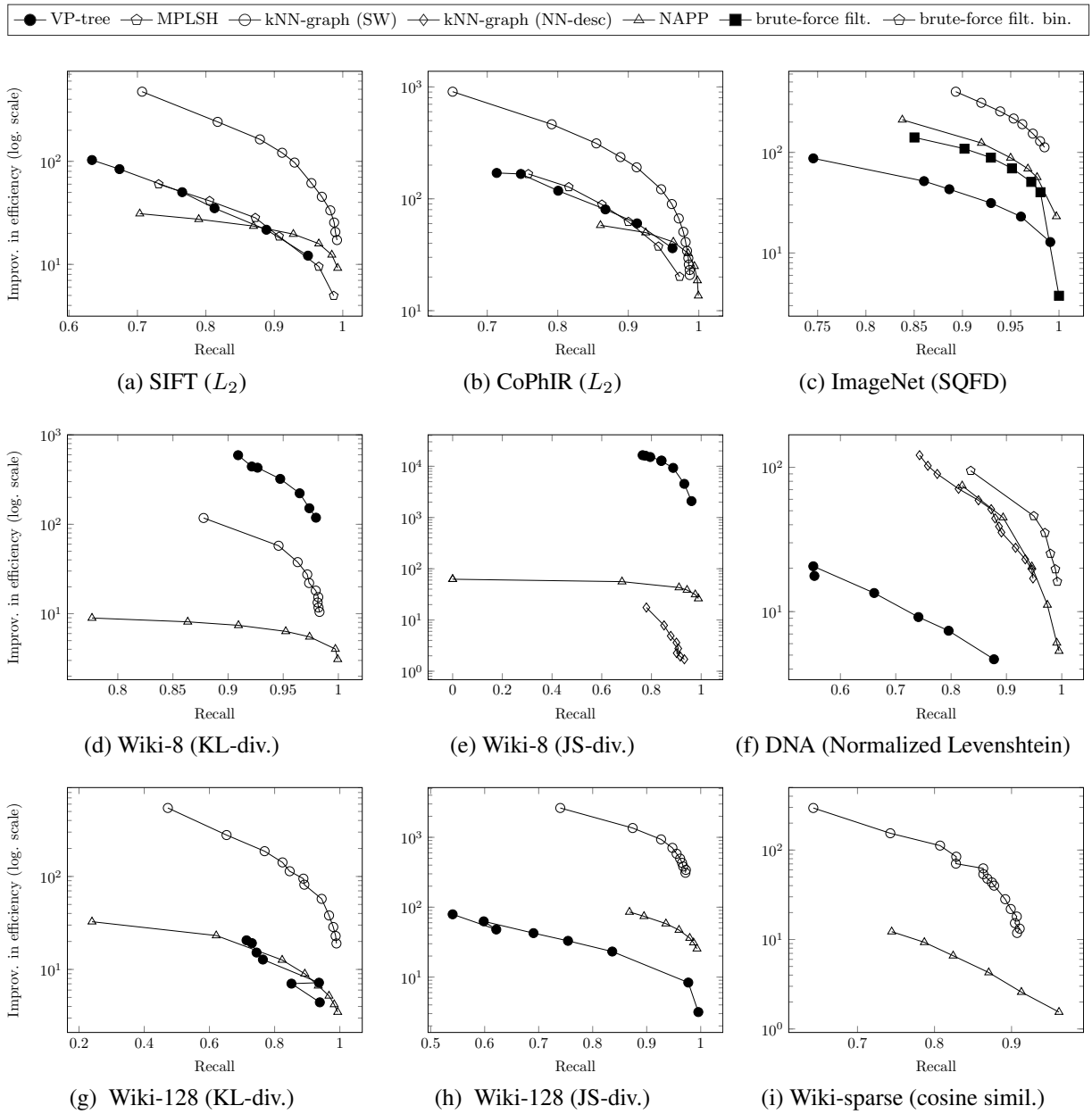

(a) SIFT ($L_2$) (b) Wiki-sparse (cosine similarity) (c) Wiki-8 (KL-divergence)

(d) SIFT ($L_2$) (e) Wiki-sparse (cosine similarity) (f) Wiki-128 (KL-divergence)

(g) DNA (Normalized Levenshtein) (h) ImageNet (SQFD) (i) Wiki-128 (JS-divergence)

Figure 2.10: A fraction of candidate records (on the y-axis) that are necessary to retrieve to ensure a desired recall level (on the x-axis) for 10-NN search.

The candidate entries are ranked in a projected space using either the cosine similarity (only for Wiki-sparse) or $L_2$ (for all the other data sets). Two types of projections are used: random projections (rand) and permutations (perm). In each plot, there are several lines that represent projections of different dimensionality. Each data (sub)set in this experiment contains one million entries.




(a) SIFT ($L_2$) (b) CoPhIR ($L_2$) (c) ImageNet (SQFD)

(d) Wiki-8 (KL-div.) (e) Wiki-8 (JS-div.) (f) DNA (Normalized Levenshtein)

(g) Wiki-128 (KL-div.) (h) Wiki-128 (JS-div.) (i) Wiki-sparse (cosine simil.)

Figure 2.11: Improvement in efficiency (on the y-axis) vs. recall (on the x-axis) on various data sets for 10-NN search.

Each plot includes one of the two implemented $k$-NN graph algorithms: Small World (SW) or NN-descent (NN-desc).

At the same time, for majority of projections in other panels, scanning 1% of the candidate permutations of dimensionality 1024 achieves an almost perfect recall. In other words, for some data sets, it is, indeed, possible to obtain a small set of candidate entries containing a true near-neighbor by searching in the permutation space.

#### 2.3.1.5 Evaluation of Efficiency vs Recall

In this section, we use complete data sets listed in Table 2.4. Figure 2.11 shows nine data set specific panels with improvement in efficiency vs. recall. Each curve captures method-specific results with parameter settings tuned to achieve recall in the range of 0.85-0.95.

It can be seen that in most data sets the permutation method NAPP is a competitive baseline. In particular, Panels 2.11a and 2.11b show NAPP outperforming the state-of-the art implementation of the multi-probe LSH (MPLSH) for recall larger than 0.95. This is consistent with findings of Chávez et al. [69].

In that, in our experiments, there was no single best method. $k$-NN graphs substantially outperform other methods in 6 out of 9 data sets. However, in low-dimensional data sets shown in Panels 2.11d and 2.11e, VP-tree outperforms the other methods by a wide margin. The Wiki-sparse data set (see Panel 2.11i), which has high representational dimensionality, is quite challenging. Among implemented methods, only $k$-NN graphs are efficient for this set.

Interestingly, the winner in Panel 2.11f is a brute-force filtering using binarized permutations. Furthermore, the brute-force filtering is also quite competitive in Panel 2.11c, where it is nearly as efficient as NAPP. In both cases, the distance function is computationally intensive and a more sophisticated permutation index does not offer a substantial advantage over a simple brute-force search in the permutation space.

Good performance of $k$-NN graphs comes at the expense of long indexing time. For example, it takes almost four hours to built the index for the Wiki-sparse data set using as many as four threads (see Table 2.5). In contrast, it takes only 8 minutes in the case of NAPP (also using four threads). In general, the indexing algorithm of $k$-NN graphs is substantially slower than the indexing algorithm of NAPP: it takes up to an order of magnitude longer to build a $k$-NN graph. One exception is the case of Wiki-128 where the distance is the JS-divergence. For both NAPP and $k$-NN graph, the indexing time is nearly 40 minutes. However, the $k$-NN graph retrieval is an order of magnitude more efficient.

Compared to VP-tree, both NAPP and brute-force searching of permutations have high indexing costs. These indexing costs are often dominated by time spent on computing permutations. Recall that obtaining a permutation entails $m$ distance computations. Thus, building an index entails $N \cdot m$ distance computations, where $N$ is the number of data points.

We observe that all evaluated methods perform reasonably well in the case of non-metric data. Under close inspection, all of the distances possess a nice property: Their monotonic transformation is $\mu$-defective, see Eq (2.3).

Indeed, a monotonic transformation of the cosine similarity is the metric function (namely, the angular distance) [295]. The square root of the JS-divergence is metric function called Jensen-Shannon distance [103]. The square root of all Bregman divergences (which include the KL-divergence) is $\mu$-defective as well [1]. The normalized Levenshtein distance is a non-metric distance. However, for many realistic data sets, the triangle inequality is rarely violated. In

| Name | # of rec. | Dimens. | Source |
|---|---|---|---|
| RandHist-8 | $0.5 \cdot 10^6$ | 8 | Histograms sampled uniformly from a simplex |
| RCV-8 | $0.5 \cdot 10^6$ | 8 | 8-topic LDA [38] RCV1 [180] histograms |
| Wiki-8 | $2 \cdot 10^6$ | 8 | 8-topic LDA [38] Wikipedia histograms |
| RandHist-32 | $0.5 \cdot 10^6$ | 32 | Histograms sampled uniformly from a simplex |
| RCV-128 | $0.5 \cdot 10^6$ | 128 | 128-topic LDA [38] RCV1 [180] histograms |
| Wiki-128 | $2 \cdot 10^6$ | 128 | 128-topic LDA [38] Wikipedia histograms |
| Manner | $1.46 \cdot 10^5$ | $123 \cdot 10^5$ | TF×IDF vectors created from Yahoo Answers *Manner* [280] (see §3.1 for details) |

Table 2.6: Data sets used in auxiliary experiments

Random histograms are created by randomly sampling from the exponential distribution with subsequent normalization.[a]. Wikipedia topic histograms are generated using Gensim [250].
To generate Yahoo Answers TF×IDF vectors, we first create an in-memory index using a pipeline described in § 3.1. Then, we use a special conversion utility to generate sparse vectors in text format.[b].

[a] `https://github.com/searchivarius/nmslib/blob/master/data/genhist_unif.py`
[b] `https://github.com/oaqa/knn4qa/blob/bigger_reruns/scripts/data/run_extr_query_doc_vect.sh`

particular, we verified that this is the case of our data set. The normalized Levenshtein distance is approximately metric and, thus, it is approximately $\mu$-defective (with $\mu = 1$).

Formally, this means that there exists a continuous, non-negative and strictly monotonic transformation $f(x) \geq 0$, $f(0) = 0$ such that $f(d(x, y))$ is a $\mu$-defective distance function, see Eq (2.3). Thus, the distance satisfies the following inequality:

$$|f(d(q, a)) - f(d(q, b))| \leq \mu f(d(a, b)), \mu > 0 \tag{2.19}$$

If Inequality (2.19) holds, due to properties of $f(x)$, $d(a, b) = 0$ and $d(q, a) = 0$ implies $d(q, b) = 0$. Similarly if $d(q, b) = 0$, but $d(q, a) \neq 0$, $d(a, b)$ cannot be zero either. Moreover, for a sufficiently large $d(q, a)$ and small $d(q, b)$, $d(a, b)$ cannot be small.

This, in turn, means that all our distances have another nice albeit informal extended neighbor-of-my-neighbor property (see p. 16). This property ensures that (1) two points close to some third point are close to each other; and (2) points cannot be close to each other if one is close to a third point, but another is far. We believe that fulfillment of these informal properties is key to performance of all our methods especially for SW-graph and NAPP.

### 2.3.2 Experiments with Challenging Distances

Here, we use an experimental setup, which differs in several ways from the our main experiment setup. First, we use different data sets with a focus on hard-to-tackle statistical distances. By measuring how effective are symmetrization approaches, we can identify cases where distances are substantially non-symmetric. Demonstrating good performance in such cases is crucial to empirical verification of methods' generality.

| Denotation/Name | d(x,y) | Notes |
|---|---|---|
| Symmetric distances | | |
| $L_2^2$ (Squared Euclidean) | $\sum_{i=1}^{m}(x_i - y_i)^2$ | |
| $L_p$ | $\left[\sum_{i=1}^{m}(x_i - y_i)^p\right]^{1/p}$ | $p \in 0.125, 0.25, 0.5$ |
| Cosine distance | $1 - \frac{\langle x, y\rangle}{\|\|x\|\|_2 \|\|y\|\|_2}$ | |
| Non-symmetric distances | | |
| Kullback-Leibler diverg. (KL-div.) [171] | $\sum_{i=1}^{m} x_i \log \frac{x_i}{y_i}$ | |
| Itakura-Saito distance [143] | $\sum_{i=1}^{m}\left[\frac{x_i}{y_i} - \log\frac{x_i}{y_i} - 1\right]$ | |
| Rényi diverg. [251] | $\frac{1}{\alpha-1}\log\left[\sum_{i=1}^{m} p_i^{\alpha} q_i^{1-\alpha}\right]$ | $\alpha \in 0.25, 0.75, 2$ |
| BM25 similarity [254] | See § 3.1.1.1 for details | |

Table 2.7: Potentially Challenging Distance Functions

Thus, in addition to KL-divergence, we employ the following distance functions:

- the Itakura-Saito distance;
- Rényi divergence for three values of the parameter $\alpha$;
- the squared Euclidean distance $L_2^2$;
- the $L_p$ distance with three values of parameter $p$ ($p < 1$);
- The TF×IDF similarity BM25.

Note that *all* the employed distances are non-metric and most of them are not symmetric. In particular, note that $L_p$ distances are metrics only when $p \geq 1$, while we employ them only for $p < 1$. With respect to BM25 and other TF×IDF -based distances, as we explain on p. 19, these distances are not symmetric, because swapping arguments $x$ and $y$ in $d(x, y)$ usually changes the value of the distance. However, the respective top-$k$ retrieval problem can be easily recast as a maximum inner product search, i.e., as a search problem with a symmetric distance. This trick, which can be seen a natural way to symmetrize the non-metric distance BM25, (in addition to other simple symmetrization approaches) is discussed in § 2.3.2.4 in more detail. There, we also study the effect of distance symmetrization applied at different stages of $k$-NN search.

The complete list of employed distances and data sets is presented in Table 2.7 and 2.6, respectively. Note, however, that we do not use all the combinations of distances and data sets in a single experiment. Furthermore, because experimenting with the full gamut of complete data sets can be quite expensive, we often use only subsets. More specific details regarding the choice of the data sets and respective distances are provided in respective subsections § 2.3.2.2-2.3.2.4.

| | RCV-8 | | Wiki-8 | | RandHist-8 | | Wiki-128 | |
|---|---|---|---|---|---|---|---|---|
| | Recall | Impr. in eff. | Recall | Impr. in eff. | Recall | Impr. in eff. | Recall | Impr. in eff. |
| $L_p(p = 0.125)$ | 0.41 | 1065 | 0.66 | 15799 | 0.45 | 136 | 0.07 | 14845 |
| $L_p(p = 0.25)$ | 0.61 | 517 | 0.78 | 14364 | 0.66 | 115 | 0.09 | 396 |
| $L_p(p = 0.5)$ | 0.91 | 926 | 0.94 | 14296 | 0.92 | 174 | 0.50 | 33 |
| $L_2^2$ | 0.69 | 1607 | 0.78 | 5605 | 0.56 | 1261 | 0.55 | 114 |
| Cosine dist. | 0.67 | 1825 | 0.62 | 3503 | 0.58 | 758 | 0.73 | 55 |
| Rényi div. ($\alpha = 0.25$) | 0.66 | 5096 | 0.70 | 24246 | 0.50 | 3048 | 0.48 | 1277 |
| Rényi div. ($\alpha = 0.75$) | 0.61 | 9587 | 0.66 | 35940 | 0.50 | 4673 | 0.50 | 468 |
| Rényi div. ($\alpha = 2$) | 0.40 | 22777 | 0.66 | 46122 | 0.38 | 11762 | 0.71 | 55 |
| KL-div. | 0.52 | 1639 | 0.67 | 5271 | 0.46 | 610 | 0.56 | 41 |
| Itakura-Saito | 0.46 | 706 | 0.69 | 4434 | 0.41 | 1172 | 0.14 | 384 |

Table 2.8: Efficiency-effectiveness results for metric VP-tree on non-metric data for 10-NN search (using complete data sets).

#### 2.3.2.1 Evaluating Complexity of the Problem

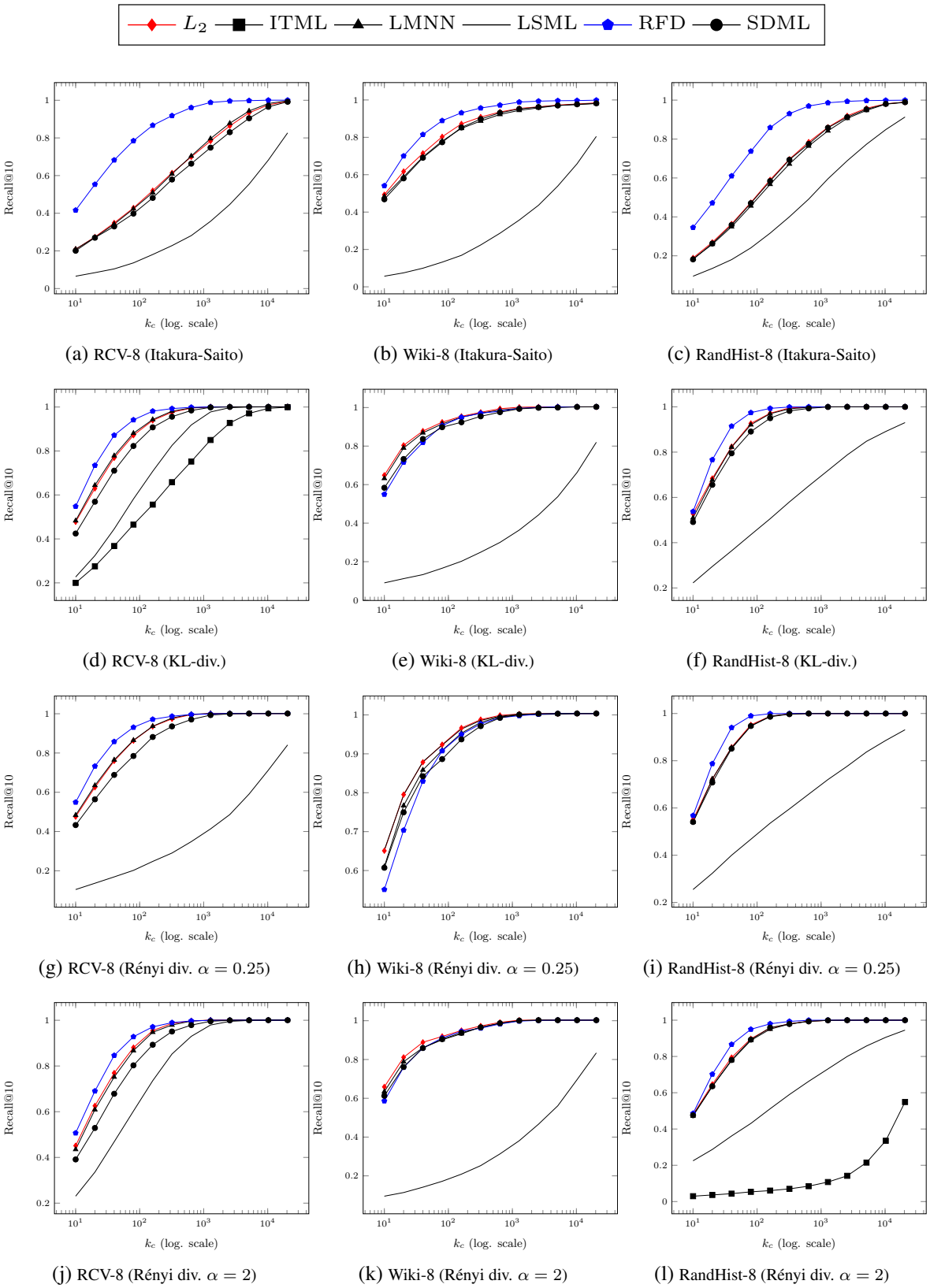
In this section, we evaluate the hardness of our data sets by using simple reductions to a search problem where a distance is a symmetric and/or a metric distance. First, in § 2.3.2.2, we pretend that we deal with a metric distance and carry out indexing and searching using a classic metric VP-tree [234, 293, 323]. If such an application resulted in an efficient yet accurate search procedure, we clearly dealt only with a mildly non-metric data sets.[21] In § 2.3.2.3 and § 2.3.2.4, we assess an effectiveness of the approach, where the original hard-to-tackle distance is replaced with an "easier" proxy distance. To this end, we experiment with several symmetrization and metric-learning methods.

#### 2.3.2.2 Can We Simply Use a Metric VP-tree?

In this subsection, we simply apply the classic metric VP-tree [234, 293, 323] to a variety of data sets. Experiments are run on our personal laptop (i7-4700MQ @ 2.40GHz with 16GB of memory). For each data set, we use 1K randomly selected data points as queries (these queries are removed from the data set). We employ four complete data sets and ten distance functions. Because VP-tree works well only for low-dimensional data, three out of four data sets are low-dimensional (8 dimensions each).

In Table 2.8, we show improvement in efficiency and the respective recall. As a reminder, here we use a standard pruning rule defined by the triangle inequality. Hence, unlike our modification of the VP-tree, where we can obtain various efficiency-effectiveness trade-offs by modifying the pruning rule (see § 2.2.3), each experiment with a standard metric VP-tree is represented by a single point in the efficiency-effectiveness space.

[21] We thank Karina Figueroa Mora for proposing this set of experiments.




(a) RCV-8 (Itakura-Saito) (b) Wiki-8 (Itakura-Saito) (c) RandHist-8 (Itakura-Saito)

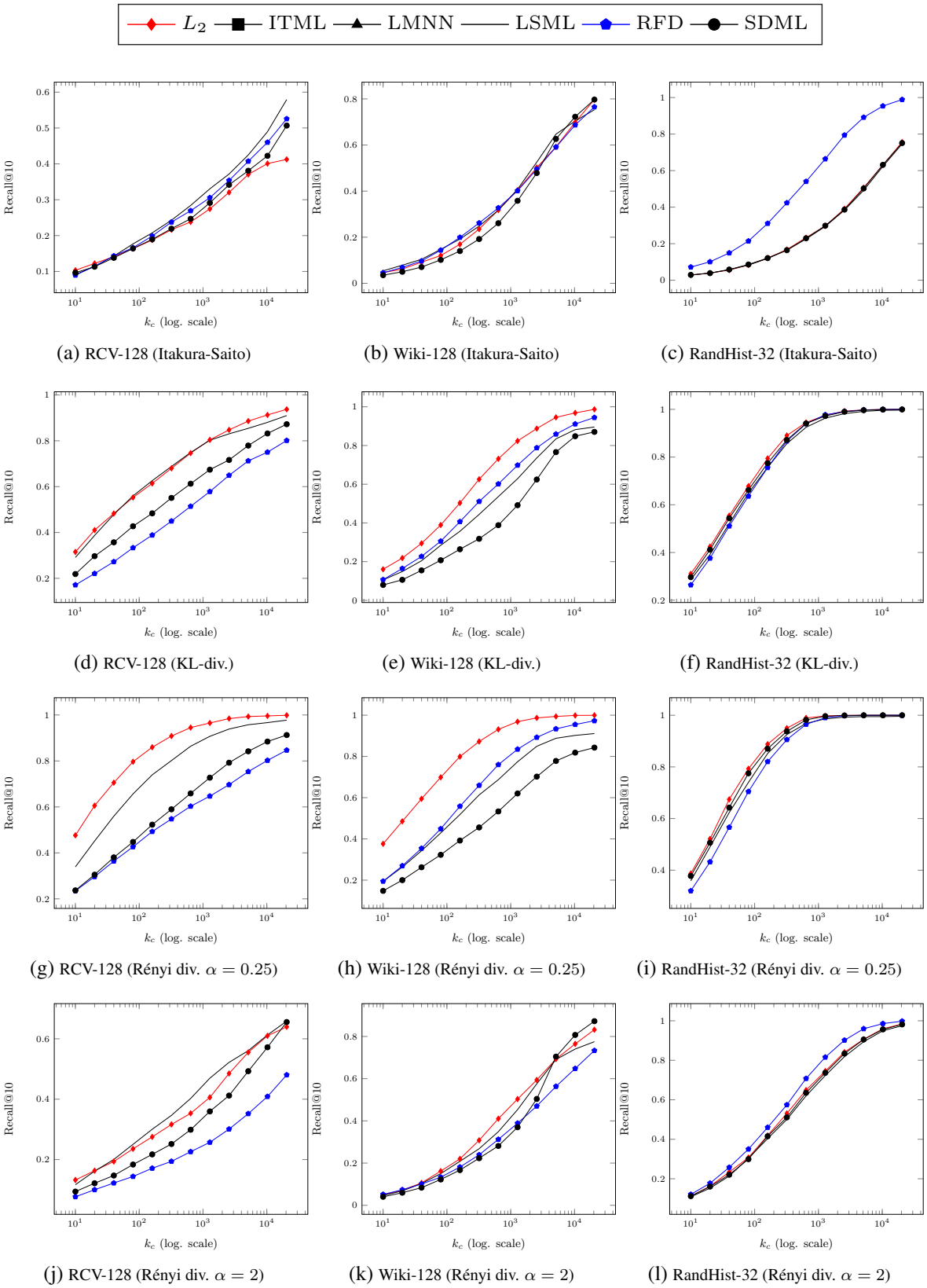
(d) RCV-8 (KL-div.) (e) Wiki-8 (KL-div.) (f) RandHist-8 (KL-div.)

(g) RCV-8 (Rényi div. $\alpha = 0.25$) (h) Wiki-8 (Rényi div. $\alpha = 0.25$) (i) RandHist-8 (Rényi div. $\alpha = 0.25$)

(j) RCV-8 (Rényi div. $\alpha = 2$) (k) Wiki-8 (Rényi div. $\alpha = 2$) (l) RandHist-8 (Rényi div. $\alpha = 2$)

Figure 2.12: Effectiveness of distance learning methods for 10-NN search (part I). Best viewed in color.

The number of data points is 200K. The x-axis displays the number of candidate entries $k_c$ necessary to achieve a given level of recall.




(a) RCV-128 (Itakura-Saito) (b) Wiki-128 (Itakura-Saito) (c) RandHist-32 (Itakura-Saito)

(d) RCV-128 (KL-div.) (e) Wiki-128 (KL-div.) (f) RandHist-32 (KL-div.)

(g) RCV-128 (Rényi div. $\alpha = 0.25$) (h) Wiki-128 (Rényi div. $\alpha = 0.25$) (i) RandHist-32 (Rényi div. $\alpha = 0.25$)

(j) RCV-128 (Rényi div. $\alpha = 2$) (k) Wiki-128 (Rényi div. $\alpha = 2$) (l) RandHist-32 (Rényi div. $\alpha = 2$)

Figure 2.13: Effectiveness of distance learning methods for 10-NN search (part II). Best viewed in color.

The number of data points is 200K. The x-axis displays the number of candidate entries $k_c$ necessary to achieve a given level of recall.

| Mnemonic name | Metric? | Description | Parameters (if different from default) |
|---|---|---|---|
| $L_2$ | ✓ | This quasi-learning method simply computes $L_2$ between vector representations. | |
| LMNN | ✓ | Large Margin Nearest Neighbor (LMNN) metric learning obtains a linear space transformation $G$ of the original vector space (see Eq. 2.9) via a convex optimization problem with a hinge loss [313]. | k=10 learn_rate=$10^{-6}$ max_iter=1000 |
| ITML | ✓ | Information Theoretic Metric Learning (ITML) obtains a Mahalanobis matrix $A$ (see Eq. 2.8) via minimizing the LogDet divergence subject to linear constraints [87]. | num_constraints=$250^2$ max_iter=1000 |
| SDML | ✓ | Sparse Determinant Metric Learning (SDML) obtains a sparse Mahalanobis matrix $A$ (see Eq. 2.8) via a $L_1$-penalized log-determinant regularization [248]. | num_constraints=$250^2$ |
| LSML | ✓ | Least Squares Metric Learning (LSML) obtains a Mahalanobis matrix $A$ (see Eq. 2.8) that minimizes a convex objective function corresponding to the sum of squared residuals of constraints [188]. | num_constraints=$250^2$ max_iter=1000 |
| RFD | ✗ | Random Forest Distance (RFD) employs random forests to train a classifier distinguishing between near and far neighbors. The set of predictors is composed of the scalar $\|\|x-y\|\|_2$ and the vector $(x+y)/2$ [317]. | # of constr. per class: 400 ntrees=500 iminnodesize=5 |

Table 2.9: Distance learning methods

As we can see from Table 2.8, nearly all the combinations of data and distance functions are substantially non-metric: Searching using a metric VP-tree is usually fast, but the accuracy is low. In particular, this is true for Wiki-8 and Wiki-128 data sets with KL-divergence, which are used in the experimental evaluation of § 2.3.1. One exception, is the $L_p$ distance for $p = 0.5$, where recall of about 90% is achieved for three low-dimensional data sets. However, as $p$ decreases, the recall decreases sharply, i.e., the distance function becomes "less" metric. To summarize, we clearly deal with challenging non-metric data sets, where both accurate and efficient retrieval is not possible to achieve by a straightforward application of metric-space search methods.

#### 2.3.2.3 Experiments with Distance Learning

In this section, we assess the effect of replacing the original distance with a learned one. The learning methods are listed in Table 2.9. The first method replaces the original distance with the Euclidean distance between vector representations. It is clearly *not* a distance learning method, but we use it as a baseline.

The remaining methods are true learning methods, which (in all but one case) learn a global linear transformation of the data (Mahalanobis metric learning). The value of the Euclidean distance between transformed vectors is used as a proxy distance function. The space transformation is trained as a classifier that learns to distinguish between close and distant data points.

To this end, we select a number of groups $C_i$ of close objects (further details are given below), each of which represents a single class. Given this set of objects, each method creates a training set of positive and negative examples. A positive example set contains pairs of points that should be treated as similar, i.e., near points, while the negative example set contains pairs of points that should be treated as distinct. The underlying idea is to learn a distance that (1) pulls together points from the positive example set and (2) pushes points from the negative example set apart.

A positive example set contains data point pairs $(x, y)$ within the same class $C_i$ ($x \in C_i,\ y \in C_i$), while a negative example set contains pairs $(x, z)$ from different classes ($x \in C_i,\ z \in C_j$ such that $i \neq j$). Methods differ in deciding which positive and negative example pairs are included into the training set. For example, in Large Margin Nearest Neighbor (LMNN) metric learning [313], for each data point $x$, there is $k$ positive example pairs $(x, y_i)$, where $\{y_i\}$ are $k$ nearest neighbors of $x$. A negative example set includes all pairs $(x, z)$, where $z$ belongs to a different class than $x$ ($x \in C_i,\ z \in C_j$ such that $i \neq j$).[22] These sets can be used to formulate a loss function, which typically aims to penalize large distances for positive examples and small distances for negative examples. The training set in LMNN can be quite large. Many other methods have a limit on the total number of positive/negative examples or on the number of examples per class $C_i$.

In addition to Mahalanobis distance learning methods, which learn a true metric via linear space transformation, we also experiment with a non-metric distance learning method called Random Forest Distance (RFD), which is shown to outperform Mahalanobis distances on a number of data sets [317]. RFD also operates by creating a set of constraints. Then, it trains a random forests classifier to distinguish between near and far neighbors. To this end, it uses a set of predictors composed of the scalar $||x - y||_2$ and the vector $(x + y)/2$, where $(x, y)$ is either a positive or a negative example pair. Unlike a Mahalanobis learning method, which learns a global linear transformation, RFD attempts to learn a distance that adapts to local structure by including the midpoint of a segment connecting $x$ and $y$ into the set of predictors. Although RFD is symmetric, it is not necessarily a metric distance [317].

Distance learning methods are tailored to a classification (or learning-to-rank) task, where classes $C_i$ are specified by the user. In contrast, we have the actual distance with no extrinsically specified classes of similar and dissimilar data points. To recast a distance learning problem into a classification problem, we obtain well separated $k$-neighborhoods via rejection sampling. Elements of one neighborhood are placed into a single class $C_i$. To ensure good separation, we require that $k'$-neighborhoods ($k' > k$) of sampled points do not overlap.

[22] More details can be inferred from Eq. (11) and Eq. (12) in corresponding respective paper [313].

The sampling process is guided by two parameters: the size of the neighborhood $k$ and the size of the "padded" neighborhood $k' > k$. The sampling process first selects a random query point $q$ and finds its *exact* $k$-neighborhood via the brute-force search. Data points that are $k$-nearest neighbors of the query are placed into the same class $C_i$, unless the $k'$-neighborhood of the query $k$ has a non-empty intersection with already selected points or their $k$-th nearest neighbors. In this case we reject $q$, because its $k$-neighborhood is not well separated from $k$-neighborhoods of other training points. The test set is created by randomly selecting query points that are different from previously selected queries. For each test query, we also compute its exact $k$-neighborhood. However, we do not require it to be separated from $k$-neighborhoods of other queries.

For efficiency reasons, there are trade-offs to be made with respect to the following parameters:

- The number of data points;
- The number of training query points;
- The number of iterations in the rejection sampling process (before we give up to find another query with a well-separated neighborhood);
- The size of the padded neighborhood $k'$.

Using larger data sets makes it easier to obtain well-separated neighborhoods. However, it also increases the overall testing time. Increasing the maximum number of iterations in rejection sampling makes it more likely that a training data set is created successfully. Yet, it also increases training times. Furthermore, we would like to select $k'$ to be much larger than $k$. However, only few data points $k'$-neighborhoods are well separated when $k'$ is large.

In summary, we tweaked parameters by increasing the size of the data set and the maximum number of iterations (up to 400) as well as by decreasing $k'$ (down to 20) until we were able to complete experiments in about one week. As in other experiments of this section, we select $k = 10$. The number of data points is equal to 200K. Both the training and the testing sets have 250 queries. Despite the number of queries is small, a learning method has to process a sizable training set, because of the large number of constraints per query. For example, for RFD, the overall number of positive and negative example pairs is 100K.

At test time, we measure how well the learned distance preserves original neighborhoods. To this end, we carry out an *exact brute-force* $k_c$-NN search using the learned distance, which return the list of $k_c$ candidates (the number of candidates $k_c = k \cdot 2^i$, where $i$ ranges from zero to 11). We than compute recall, which is equal to the relative overlap between the set of $k_c$ candidates and the original $k$-neighborhood of the query. The more accurately the learned distance "mimics" the original one, the fewer is number of candidates $k_c$ is required to achieve high recall. In the best case $k_c = k$, which means that the learned distance ideally preserves the structure of original neighborhoods. An analogous experiment is carried out in § 2.3.1.4 to evaluate the quality of projections. In addition, to the quality of the learned distance, we measure a processing time of the query.

Experiments are carried on an *r3* Amazon EC2 instance, which has two virtual cores and 15GB of RAM. The code is implemented as a Python 2.7 script. Mahalanobis distance learning is provided by the metric-learn library [56], which accelerates computation using Numpy [294] and SciPy [154]. We also use element-wise Numpy's vector operations to implement the brute-force search. RFD is provided by its author Caiming Xiong [317]: It is implemented in C++ with

**Exposition 2.12: Degradation in efficiency and effectiveness related to increase in $k$**

Empirical evidence suggests (see, e.g., Figure 3.3 in § 3.3) that recall degrades as $k$ increases. In our opinion, this degradation can be explained as follows: All approximate and exact methods for $k$-NN search that we know use heuristics to distinguish between potential nearest neighbors and points that are farther away from the query. In a nutshell all such heuristics (that we are aware of) crucially rely on the assumption that the distance between the query and a nearest neighbor is much smaller than the distance between the query and non-neighbor points. In that, as $k$ increases, the distance from the query to its farthest $k$-nearest neighbors approaches the average distance from the query to a randomly selected non-neighbor data point. As a result, it becomes hard to reliably distinguish between true (but relatively far) nearest neighbors and non-neighbor points.

In particular, all exact metric-space search methods that we know (see § 2.2.2.1) rely on index-time space partitioning with search-time pruning based on the triangle inequality. As explained on p. 32, a respective $k$-NN search is simulated via a range search with the shrinking radius, which is equal to the distance from the query to the $k$-th data point encountered by the search procedure. A partition can be safely discarded if and only if the minimum distance from the query to this partition is larger than the radius of the query (i.e., there is a guarantee—based on the triangle inequality—that the query ball does not intersect the partition). It can be seen that the higher is $k$, the larger are these query radii. In turn, the larger are the query radii, the more frequently the query ball intersects with a partition and, consequently, the less effective is partition pruning.

bindings for Python [23].

We use 200K element subsets of the first six data sets listed in Table 2.6. The results for low-dimensional data sets are presented in Figure 2.12. The results for high-dimensional data sets are presented in Figure 2.13. Looking at Figure 2.12, we can make several observations:

- No method for Mahalanobis *metric* distance learning is better than a fake learning method $L_2$, which simply computes the Euclidean distance between vectors.
- In all but two cases, the non-metric learning method RFD is roughly at par with $L_2$: in some cases $L_2$ is slightly better and in other cases RFD is slightly better.
- Achieving a 90% level of accuracy requires a large number of candidates ($10^2$–$10^3$). In that, it is especially hard to get accurate results for the Itakura-Saito distance.

We make similar observations (see Figure 2.13) from high-dimensional data, which is much more challenging:

- No metric learning method outperforms $L_2$, whereas $L_2$ is markedly better than learning approaches in two cases (Panels 2.13g, 2.13h).
- RFD is better than $L_2$ only in two cases (Panels 2.13a and 2.13c).
- Achieving a 90% level of accurace requires a large number of candidates ($10^3$–$10^4$). The number of required candidates is about an order of magnitude higher compared to low-

[23] `http://www.stat.ucla.edu/~caiming/codes/RFD_Package.zip`

dimensional data.

- Furthermore, for four test cases (see Panels 2.13a, 2.13b,2.13j, 2.13k), we fail to reach the 90% recall despite retrieving as many as 20480 candidate points, i.e., after retrieving about 10% of a data set.

These observations support the point of view that implementing a high-accuracy non-metric search by using a proxy candidate-generation *metric* distance is hardly a promissing line of research. The problem is that our proxy metric distances are not accurate. Therefore, to obtain a high accuracy, we need to generate a large number of candidates: This number can be orders of magnitude larger than $k$.

Candidates can be generated using either an exact brute-force search or an approximate $k$-NN search. The brute-force search introduces a substantial overhead. It also does not make sense for some statistical distances, which can be computed nearly as efficiently as the Euclidean distance. For example, NMSLIB bridges the gap between the speed of computation of the Euclidean distance and KL-divergence by precomputing logarithms at index time [43, 219].

In turn, computation of the proxy distance can be expensive as well. We can see that in several cases using RFD can make a difference with respect to the accuracy of approximating the original distance. However, in our experiments, it takes 30-130 seconds to compute RFD for a small set of 200K documents, which amounts to computing 2-6K distances per second. Despite the testing script is written in Python, we believe that these numbers truly represent the efficiency of RFD (or lack thereof), because the underlying RFD implementation is written in C++ (and is invoked from a Python script). In contrast, on a similar hardware, we can compute about 30 million Euclidean distances between 128-dimensional vectors, which is a four-order magnitude difference in efficiency.

Approximate $k$-NN search with reasonably small $k$ can be substantially more efficient than the brute-force search. However, for the purpose of candidate generation, we need to use the number of candidates that is (sometimes much) larger than the original $k$. This leads to reduction in both efficiency and effectiveness (see Exposition 2.12).

All in all, we believe that results in Figures 2.12 and 2.13 support the argument of Jacobs et al. [146] that mapping non-metric data to a Euclidean space is a coercion that is often against the nature of a similarity measure. We argue that such a coercion does not make sense, when an efficient and accurate non-metric search is possible without carrying out such a transformation. That said, we see that on *our* data if good $k$-NN accuracy is achievable by considering a small number of candidate entries ranked using a proxy Mahalanobis *metric* distance, nearly equally good results can be achieved by using the Euclidean distance directly. A likely reason for this is that Mahalanobis distance learning does not truly learn a new distance from scratch. It rather learns how to scale (and sometimes mix) features so that all important features contribute equally to a classification decision in the case when the distance is used for $k$-NN -based classification. However, we experiment with statistical distances where all vector dimensions contribute independently and with equal weights. In fact, an arbitrary permutation of arguments does not change the value of such a distance. Thus, scaling and/or mixing of vector dimensions does not produce better results than $L_2$.

| Failed to reach 99% recall? | Data set name | Space name | Recall reached | $k_c$ (cand. $k$) | Best symmetr. method |
|---|---|---|---|---|---|
| | RCV-8 | Itakura-Saito | 99.1% | 20 | min |
| | RCV-8 | KL-div. | 99.8% | 20 | min |
| | RCV-8 | Rényi div. $\alpha = 0.25$ | 99.9% | 20 | avg |
| | RCV-8 | Rényi div. $\alpha = 0.75$ | 100.0% | 20 | min |
| | RCV-8 | Rényi div. $\alpha = 2$ | 99.2% | 20 | min |
| | Wiki-8 | Itakura-Saito | 99.3% | 20 | min |
| | Wiki-8 | KL-div. | 99.2% | 40 | min |
| | Wiki-8 | Rényi div. $\alpha = 0.25$ | 99.7% | 20 | avg |
| | Wiki-8 | Rényi div. $\alpha = 0.75$ | 99.8% | 20 | min |
| | Wiki-8 | Rényi div. $\alpha = 2$ | 99.2% | 160 | min |
| | RandHist-8 | Itakura-Saito | 99.6% | 40 | avg |
| | RandHist-8 | KL-div. | 99.4% | 20 | min |
| | RandHist-8 | Rényi div. $\alpha = 0.25$ | 100.0% | 20 | min |
| | RandHist-8 | Rényi div. $\alpha = 0.75$ | 100.0% | 20 | min |
| | RandHist-8 | Rényi div. $\alpha = 2$ | 99.7% | 160 | min |
| ✓ | RandHist-32 | Itakura-Saito | 95.6% | 5120 | min |
| | RandHist-32 | KL-div. | 99.8% | 160 | min |
| | RandHist-32 | Rényi div. $\alpha = 0.25$ | 100.0% | 20 | min |
| | RandHist-32 | Rényi div. $\alpha = 0.75$ | 99.9% | 40 | min |
| ✓ | RandHist-32 | Rényi div. $\alpha = 2$ | 99.0% | 2560 | min |
| | RCV-128 | Itakura-Saito | 99.1% | 80 | min |
| | RCV-128 | KL-div. | 99.7% | 40 | min |
| | RCV-128 | Rényi div. $\alpha = 0.25$ | 99.6% | 80 | avg |
| | RCV-128 | Rényi div. $\alpha = 0.75$ | 99.4% | 20 | min |
| | RCV-128 | Rényi div. $\alpha = 2$ | 99.1% | 80 | min |
| | Wiki-128 | Itakura-Saito | 99.5% | 20 | min |
| | Wiki-128 | KL-div. | 99.3% | 40 | min |
| | Wiki-128 | Rényi div. $\alpha = 0.25$ | 99.2% | 160 | avg |
| | Wiki-128 | Rényi div. $\alpha = 0.75$ | 99.6% | 20 | min |
| | Wiki-128 | Rényi div. $\alpha = 2$ | 99.3% | 80 | min |
| | Manner | BM25 | 99.8% | 1280 | avg |

Table 2.10: Loss in effectiveness due to symmetrization for 10-NN search (using at most 500000 records from each collection).

| Failed to reach 99% recall? | Data set name | Space name | Recall reached | $k_c$ (cand. $k$) | Best symmetr. method |
|---|---|---|---|---|---|
| | RCV-8 | Itakura-Saito | 99.9% | 400 | min |
| | RCV-8 | KL-div. | 100.0% | 200 | min |
| | RCV-8 | Rényi div. $\alpha = 0.25$ | 99.9% | 200 | avg |
| | RCV-8 | Rényi div. $\alpha = 0.75$ | 99.9% | 200 | min |
| | RCV-8 | Rényi div. $\alpha = 2$ | 99.1% | 200 | min |
| | Wiki-8 | Itakura-Saito | 99.0% | 200 | min |
| | Wiki-8 | KL-div. | 99.4% | 400 | min |
| | Wiki-8 | Rényi div. $\alpha = 0.25$ | 99.2% | 200 | min |
| | Wiki-8 | Rényi div. $\alpha = 0.75$ | 99.6% | 200 | min |
| | Wiki-8 | Rényi div. $\alpha = 2$ | 99.0% | 12800 | min |
| | RandHist-8 | Itakura-Saito | 99.1% | 400 | avg |
| | RandHist-8 | KL-div. | 99.2% | 200 | min |
| | RandHist-8 | Rényi div. $\alpha = 0.25$ | 100.0% | 200 | min |
| | RandHist-8 | Rényi div. $\alpha = 0.75$ | 100.0% | 200 | avg |
| | RandHist-8 | Rényi div. $\alpha = 2$ | 99.2% | 1600 | min |
| ✓ | RandHist-32 | Itakura-Saito | 98.6% | 51200 | min |
| | RandHist-32 | KL-div. | 99.6% | 800 | min |
| | RandHist-32 | Rényi div. $\alpha = 0.25$ | 100.0% | 200 | min |
| | RandHist-32 | Rényi div. $\alpha = 0.75$ | 100.0% | 400 | min |
| | RandHist-32 | Rényi div. $\alpha = 2$ | 99.4% | 12800 | min |
| | RCV-128 | Itakura-Saito | 99.2% | 3200 | min |
| | RCV-128 | KL-div. | 99.8% | 400 | min |
| | RCV-128 | Rényi div. $\alpha = 0.25$ | 99.8% | 800 | avg |
| | RCV-128 | Rényi div. $\alpha = 0.75$ | 99.8% | 200 | min |
| | RCV-128 | Rényi div. $\alpha = 2$ | 99.2% | 1600 | min |
| | Wiki-128 | Itakura-Saito | 99.3% | 1600 | min |
| | Wiki-128 | KL-div. | 99.4% | 400 | min |
| | Wiki-128 | Rényi div. $\alpha = 0.25$ | 99.7% | 1600 | avg |
| | Wiki-128 | Rényi div. $\alpha = 0.75$ | 99.6% | 200 | min |
| | Wiki-128 | Rényi div. $\alpha = 2$ | 99.1% | 1600 | min |
| | Manner | BM25 | 99.8% | 3200 | avg |

Table 2.11: Loss in effectiveness due to symmetrization for 100-NN search (using at most 500000 records from each collection).

**Exposition 2.13: On similarity of query and document distributions**

In what follows we explain why it could be useful if distribution of queries is similar to distribution of documents. Note that this is not a formal proof. Imagine that $x$ is a nearest neighbor of the query $q$. When $x$ is inserted into a proximity graph, the insertion procedure traverses the existing graph and computes distances between $x$ and graph nodes. The graph traversal procedure uses the values of these distances to decide where to move next. Likewise, during retrieval we compute distances between $q$ and graph nodes. Again, the values of these distances guide the query-time traversal procedure. If the distance values computed between the query and the graph node during retrieval are similar to the distance values between the nearest neighbor $x$ and the graph node computed during insertion, then the query-time traversal procedure will generally follow the path of the index-time traversal procedure. Thus, $q$ will eventually reach $x$ or some other data points close to $x$.

#### 2.3.2.4 Experiments with Distance Symmetrization

In § 2.3.2.3, we provide evidence that learning a proxy metric distance to replace the original $k$-NN search can be quite hard. In that, a comparable or better accuracy can often be achieved by merely using $L_2$ as a proxy distance. In this subsection, we consider another, less drastic, way to simplify the search problem, namely, a distance symmetrization. We evaluate the effect of the distance symmetrization in three scenarios:

- The first scenario is similar to the test setup of § 2.3.2.3. The symmetrized distance is used to generate a list of $k_c$ candidates, whose completeness is subsequently evaluated. The candidate generation step employs an *exact* brute-force $k$-NN search with the symmetrized distance.
- In the second scenario, a symmetrized distance is used for *both* indexing and retrieval (i.e., we rely on full symmetrization). The search procedure is carried out by SW-graph, which generates a list of $k_c$ candidates. Then, candidates are compared exhaustively with the query. Unlike the first experimental scenario of this subsection, the list of $k_c$ candidates is obtained via an *approximate* $k$-NN search.
- The third scenario relies on a partial, i.e., index-time only, symmetrization. Specifically, the symmetrized distance is used only to construct a proximity graph (using the SW-graph indexing algorithm). Then, the search procedure uses the original, non-symmetrized distance to "guide" the search through the proximity graph. This approach generates a *final* list $k$ nearest neighbors rather than $k_c$ candidates, which do not need to be re-ranked.

Given a non-symmetric distance, there are two folklore approaches to make it symmetric, which use the value of the original distance $d(x, y)$ as well as the value of the distance function obtained by reversing arguments: $d_{\text{reverse}}(x, y) = d(y, x)$. Informally, we call the latter an *argument-reversed* distance. In the case of an average-based symmetrization, we compute the symmetrized distance as an average of the original and argument-reversed distances:

$$d_{\text{sym}} = \frac{d(x, y) + d_{\text{reverse}}(x, y)}{2} = \frac{d(x, y) + d(y, x)}{2} \tag{2.20}$$

In the case of a min-based symmetrization, we use their minimum:

$$d_{\text{sym}} = \min\left(d(x,y), d_{\text{reverse}}(x,y)\right) = \min\left(d(x,y), d(y,x)\right) \tag{2.21}$$

Symmetrization techniques given by Eq. (2.20) and Eq. (2.21) are suboptimal in the sense that a *single* computation of the symmetrized distance entails *two* computations of the original distance. We can do better, when a distance function permits a more *natural* symmetrization, in particular, in the case of TF×IDF, the naturally symmetrized distance is given by the formula:

$$d(x,y) = -\sum_{x_i=y_i} \text{TF}(x)_i \cdot \text{TF}(y)_i \cdot \text{IDF}(y)_i =$$

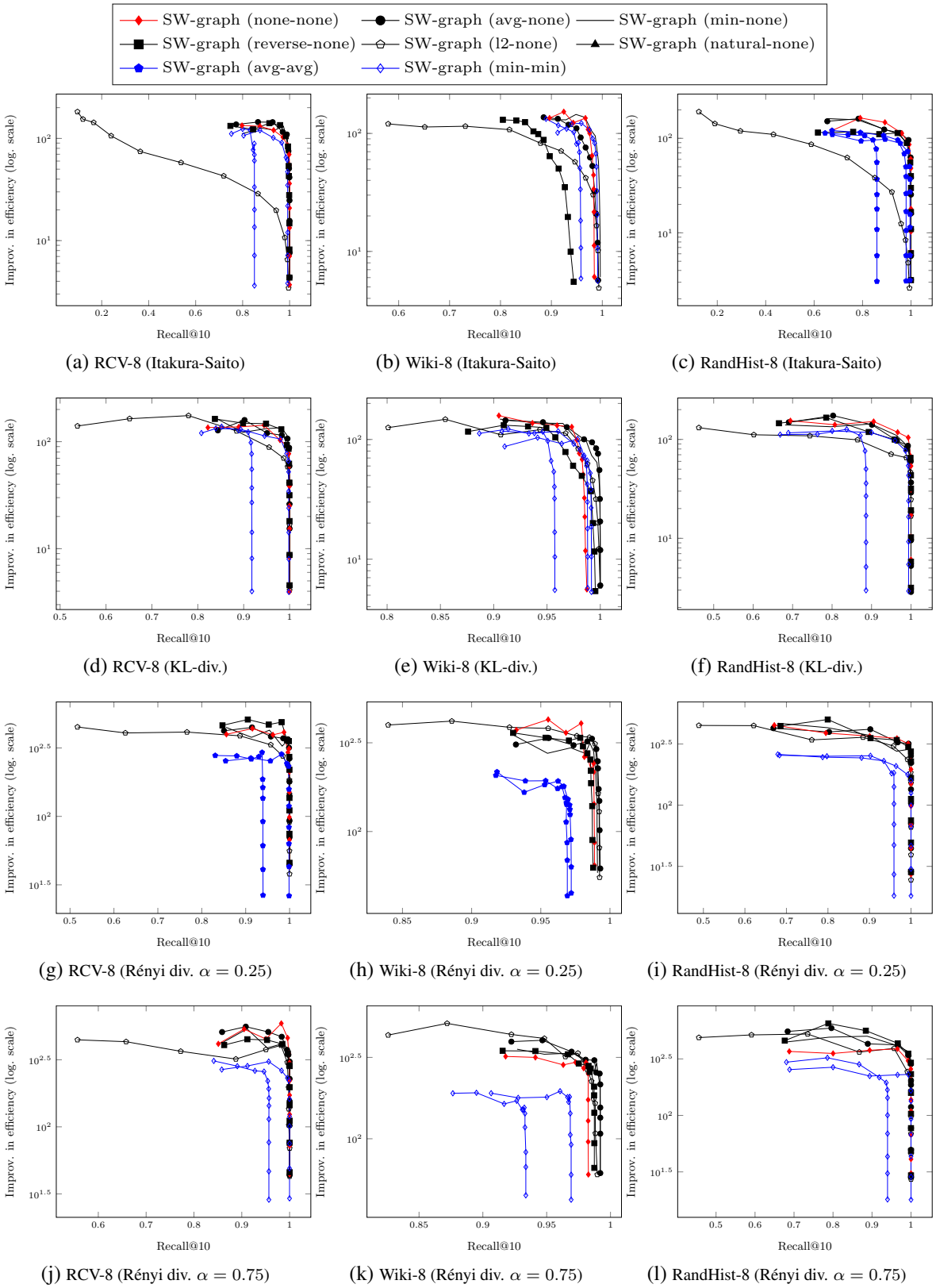
$$= -\sum_{x_i=y_i} \left(\text{TF}(x)_i \sqrt{\text{IDF}(x)_i}\right) \cdot \left(\text{TF}(y)_i \sqrt{\text{IDF}(y)_i}\right) \tag{2.22}$$

As noted on p. 19, although TF×IDF is not symmetric its value can be represented as the *symmetric* inner product between sparse query and document vectors. It may be even better to "share" a value of $\text{IDF}_i$ between the query and the document vectors by "assigning" each vector the value $\sqrt{\text{IDF}_i}$. More formally, in this "shared" setting a query vector is represented by the values $\sqrt{\text{IDF}_i}$, whereas a document vector is represented by the values $\sqrt{\text{IDF}_i} \cdot \text{TF}_i$. Sharing of `IDF` values makes queries more similar to documents (in $L_2$ and the cosine similarity), which may improve performance of a graph-based search algorithm (see Exposition 2.13). Although this symmetrization method is hardly novel, we have not seen it in the literature. Henceforth, for TF×IDF we use a natural symmetrization approach with sharing of `IDF` values. We compare this approach to the average- and min-symmetrization.

In the first series of experiments, we assess the accuracy of only generic symmetrization approaches in a re-ranking experiment. We carried out an experiment for $k = 10$ and $k = 100$ and measured effectiveness for various $k_c = k \cdot 2^i$, $i \leq 7$ (used $i \leq 7$ in several hard cases). The results for 31 combination of data sets and distance functions are reported in Tables 2.10 and 2.11, where we report the minimum $k_c$ for which we reach a 99% recall. In three cases, this is not possible. We highlight these cases by check marks in the first table column. In each test case, we randomly split data three times and average results over three splits. For all distances except Rényi divergence we use 1K queries for each split, i.e., the total number of queries is 3K. Because Rényi divergence is slow to compute we use only 200 queries per split (i.e., the overall number of queries is 600).

From Table 2.10 and 2.11 it can be immediately seen that in most cases symmetrization has only a small impact on accuracy. Specifically, in most cases getting a nearly perfect recall requires a relatively small number of candidates $k_c \leq 4 \cdot k$. However, there are three truly difficult data sets where symmetrization is not accurate: Itakura-Saito distance with RandHist-32, Rényi divergence with RandHist-32, and BM25 with Manner. In particular, in the case of the Itakura-Saito distance and RandHist-32 we fail to get a 99% recall despite using $k_c = 5120$ for $k = 10$ and $k_c = 51200$ for $k = 100$.

In the second and the third scenario, we use the same 31 combination of data sets and distance functions. However, we now experiment with index- and query-time symmetrization in an actual indexing algorithm SW-graph rather than relying on the brute-force search. For the second



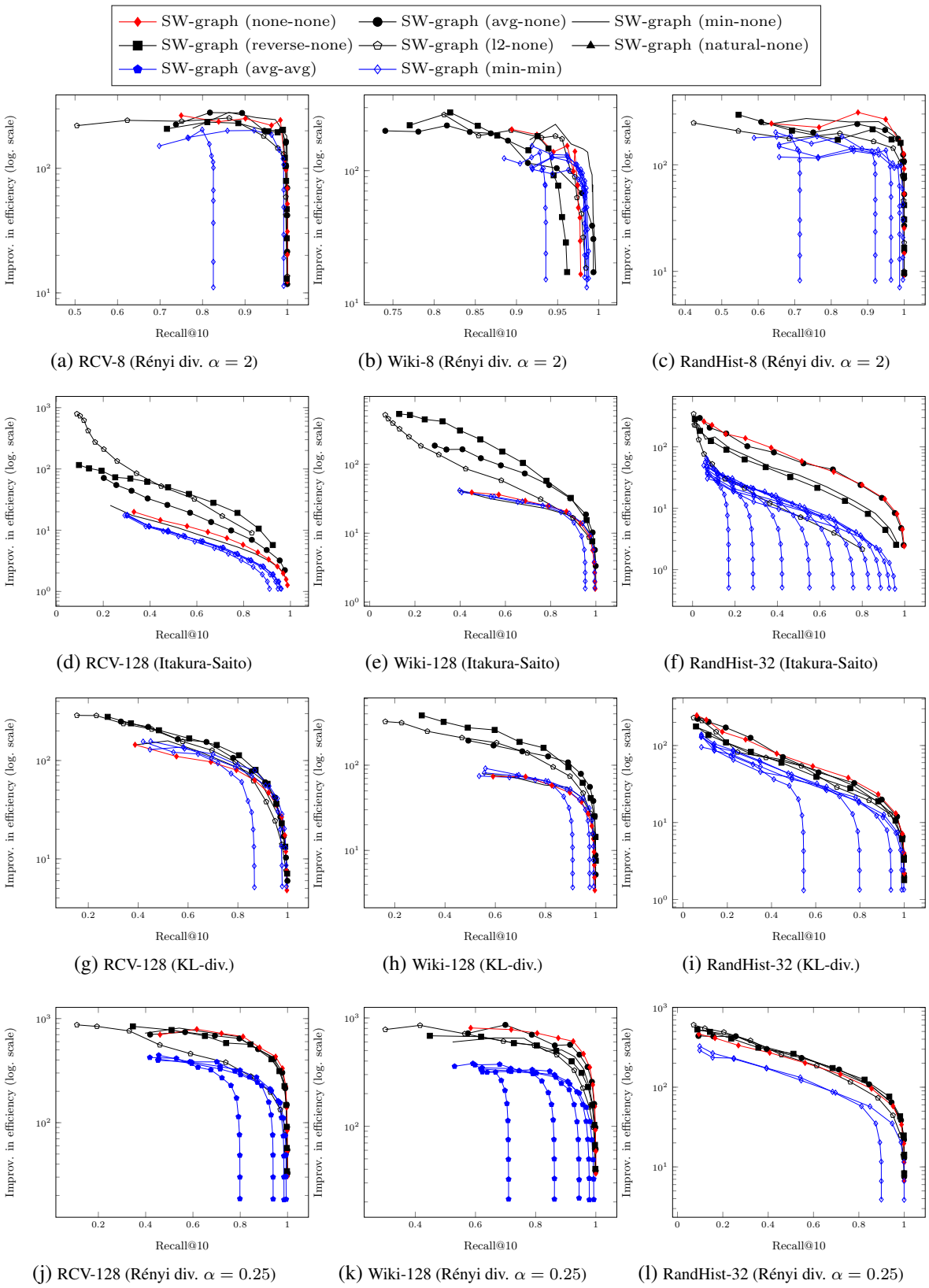

(a) RCV-8 (Itakura-Saito) (b) Wiki-8 (Itakura-Saito) (c) RandHist-8 (Itakura-Saito)

(d) RCV-8 (KL-div.) (e) Wiki-8 (KL-div.) (f) RandHist-8 (KL-div.)

(g) RCV-8 (Rényi div. $\alpha = 0.25$) (h) Wiki-8 (Rényi div. $\alpha = 0.25$) (i) RandHist-8 (Rényi div. $\alpha = 0.25$)

(j) RCV-8 (Rényi div. $\alpha = 0.75$) (k) Wiki-8 (Rényi div. $\alpha = 0.75$) (l) RandHist-8 (Rényi div. $\alpha = 0.75$)

Figure 2.14: Efficiency/effectiveness trade-offs of symmetrization in 10-NN search (part I). The number of data points is at most 500K. Best viewed in color.



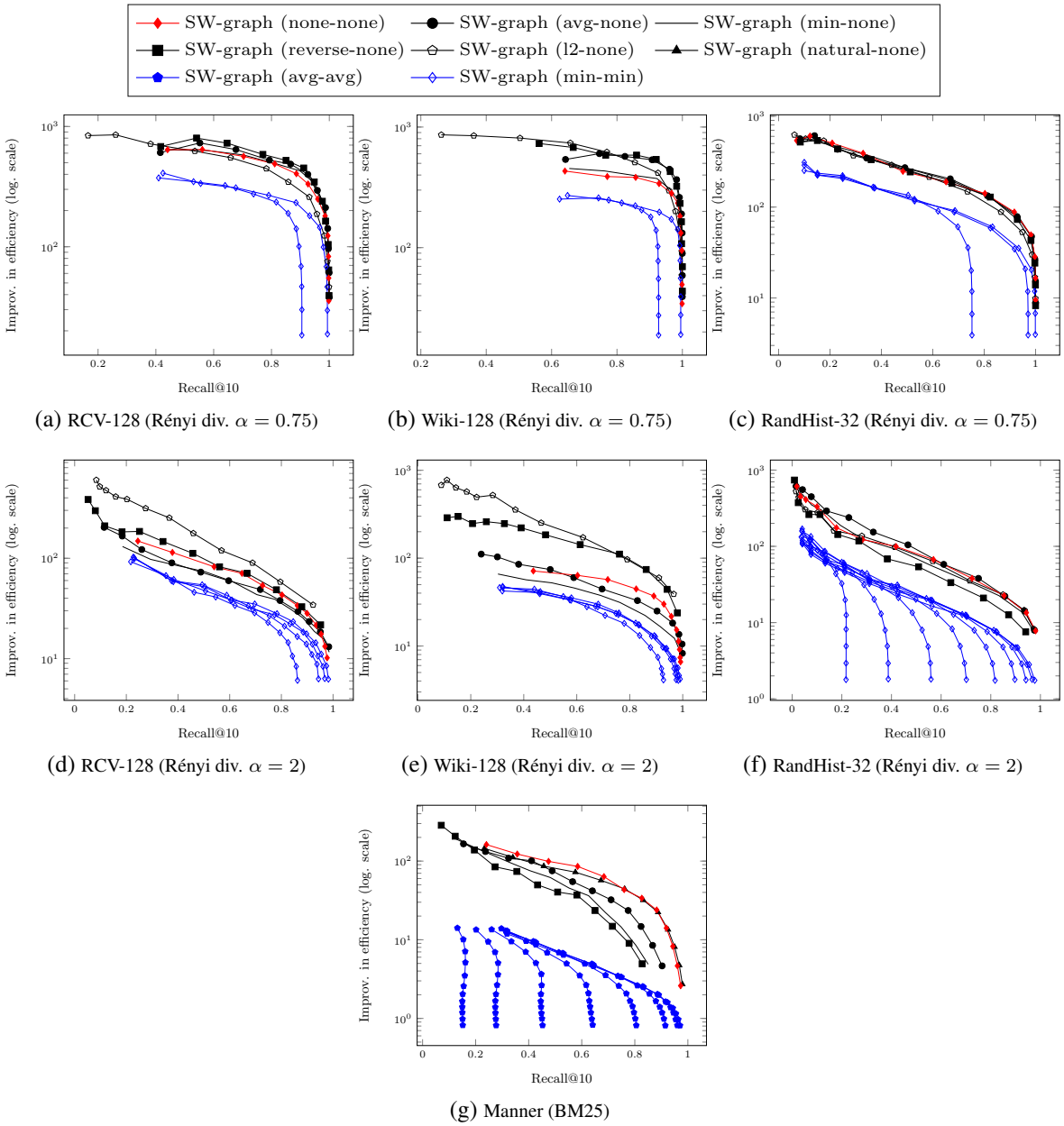

(a) RCV-8 (Rényi div. $\alpha = 2$) (b) Wiki-8 (Rényi div. $\alpha = 2$) (c) RandHist-8 (Rényi div. $\alpha = 2$)

(d) RCV-128 (Itakura-Saito) (e) Wiki-128 (Itakura-Saito) (f) RandHist-32 (Itakura-Saito)

(g) RCV-128 (KL-div.) (h) Wiki-128 (KL-div.) (i) RandHist-32 (KL-div.)

(j) RCV-128 (Rényi div. $\alpha = 0.25$) (k) Wiki-128 (Rényi div. $\alpha = 0.25$) (l) RandHist-32 (Rényi div. $\alpha = 0.25$)

Figure 2.15: Efficiency/effectiveness trade-offs of symmetrization in 10-NN search (part II). The number of data points is at most 500K. Best viewed in color.




(a) RCV-128 (Rényi div. $\alpha = 0.75$) (b) Wiki-128 (Rényi div. $\alpha = 0.75$) (c) RandHist-32 (Rényi div. $\alpha = 0.75$)

(d) RCV-128 (Rényi div. $\alpha = 2$) (e) Wiki-128 (Rényi div. $\alpha = 2$) (f) RandHist-32 (Rényi div. $\alpha = 2$)

(g) Manner (BM25)

Figure 2.16: Efficiency/effectiveness trade-offs of symmetrization in 10-NN search (part III). The number of data points is at most 500K. Best viewed in color.

scenario, we use two actual symmetrization strategies (the minimum- and the average-based symmetrization) as well as two methods of quasi-symmetrization. In the first method, we build the proximity graph using the Euclidean distance between vectors. In the second method, we build the proximity graph using the argument-reversed distance.

We verified that none of these quasi-symmetrization approaches would produce a better list of candidates in the first scenario, where the brute-force search is used to produce a candidate list. In fact, in many cases a quasi-symmetrization method needs to produce a much longer candidate list. For example, for Wiki-128 and KL-divergence, it takes $k_c = 40$ candidates to exceed a 99% recall in a 10-NN search for the minimum-based symmetrization. For the $L_2$-based symmetrization, it takes as many as $k_c = 320$ candidates. The results are even worse for the filtering based on the

argument-reversed distance: By using as many as $k_c = 1280$ candidates we obtain a recall of only $95.6\%$. We, nevertheless, want to assess if the graph-construction algorithm is sensitive to the quality of the distance approximation/symmetrization.

The results for scenarios two and three are presented in Figures 2.14-2.16 where plots are sorted roughly in the order of increasing dimensionality. We test several modifications of SW-graph each of which has an additional marker in the form: `a-b`, where `a` denotes a type of index-time symmetrization and `b` denotes a type of query-time symmetrization. Red plots represent the unmodified SW-graph algorithm, which is labeled as SW-graph (none-none).

Black plots represent modifications, where symmetrization is used only during indexing: SW-graph (avg-none), SW-graph (min-none), SW-graph (l2-none), SW-graph (reverse-none), and SW-graph (natural-none). The first two symmetrizations are average- and minimum-based. SW-graph (l2-none) is a quasi-symmetrization relying on computation of $L_2$. SW-graph (reverse) builds the graph using the reversed-argument distance. SW-graph (natural-none) is a natural symmetrization of BM25 described by Eq. (2.22), which is used only for one data set.

Blue plots represent the scenario where SW-graph is built using a fully symmetrized distance. The index is used to carry out a $k_c$-NN search, which produces a list of candidates for further verification. Depending on which symmetrization was more effective in the first series of experiments (see results in Tables 2.10-2.11), we use either SW-graph (min-min) or SW-graph (avg-avg), which stand for full minimum- or average- based symmetrization. Because we do not know what is an optimum number of candidate records, we experiment with $k_c = k \cdot 2^i$ for successive integer values $i$. The larger is $i$, the more accurate is the filtering step. However, it does not make sense to increase $i$ beyond the point where the filtering accuracy reaches 99%. For this reason, the minimum value of $i$ is zero and the largest value of $i$ is taken from the last column of Table 2.10. For the remaining parameters of SW-graph we choose values that are known to perform well in other experiments. In particular, we choose `NN`=15, `efConstruction`=100, and `efSearch` = $2^j$ for $0 \leq j \leq 12$. Analogous to the first scenario, we use 31 combination of data sets and distances. In each test, we randomly split data (into queries and indexable data) three times and average results over three splits.

The results are presented in Figures 2.14-2.16. Note that results for low-dimensional data sets are shown only in Figure 2.14 and in the first row of Figure 2.15. We can see that in many cases there is little difference among best runs with the fully symmetrized distance (a method SW-graph (min-min) or SW-graph (avg-avg)) the runs produced by methods with true index-time symmetrization (SW-graph (min-none), SW-graph (avg-none)), and the original unmodified search algorithm (SW-graph (none-none)). Furthermore, we can see that there is virtually no difference between SW-graph (min-none), SW-graph (avg-none), and SW-graph (none-none). However, sometimes all fully-symmetrized runs (for all values of $k_c$) are noticeably less efficient (see, e.g., Panels 2.14h and 2.14k). This difference is more pronounced in the case of high-dimensional data. Here, full symmetrization leads to a substantial (up to an order of magnitude) loss in performance in most cases.

Effectiveness of index-time symmetrization varies from case to case and there is no definitive winner. First, we note that in four cases index-time symmetrization is beneficial (Panels 2.15d, 2.15e, 2.16d, 2.16e). In particular, in Panels 2.15d, 2.15e, 2.16e, there is an up to $10\times$ speedup. However, it can be achieved by using an argumented-reversed distance (Panels 2.15d, 2.15e) or $L_2$ (2.16e).

This finding is quite surprising given that these quasi-symmetrization approaches do not perform well in the first series of re-ranking experiments. In particular, for $L_2$ and Wiki-128 reaching a 99% recall requires $k_c = 640$ compared to $k_c = 80$ for min-based symmetrization. For the Itakura-Saito and data sets RCV-128 and Wiki-128, it takes $k_c \leq 80$ to get a 99% recall. However, using the argument-reversed distance, we do not even reach the recall of 60% despite using a large $k_c = 1280$. It is worth noting, however, that in several cases using argument-reversed distance at index time leads to substantial degradation in performance (see, e.g., Panels 2.14b and 2.16g).

In conclusion of this section, we want to emphasize that in all test cases the best performance is achieved using either the unmodified SW-graph or the SW-graph with a proxy distance function to build the proximity graph. However, there is not a single case where performance is improved by using the fully symmetrized distance at both indexing and querying steps.

Furthermore, in all three especially challengine cases: Itakura-Saito distance with RandHist-32, Rényi divergence with RandHist-32, and BM25 with Manner, SW-graph has excellent performance. In all three cases (see Panels 2.15f, 2.16f,2.16g), there is more than a 10× speed up at 90% recall. Note that in these three cases data is substantially non-symmetric: Depending on the case, to accurately retrieve 10 nearest neighbors with respect to the original metric, it requires to obtain 1-5K nearest neighbors using its symmetrized variant (see Table 2.10).

### 2.3.3 Comparison to TriGen

In this section, we compare two approaches of adapting a metric-space search method to non-metric data. The first approach is a novel method to learn a pruning rule, which we discuss in § 2.2.3. The second approach called TriGen consists in "stretching" the distance function using a monotonic concave transformation [271] that reduces non-metricity of the distance. TriGen is designed only for *bounded*, *semimetric* distances (see Table 2.2), which are symmetric, non-negative, and become zero only for identical data points.

Let $x$, $y$, $z$ be an arbitrary ordered triple of points such that $d(x, y)$ is the largest among three pairwise distances, i.e., $d(x, y) \geq \max(d(x, z), d(z, y))$. If $d(x, y)$ is a metric distance, the following conditions should all be true:

$$\begin{aligned} d(x, y) &\leq d(x, z) + d(z, y) \\ d(y, z) &\leq d(y, x) + d(x, z) \\ d(x, z) &\leq d(x, y) + d(y, z) \end{aligned} \tag{2.23}$$

Because $d(x, y) \geq \max(d(x, z), d(z, y))$, the second and the third inequalities in (2.23) are trivially satisfied for (not necessarily metric) *symmetric* and *non-negative* distances. However, the first condition can be violated, if the distance is non-metric. The closer is the distance is to the metric distance, the less frequently we encounter such violations. Thus, it is quite reasonable to assess the degree of deviation from metricity by estimating a probability that the triangle inequality is violated (for a randomly selected triple), which is exactly what is suggested by Skopal [271].

Skopal proposes a clever way to decrease non-metricity by constructing a new distance $f(d(x, y))$, where $f()$ is a monotonically increasing concave function. The concave function

"stretches" the distance and makes it more similar to a true metric compared to the original distance $d(x, y)$. As we show in § 2.3.2.2 a straightforward application of the metric space search method VP-tree to a challenging non-metric distance results in low retrieval accuracy (see results Table 2.8). However, by stretching the distance using the TriGen transformation, it can be possible to improve accuracy (at the expense of reduced efficiency).

Specifically, Skopal proves that a monotonic concave transformation can reduce the number of cases when the triangle inequality is violated. At the same time, due to the monotonicity of such a transformation, the $k$-NN search using the modified distance produces the same result as the $k$-NN search using the original distance. Thus, the TriGen strategy to dealing with non-metric data consists in (1) employing a monotonic transformation that makes a distance approximately metric while preserving the original set of nearest neighbors, and (2) indexing data using an exact metric-space access method.

A TriGen mapping $f(x)$ (defined for $0 \leq x \leq 1$) is selected from the union of two parametric families of concave functions, which are termed as bases:

- A fractional power base $FP(x, w) = x^{\frac{1}{1+w}}$;
- A Rational Bézier Quadratic (RBQ) base $RBQ_{(a,b)}(x, w)$, $0 \leq a < b \leq 1$. The exact functional form of RBQ is not relevant to this discussion (see [271] for details).

Note that parameters $w$, $a$, and $b$ are treated as constants, which define a specific functional form. By varying these parameters we can design a necessary stretching function. The larger is the value of $w$ the more concave is the transformation and the more "metric" is the transformed distance. In particular, as $w \to \infty$, both RBQ and FP converge to one minus the Dirac delta function: The limit function of all bases is equal to zero for $x = 0$ and to one for $0 < x \leq 1$. As noted by Skopal, applying such a degenerate transformation produces a *trivial* metric space where $d(x, x) = 0$ and is a constant otherwise.

A learning objective of TriGen, however, is to select a *single* concave function that satisfies the accuracy requirements while allowing for efficient retrieval. The fraction of violations is computed for a set of `trigenSampleTripletQty` ordered data point triples sampled from a set of `trigenSampleQty` data points, which are, in turn, are selected randomly from the data set (uniformly and without replacement). The fraction of violations is required to be above the threshold `trigenAcc`. Values `trigenSampleTripletQty`, `trigenSampleQty`, and `trigenAcc` are all parameters in our implementation of TriGen. To assess efficiency Skopal [271] uses the value of an intrinsic dimensionality as a proxy metric (see Exposition 2.2). The idea is that the modification of the distance with the lowest intrinsic dimensionality should result in the fast retrieval method.

Because it is not feasible to optimize over the infinite set of transformation functions, TriGen employs a finite pool of bases, which includes FB and multiple RBQ bases for alls possible combinations of parameters $a$ and $b$ such that $0 \leq a < b \leq 1$. For each base, TriGen uses a binary search to find the minimum parameter $w$ such that the transformed distance deviates from a metric distance within specified limits. Then the base with minimum intrinsic dimensionality is selected.

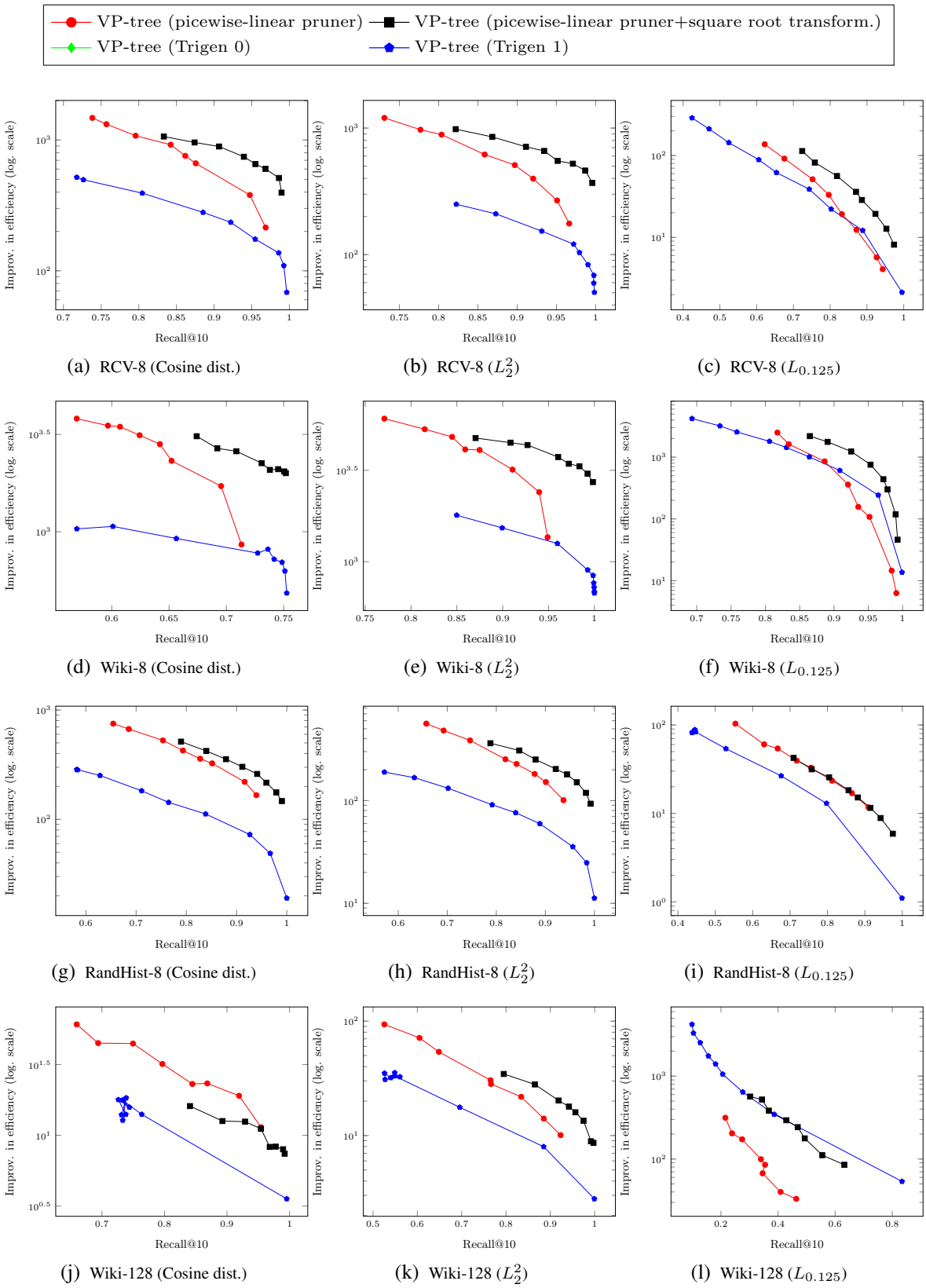
TriGen has two major limitations: (in addition to be non-negative) the distance should be symmetric and bounded. Bounding can be easily provided by using $\min(d(x, y)/d_{\text{max}}, 1)$ instead of the original distance.[24] Note that $D_{\text{max}}$ is an empirically estimated maximum distance (by

[24]For efficiency reasons this is simulated via multiplication by inverse maximum distance.

computing $d(x, y)$ for a sample of data set point pairs).

As noted by Skopal, searching with a non-symmetric distance can be partially provided by a filter-and-refine approach, where a fully symmetrized distance (e.g., a minimum of the original and argument-reversed distance) is used during the filtering step. However, as we learn in § 2.3.2.4, the filtering step carries out a $k_c$-NN search with $k_c$ (sometimes substantially) larger than $k$. This is required to compensate for the lack of accuracy by replacing the original distance with the symmetrized one.

Using $k_c > k$ leads to reduced efficiency (see Exposition 2.12). Thus, instead of the complete symmetrization, we do retrieval with $k_c = k$. In addition, we explore two variants of TriGen, where one variants does not compute the symmetrized distance for data points in the buckets.



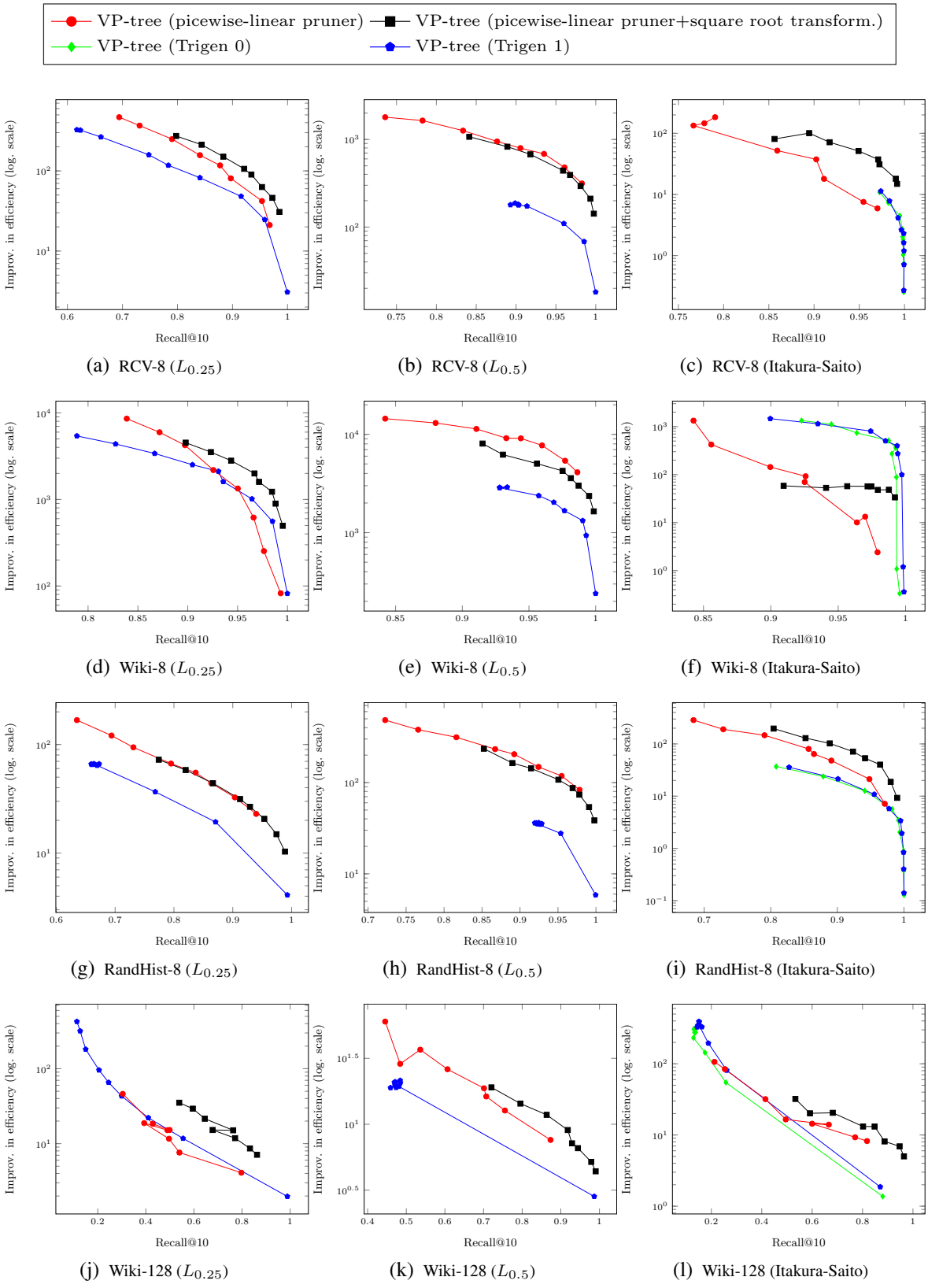

(a) RCV-8 (Cosine dist.) (b) RCV-8 ($L_2^2$) (c) RCV-8 ($L_{0.125}$)

(d) Wiki-8 (Cosine dist.) (e) Wiki-8 ($L_2^2$) (f) Wiki-8 ($L_{0.125}$)

(g) RandHist-8 (Cosine dist.) (h) RandHist-8 ($L_2^2$) (i) RandHist-8 ($L_{0.125}$)

(j) Wiki-128 (Cosine dist.) (k) Wiki-128 ($L_2^2$) (l) Wiki-128 ($L_{0.125}$)

Figure 2.17: Improvement in efficiency vs recall for VP-tree based methods in 10-NN search (part I). Best viewed in color.



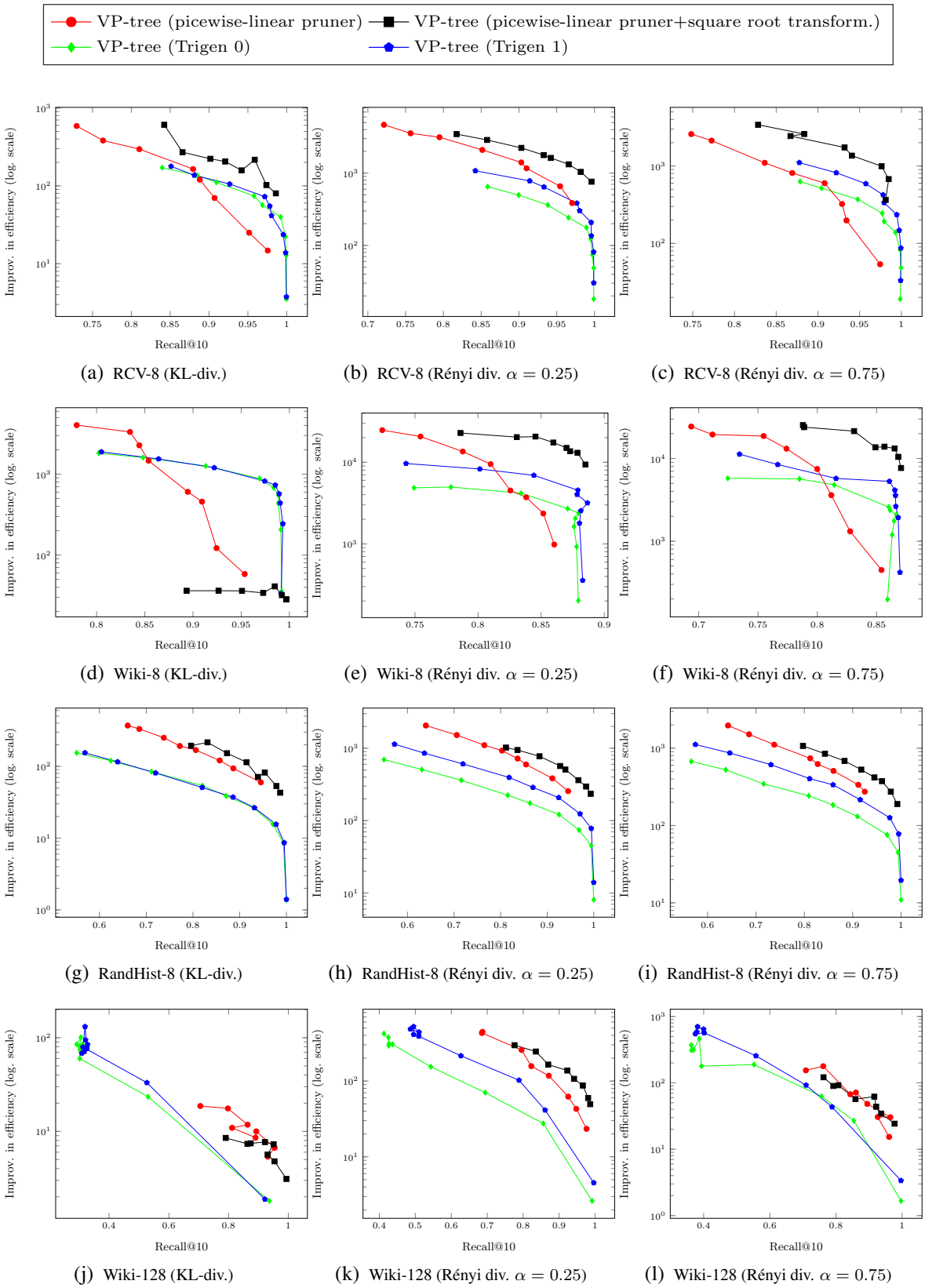

(a) RCV-8 ($L_{0.25}$) (b) RCV-8 ($L_{0.5}$) (c) RCV-8 (Itakura-Saito)

(d) Wiki-8 ($L_{0.25}$) (e) Wiki-8 ($L_{0.5}$) (f) Wiki-8 (Itakura-Saito)

(g) RandHist-8 ($L_{0.25}$) (h) RandHist-8 ($L_{0.5}$) (i) RandHist-8 (Itakura-Saito)

(j) Wiki-128 ($L_{0.25}$) (k) Wiki-128 ($L_{0.5}$) (l) Wiki-128 (Itakura-Saito)

Figure 2.18: Improvement in efficiency vs recall for VP-tree based methods in 10-NN search (part II). Best viewed in color.



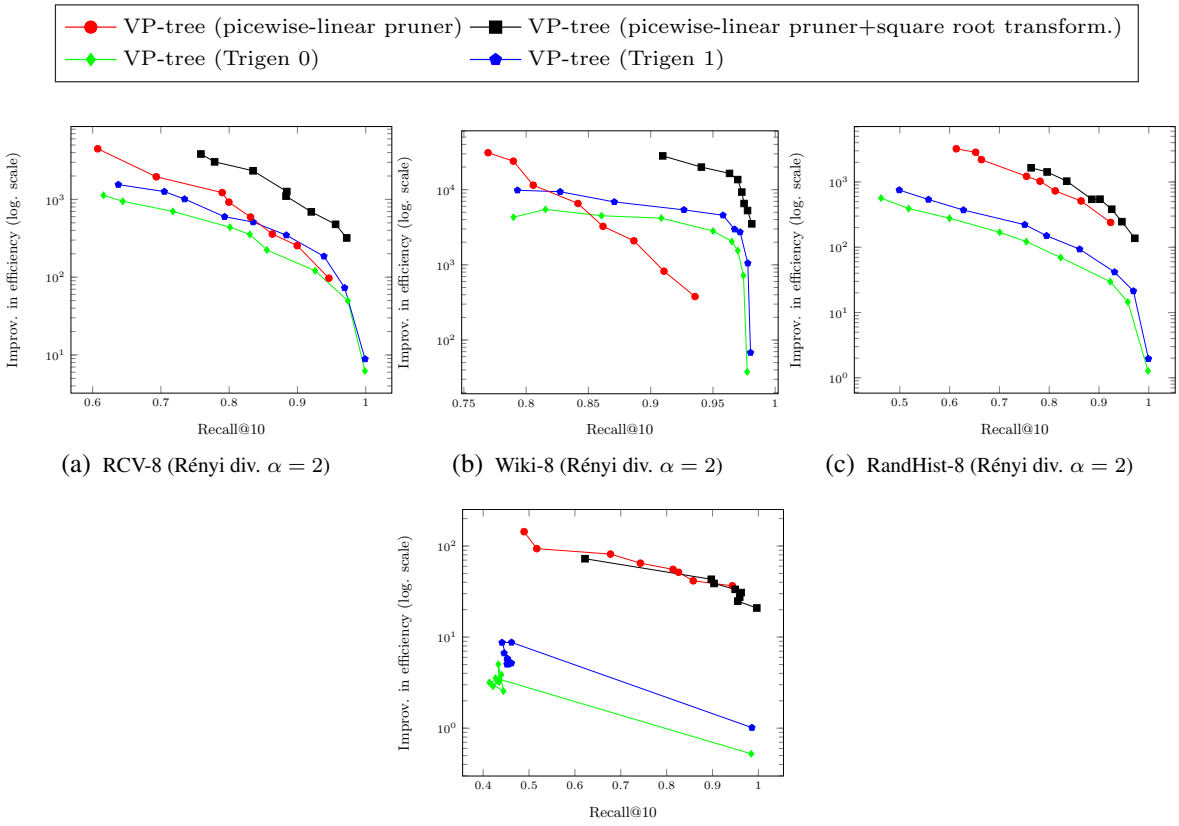

(a) RCV-8 (KL-div.) (b) RCV-8 (Rényi div. $\alpha = 0.25$) (c) RCV-8 (Rényi div. $\alpha = 0.75$)

(d) Wiki-8 (KL-div.) (e) Wiki-8 (Rényi div. $\alpha = 0.25$) (f) Wiki-8 (Rényi div. $\alpha = 0.75$)

(g) RandHist-8 (KL-div.) (h) RandHist-8 (Rényi div. $\alpha = 0.25$) (i) RandHist-8 (Rényi div. $\alpha = 0.75$)

(j) Wiki-128 (KL-div.) (k) Wiki-128 (Rényi div. $\alpha = 0.25$) (l) Wiki-128 (Rényi div. $\alpha = 0.75$)

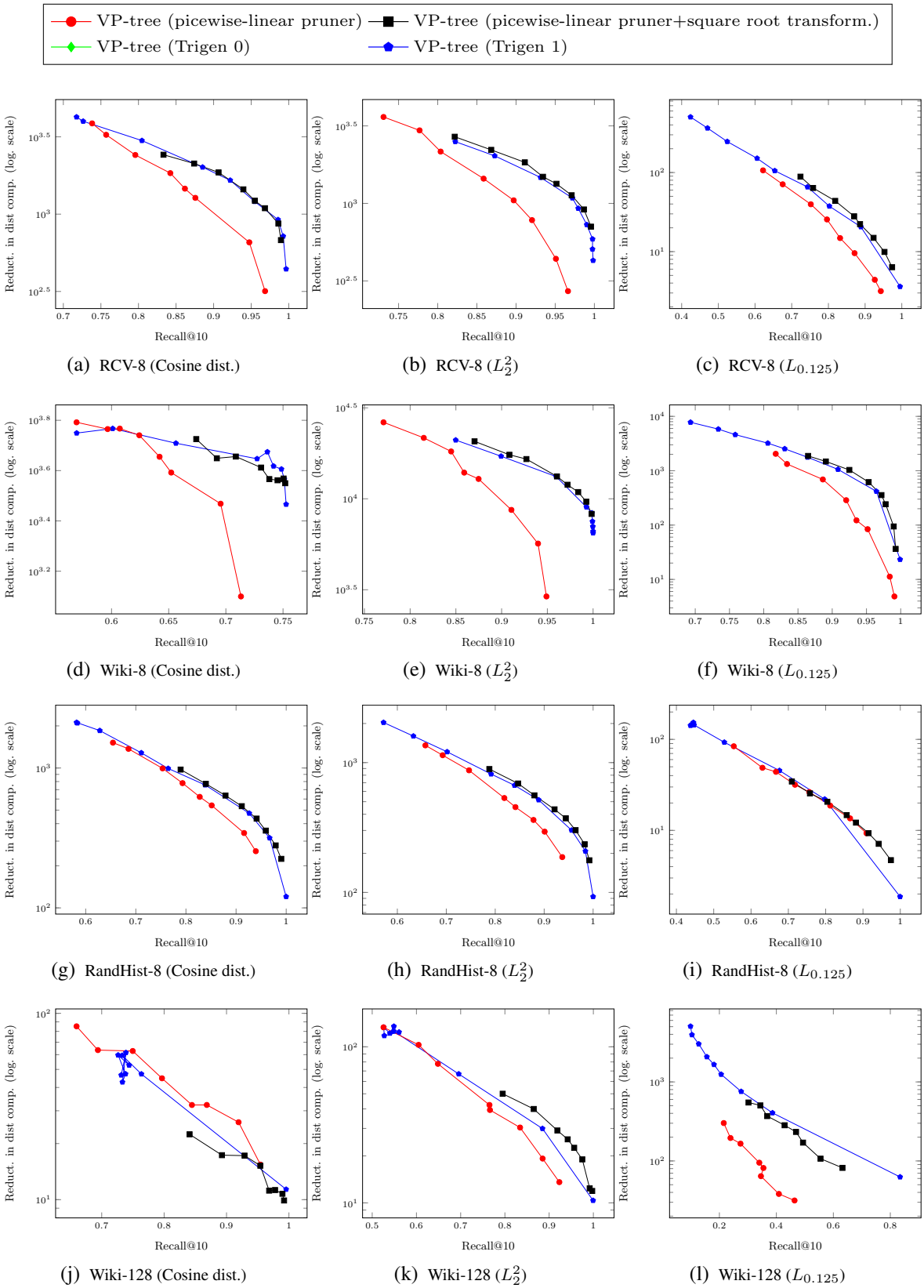
Figure 2.19: Improvement in efficiency vs recall for VP-tree based methods in 10-NN search (part III). Best viewed in color.




(a) RCV-8 (Rényi div. $\alpha = 2$) (b) Wiki-8 (Rényi div. $\alpha = 2$) (c) RandHist-8 (Rényi div. $\alpha = 2$)

(d) Wiki-128 (Rényi div. $\alpha = 2$)

Figure 2.20: Improvement in efficiency vs recall for VP-tree based methods in 10-NN search (part IV). Best viewed in color.



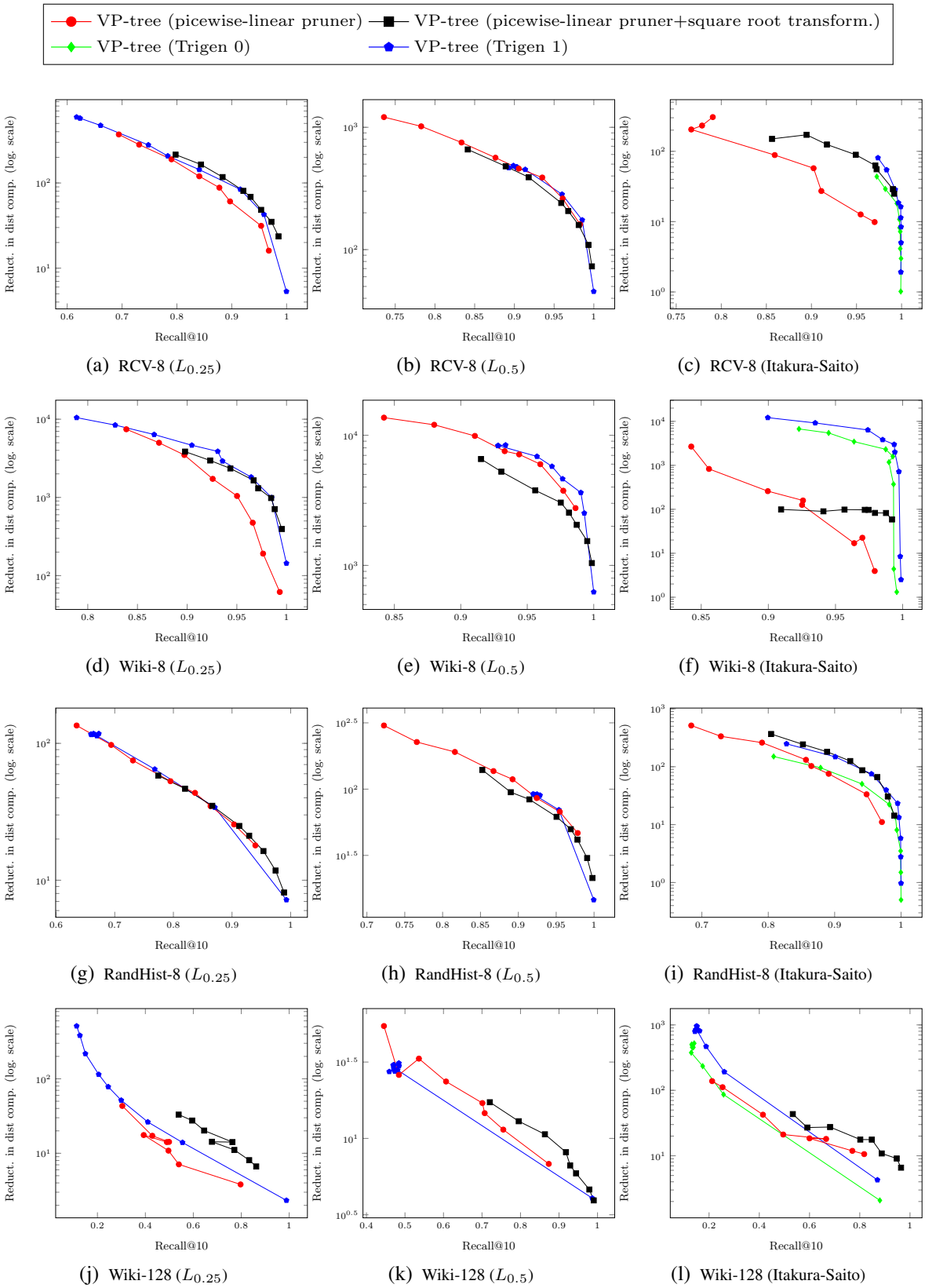

(a) RCV-8 (Cosine dist.) (b) RCV-8 ($L_2^2$) (c) RCV-8 ($L_{0.125}$)

(d) Wiki-8 (Cosine dist.) (e) Wiki-8 ($L_2^2$) (f) Wiki-8 ($L_{0.125}$)

(g) RandHist-8 (Cosine dist.) (h) RandHist-8 ($L_2^2$) (i) RandHist-8 ($L_{0.125}$)

(j) Wiki-128 (Cosine dist.) (k) Wiki-128 ($L_2^2$) (l) Wiki-128 ($L_{0.125}$)

Figure 2.21: Reduction in the number of distance computations vs recall for VP-tree based methods in 10-NN search (part I). Best viewed in color.



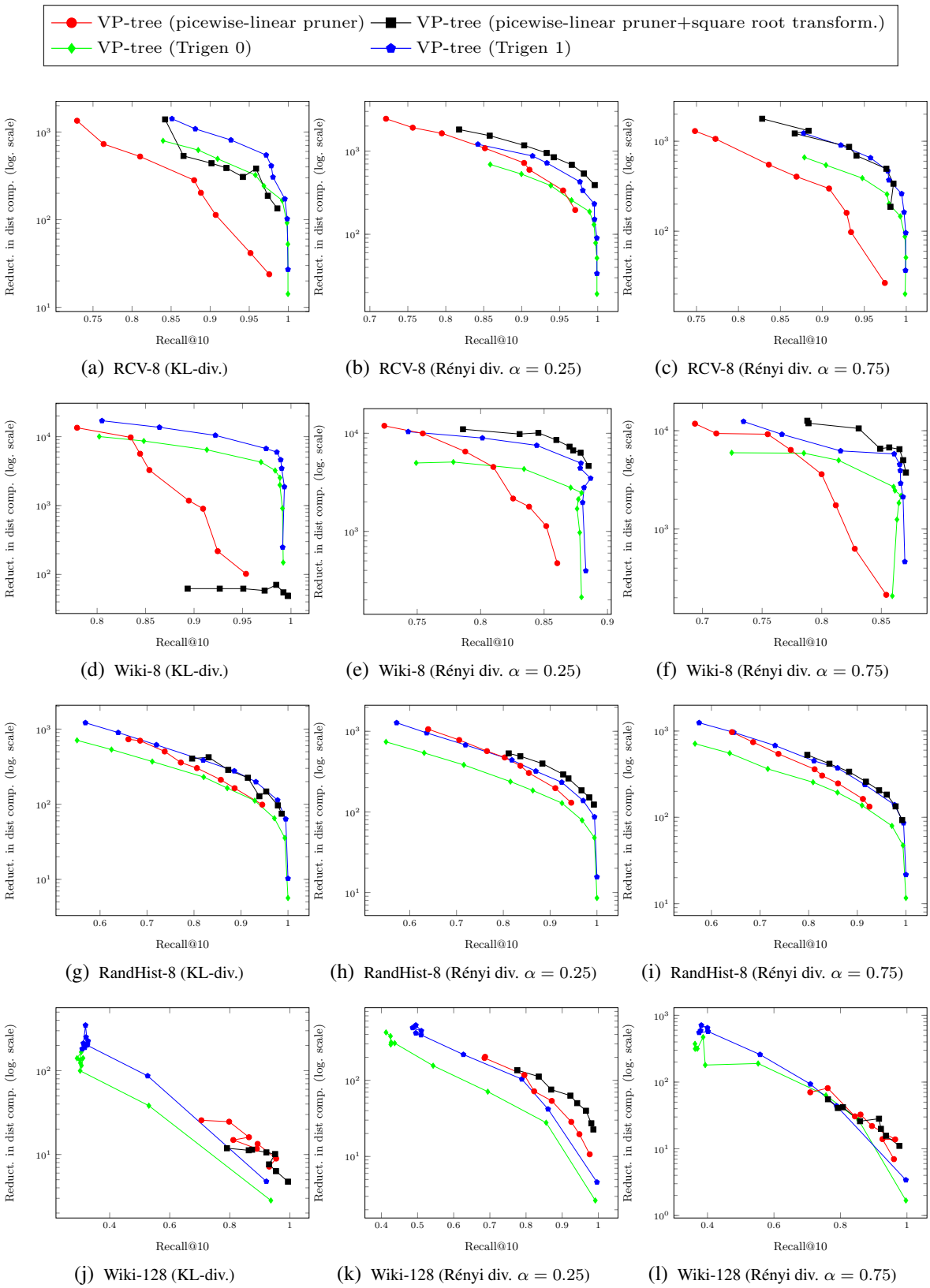

(a) RCV-8 ($L_{0.25}$) (b) RCV-8 ($L_{0.5}$) (c) RCV-8 (Itakura-Saito)

(d) Wiki-8 ($L_{0.25}$) (e) Wiki-8 ($L_{0.5}$) (f) Wiki-8 (Itakura-Saito)

(g) RandHist-8 ($L_{0.25}$) (h) RandHist-8 ($L_{0.5}$) (i) RandHist-8 (Itakura-Saito)

(j) Wiki-128 ($L_{0.25}$) (k) Wiki-128 ($L_{0.5}$) (l) Wiki-128 (Itakura-Saito)

Figure 2.22: Reduction in the number of distance computations vs recall for VP-tree based methods in 10-NN search (part II). Best viewed in color.



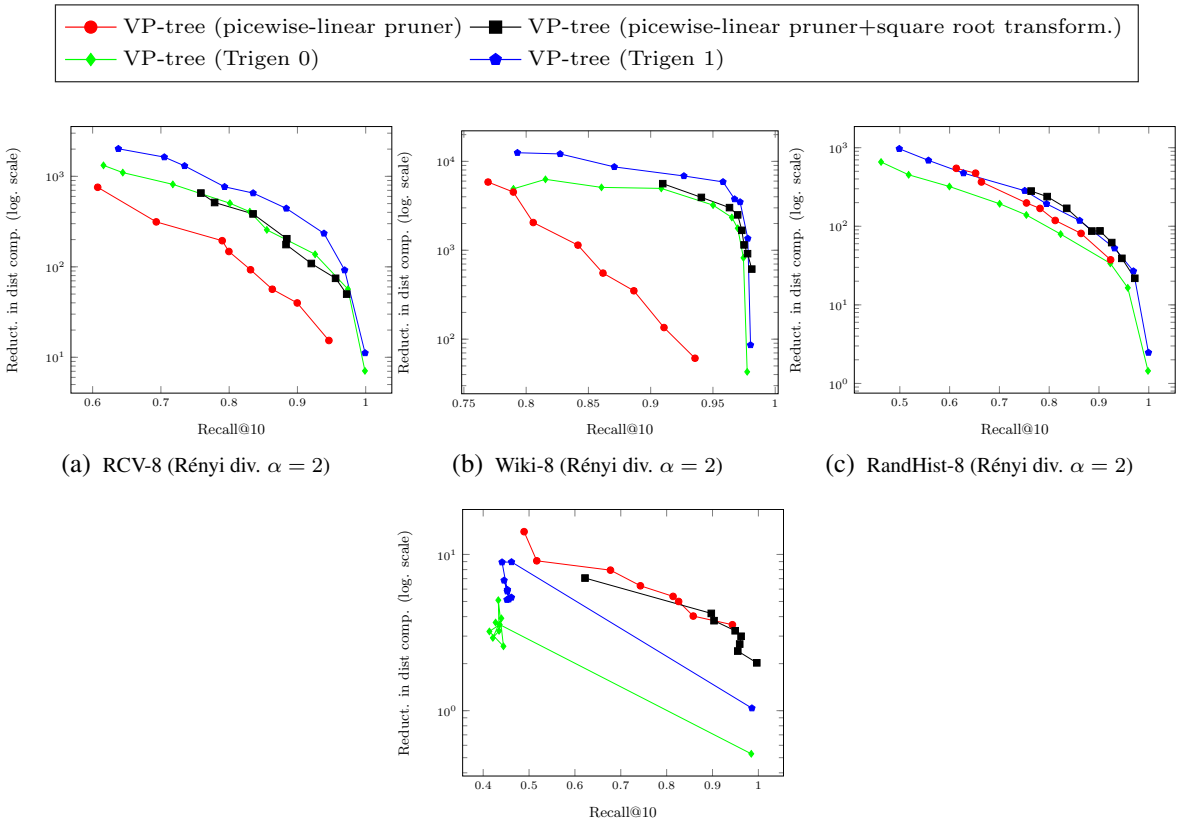

(a) RCV-8 (KL-div.) (b) RCV-8 (Rényi div. $\alpha = 0.25$) (c) RCV-8 (Rényi div. $\alpha = 0.75$)

(d) Wiki-8 (KL-div.) (e) Wiki-8 (Rényi div. $\alpha = 0.25$) (f) Wiki-8 (Rényi div. $\alpha = 0.75$)

(g) RandHist-8 (KL-div.) (h) RandHist-8 (Rényi div. $\alpha = 0.25$) (i) RandHist-8 (Rényi div. $\alpha = 0.75$)

(j) Wiki-128 (KL-div.) (k) Wiki-128 (Rényi div. $\alpha = 0.25$) (l) Wiki-128 (Rényi div. $\alpha = 0.75$)

Figure 2.23: Reduction in the number of distance computations vs recall for VP-tree based methods in 10-NN search (part III). Best viewed in color.




(a) RCV-8 (Rényi div. $\alpha = 2$) (b) Wiki-8 (Rényi div. $\alpha = 2$) (c) RandHist-8 (Rényi div. $\alpha = 2$)

(d) Wiki-128 (Rényi div. $\alpha = 2$)

Figure 2.24: Reduction in the number of distance computations vs recall for VP-tree based methods in 10-NN search (part IV). Best viewed in color.

Recall that in a typical space-partitioning method, we divide the data into reasonably large buckets (50 in our experiments). The $k$-NN search is simulated as a range search with a shrinking radius. In the case of the TriGen with symmetrization, while we traverse the tree, we compute the original and the symmetrized distances for two purposes:

- shrinking the dynamic radius of the query using the *symmetrized* distance;
- checking if the *original* distance is small enough to update the current set of $k$ nearest neighbors.

When we reach a bucket, for every data point in the bucket, we can compute both the original and the symmetrized distance. The symmetrized distance is used to update the query radius, while the original distance is used to update the set of $k$ nearest neighbors. This is our first modification of TriGen which we refer to as *Trigen 0*.

In the second variant of TriGen, which we refer to as *Trigen 1*, we use only the original distance for bucket data points. In this setup, when we process bucket data points, we shrink the dynamic query radius using values of $f(d(x,y))$ instead of $\min\left(f(d(x,y)), f(d(y,x))\right)$, We expect the radius to shrink somewhat slower compared to the full symmetrization, which would reduce the effectiveness of pruning. However, we hope that nearly halving the number of distance computations would have a larger effect on overall retrieval time.

TriGen can be used with any metric space search method. However, for an equity of comparison against our data-dependent pruning approach, we use TriGen with our implementation of VP-tree (bucket size is 50 as in other experiments). As explained above, we use two variants of TriGen (Trigen 0 and Trigen 1), none of which employs a filtering step. The core TriGen algorithm that finds an optimal transformation function has been downloaded from the author's website[25] and incorporated into NMSLIB. The optimization procedure employs a combination of parameters $a$ and $b$, where $a$ are multiples of 0.01, $b$ are multiples of 0.05, and $0 \leq a < b \leq 1$. The sampling parameters are set as follows: `trigenSampleTripletQty`=10000, `trigenSampleQty`=5000.

TriGen is compared against two variants of our VP-tree. In both variants we employ a piecewise linear approximation for the pruning rule that is learned from data as described in § 2.2.3. They are nearly identicial search methods, but the second variant uses a clever TriGen idea of applying a concave mapping to make the distance more similar to a metric one. As mentioned on p. 50, the empirically obtained pruning rule is not truly piecewise linear (see Figures 2.7-2.8), but the concave mapping reduces non-metricity of the distance. Consequently, the piecewise linear pruning rule becomes a more accurate approximation of the actual pruning rule. Unlike TriGen [271], we do not carry an extensive search for an optimal transformation but rather apply, perhaps, the simplest and fastest *monotonic* concave transformation possible, which consists in taking a square root.[26] Also note that for Bregman divergences the square root transformation makes a distance to be $\mu$-defective (see Eq (2.1) [1]. In that, the respective pruning decision function is piecewise linear.

In this series of experiments, we employ 40 combinations of data sets and distances. All distances are non-metric: We experiment with both symmetric and non-symmetric ones. Because, tree-based solutions are useful mostly for low-dimensional data, we focus on low-dimensional

[25] `http://siret.ms.mff.cuni.cz/skopal/download.htm`

[26] On Intel and a reasonably modern CPU, the square root is computed the instruction `sqrtss`, which typically takes less than 10 cycles [113].

data sets: Only six test cases employ 128-dimensional vectors. In each test, we randomly split data (into queries and indexable data) three times and average results over three splits. Each split uses 1K queries for all data sets (in each test, the total number of queries is 3K). We generate multiple runs with different accuracies. For TriGen-based methods, we vary the accuracy thresholds from 0.75 to 1. For our solutions learning the pruning rule, we vary the parameter `desiredRecall` (also from 0.75 to 1).

The experimental results are presented in two sets of tables: Tables 2.17-2.20 and Tables 2.21-2.24. In the first set of tables, we measure efficiency directly in terms of wall-clock time improvement over the brute-force search. In the second set of tables, we show improvement in the number of distance computations (again compared to the brute-force search). We make the following observations:

- There is generally little difference between two TriGen variants in most cases. However, for some data sets and recall values, TriGen 1 can be nearly twice as efficient compared to TriGen 0.
- When comparing TriGen against our VP-tree with the piecewise linear pruner in terms of pure efficiency, the results are a bit of the mixed bag. Yet, the piecewise linear pruner is better (or mostly better) in 23 plots out of 40.
- However, the piecewise linear pruner combined with the square-root distance trasnform is nearly always better than the basic piecewise linear pruner. In Panels 2.17d,2.17e, 2.19a, 2.19b, 2.19c the improvment is in the order of magnitude.
- The combination of the piecewise linear pruner with the square root transform outperforms TriGen in all but two cases, often by an order of magnitude. In Panels 2.19d and 2.18f, however, TriGen can be an order of magnitude faster than our solutions.
- Interestingly, when comparing our VP-tree methods against TriGen, there is little to no difference in terms of the reduction in the number of distance computations (see Tables 2.21-2.24). The likely explanation for this discrepancy is that the transformation functions used in the adopted TriGen implementation is expensive to compute.

## 2.4 Summary

In this section, we carry out multiple intrinsic evaluations (with a focus on high accuracy), where we study efficiency-effectiveness trade-offs of *generic* methods for $k$-NN search. This evaluation consists in cross-method comparison using diverse data (§ 2.3.1) and additional benchmarks using only non-metric data (§ 2.3.2). In § 2.3.2, we use only a limited set of search approaches, which include the brute-force search, SW-graph (§ 2.2.2.3), and two approaches relying on the classic VP-tree with a modified pruning rule.

In our experiments with diverse data, we have tested several types of approximate $k$-NN search methods for generic spaces (where both data and indices are stored in main memory). In particular, we have assessed several implementations of pivoting methods known as permutation methods.

First, we demonstrate that it is, indeed, possible to modify the pruning rule of VP-tree using a simple learning algorithm. The resulting search method works reasonably well for a variety

of metric and non-metric data sets of moderate and small dimensionality. Note, however, that VP-trees do not work well in the case of a sparse high-dimensional data. This is unfortunate, because high-dimensional data is used in many ML and IR applications. However, it may be relevant to motion planning problem commonly encountered in robotics, where $k$-NN search is applied to low-dimensional data with (sometimes) non-Euclidean distance [162, 163].

Second, we show that proximity-graph based methods have outstanding performance for a variety of metric and non-metric data sets. In particular, these methods are efficient and accurate in the case of the non-symmetric KL-divergence. We believe that this has not been previously known. In the second series of experiments, we learn, however, that the data sets employed in our preliminary experiments [220, 245] are only mildly non-symmetric: It is possible to obtain good results using a filter-and-refine approach where filtering uses a symmetrized distance. We, nevertheless, observe that for substantially non-symmetric data SW-graph still delivers excellent performance. Furthermore, even for mildly non-symmetric data best results are achieved without using a filter-and-refine approach with the symmetrized distance. Somewhat surprisingly, however, performance can be improved by using the symmetrized distance at indexing time only, i.e., to construct a proximity graph.

Third, we note that results for permutation-based search methods are a mixed bag. Permutation methods can work reasonably well for a variety of data sets. The best performance is achieved either by NAPP or by brute-force filtering of permutations. For example, NAPP can outperform the multi-probe LSH in $L_2$.

However, permutation methods can be outstripped by either proximity-graph based methods (in high dimensions) or by VP-tree (in low dimensions). Permutation methods do work well when permutation-based projections are effective and the distance function is expensive. However, we observe that the quality of permutation-based projections varies quite a bit: In particular, the projections are more accurate for symmetric metric distances than for non-symmetric KL-divergence. We, therefore, conclude that generality of permutation-based methods is limited.

What is, perhaps, most discouraging is that performance of the permutation-based method NAPP leaves a lot to be desired in the case of high-dimensional sparse data (see Panel 2.11i). However, these results are obtained by selecting pivots randomly from the data set. In § 3.1.2.2 we adopt a novel pivot generation method (proposed by our co-author David Novak), which delivers a much stronger performance.

Fourth, from comparing TriGen with our piecewise-linear pruning approach, we observe the following: (1) the best results are achieved by combining two approaches, i.e., by applying the piecewise-linear pruner to a square-root transformed distance, (2) using an expensive distance transformation function (RBQ) can dramatically reduce performance. We hypothesize that even better results can be obtained by implementing some cheap approximation of a fractional-power (FP) transforming function with the piecewise-linear pruner. The FP-transformer can be much more accurate than a simple square-root transform, while remaining close in terms of efficiency.

To conclude this section, we note that the learned pruning rule (possibly combined with a distance stretching) can be applied to other metric space search methods such as cover trees [35, 145], the multi-vantage point tree [46], or the MI-index [232]. However, our initial experiments with the multi-vantage point trees show that although the latter method requires fewer distance computations (sometimes substantially so), it is slower than the VP-tree. More advanced methods can prune the search space more effectively (e.g., by using more pivots for each partition), but

added pruning effectiveness entails higher bookkeeping costs (perhaps, due to a higher branch misprediction rate). If we can lower bookkeeping costs by, e.g., using code with fewer branches and/or employing branchless SIMD instructions, we may obtain even faster search methods for intrinsically low-dimensional data.

# Chapter 3

# Extrinsic Evaluation: Applying $k$-NN to Text Retrieval

In this chapter, we employ several question-answering (QA) data sets, to test the conjecture that for certain domains $k$-NN search can replace or complement search algorithms relying on term-based inverted indices. We consider a setting, where a question text (possibly combined with a longer description, linguistically-motivated features, and other meta-data) is submitted to a retrieval engine with an objective of finding text passages containing an answer (i.e., *answer-bearing passages*). Unlike the traditional filter-and-refine approach, which generate candidate documents using a variant of an inner-product similarity, we explore methods where the candidate-generation step is "guided" by a more complicated similarity. In this case a term-document similarity matrix is dense and the similarity function is not decomposable, i.e., it cannot be computed as a sum of term-specific values.

A typical scenario of using $k$-NN search involves a cheap similarity/distance, e.g., the cosine similarity, and an approximate search algorithm to speed up the search (e.g., LSH). Although in some cases an approximate search can slightly outperform the exact search with the same distance/similarity—see, e.g., Table 3 in [45] (column four, row four from the bottom)—we believe that approximation typically compromises retrieval accuracy. In a high-dimensional setting, we expect substantial gains in efficiency to be obtained only at the expense of substantial loss in accuracy. In particular, this is true for all data sets, similarities, and search algorithms employed in this chapter (see § 3.3.3.2 for the results and the discussion).

Unlike most prior work, we attempt to replace the cheap distance with a more accurate one. Because the more accurate distance is computationally more expensive, we employ an approximate $k$-NN search to reduce retrieval time while retaining some of the accuracy advantage over the simpler distance. We hope that using the more sophisticated computationally expensive similarity combined with approximate search may allow us to be simultaneously more efficient and effective. Additionally we hope that a more accurate similarity function may be able to find document entries that are hard or impossible to obtain using a standard retrieval pipeline based on a simple inner-product similarity function.

Our task belongs to the domain of automatic ad hoc text retrieval, where the goal is to retrieve documents relevant to an arbitrary information need specified by the user in a free textual form [19, 199, 261]. Other uses of $k$-NN search in IR and NLP include classification

[133, 161, 173, 174, 300], entity detection [189, 308], distributional similarity [122, 207, 291], first story detection [183, 210, 240, 241], collaborative filtering [33, 165], and near-duplicate detection [50].

In the context of IR and NLP, there are two main approaches to carry out a $k$-NN search. The first approach relies on a term-based inverted index to find documents that share common terms with the query. These documents are further ranked using a similarity function, which is typically some variant of a inner-product similarity similarity. Dynamic and static pruning can be used to improve efficiency, sometimes at the expense of decreased recall [94, 274, 292]. This approach supports arbitrary similarity functions, but it suffers from the problem of the vocabulary mismatch [32, 115, 280].

The second approach involves carrying out the $k$-NN search via LSH [183, 210, 240, 290]. It is most appropriate for the cosine similarity. For example, Li et al. [183] propose the following two-stage scheme to the task of finding thematically similar documents (i.e., in a *classification* task). In the first step they retrieve candidates using LSH. Next, these candidate are re-ranked using the Hamming distance between quantized TF×IDF vectors (first transformed using PCA [88, 116]). Li et al. [183] find that their approach is up to 30× faster than the classic term-based index while sometimes being equally accurate. We want to emphasize that this result is for the classification task, but we are not aware of the similar result for ad hoc retrieval.

Petrović et al. [240] applied a hybrid of LSH and the term-based index to the task of the streaming First Story Detection (FSD). The LSH keeps a large chunk of sufficiently recent documents, while the term-based index keeps a small subset of recently added documents. They report their system to be substantially faster than the state-of-the-art system—which relies on the classic term-based index—while being similarly effective. In a follow up work, Petrović et al. [241] incorporate term associations into the similarity function. Their solution relies on an approximation for the kernelized cosine similarity. The associations are obtained from an external paraphrasing database. Moran et al. [210] use the same method as Petrović et al. [241], but find synonyms via the $k$-NN search in the space of word embeddings (which works better for Twitter data). Moran et al. [210] as well as Petrović et al. [241] calculate performance using an aggregated metric designed specifically for the FSD task. Unfortunately, they do not report performance gains using standard IR metrics such as precision and recall.

To wrap up with the discussion of related work, we want to emphasize that existing studies are biased in at least two ways. First, there is a trend to test $k$-NN search in a classification-like setting, where we determine an assignment of a data point based on the assignment of nearest neighbors. It may be easier to achieve good accuracy in this task, because of redundancy: To determine a class, we do not need to know all true nearest neighbors. In fact, applying a data set compression technique, which removes most of data points (and potentially retrofits existing ones), can lead to improved performance in a $k$-NN classification task (see, e.g., a recent work by Kusner et al. [173]). Thus, the use of approximate search, which misses some of the neighbors, does not necessarily lead to a substantial reduction in accuracy.

Second, all the studies that we are aware of rely on the cosine similarity. However, as shown in the literature (see [314] and references therein), cosine similarity for bag-of-words representation is not especially effective. This is also supported by experiments of Boytsov et al. [45] (see Figure 2), where the TF×IDF cosine similarity finds 2× *fewer* answers for any given rank $N$ compared to BM25 similarity [254] (both the cosine similarity and BM25 are equally efficient).

| Notation | Explanation |
|---|---|
| $Q$ | a question |
| $q$ | a question term |
| $A$ | an answer (or an answer-bearing passage) |
| $a$ | an term in an answer (or in an answer-bearing passage) |
| $P(a\|A)$ | a probability that a term $a$ is generated by the answer $A$ |
| $P(q\|C)$ | a probability that a term $q$ is generated by the entire collection $C$ |
| $T(q\|a)$ | a probability that question term $q$ is a translation of answer term $a$ |
| $t$ | a configuration that consists of a single term $t$ |
| $\#\text{od}N(t_1 t_2 \dots t_m)$ | an "ordered" configuration of terms $t_1$, $t_2$, $\dots t_m$ within a window of $N$ terms (terms order in the window is restricted) |
| $\#\text{uw}N(t_1 t_2 \dots t_m)$ | an "unordered" configuration of terms $t_1$, $t_2$, $\dots t_m$ within a window of $N$ terms (terms order in the window is arbitrary) |
| $M$ | a result set size in top-$M$ retrieval |
| $N_D$ | a number of documents in a collection |
| $N_C$ | collection size, i.e., a total number of terms in a collection |
| $\|q\|$ | a number of query terms |
| $\|D\|$ | document length: a number of document terms |
| $\|D\|_{\text{avg}}$ | the average length of a document in a collection |
| TF$(t)$ | an in-document frequency of term configuration $t$ (it can be more than one term), i.e., a number of occurrences of term configuration in the document |
| DF$(t)$ | document frequency, i.e., a number documents containing term configuration $t$ |
| CF$(t)$ | collection frequency, i.e., a total number of occurrences of term configuration in the collection |
| IDF$(t)$ | an inverted document frequency of term configuration $t$ |
| $\text{R}_{\text{knn}}@k$ | $k$-NN recall at rank $k$, i.e., an overlap between $k$ true nearest neighbors and $k$ data points returned by a search method |
| Recall@k | answer recall, i.e., a fraction of answer-bearing passages returned by a method. |
| ERR@$k$ | Expected Reciprocal Rank at rank $k$ [61] |
| NDCG@$k$ | Normalized Discounted cumulative gain at rank $k$ [148] |

Table 3.1: Notation of Chapter 3

Furthermore, Boytsov et al. [45] find that the cosine similarity between dense representations (averaged word embeddings) is even less effective. In contrast to this finding, Brokos et al. [52] show that the cosine-similarity between averaged word embeddings is an effective model for retrieving Pubmed abstracts. However, they do not compare against standard IR baselines such as BM25, which makes an interpretation of their finding difficult. Also note that Pubmed abstracts are quite short and results could would have been worse if documents were substantially longer.

The BM25 similarity—see Eq. (19)—is a variant of an an inner-product, similarity. As we explain in § 2.1, there is a simple yet ingenious data transformation such that BM25 scores for the original data are equal to cosine similarity scores for modified data times a query-dependent constant [225] (see Exposition 2.5 for more details). In principal, this permits the use of LSH methods, however, we are not aware of any work that study efficiency-effectiveness trade-offs in this setting. Also note that such transformations are unlikely to exist for many complex similarities especially when they are *non*-symmetric (e.g., similarity BM25+Model 1 described in § 3.1.1.2). These observations motivated us to explore search methods that may work in a more generic setting, in particular, when symmetry is lacking.

The rest of the chapter is organized as follows. In § 3.1, we describe our retrieval task, similarity models, and search methods. In § 3.2, we explore approaches to design an effective similarity function that combines linguistically-motivated and lexical features. We then evaluate efficiency-effectiveness trade-offs of our models in § 3.3. In § 3.4, we summarize the main findings of this chapter. Notation used in this chapter is summarized in Table 3.1.

## 3.1 Approach

In this chapter, we focus on a task of finding answer-bearing passages extracted from three text collections. Question answering is an important task on its own. However, here we use QA data sets primarily as a testbed to demonstrate the potential of the $k$-NN search as a substitute for term-based retrieval. To weaken the curse of dimensionality and to design sufficiently generic $k$-NN search methods, we resort to approximate searching (see § 2.1.3 for a discussion that motivates the necessity of approximate search). Because of this, we need a similarity function that outstrips the baseline method BM25 by a good margin. Otherwise, gains achieved by employing a more sophisticated similarity would be invalidated by the inaccuracy of the search procedure.

One effective way to build such a similarity function is to learn a generative question-answer *translation* model, e.g., IBM Model 1 [53]. However, "... the goal of question-answer translation is to learn associations between question terms and synonymous answer terms, rather than the translation of questions into fluent answers." [253] The idea of using a translation model in retrieval applications was proposed by Berger et al. [32]. It is now widely adopted by the IR and QA communities [101, 115, 253, 275, 280, 319]. Linearly combining BM25 and *logarithms* of IBM Model 1 scores (using a simple learning-to-rank algorithm called *coordinate ascent* [205]) produces a similarity function that is 18-30% more accurate than BM25 alone (see Table 3.6). Coordinate ascent is designed to maximize the value of a *target* metric rather than a surrogate loss function. Thus, it is quite effective when the number of features is small (for a large number of features, the training process becomes infeasibly long). In particular, on our data set it either

| Question type | Question | Answer |
|---|---|---|
| | *WikiSQuAD* | |
| location | Where can one find the formerly Huguenot farms in South Africa? | Western Cape province |
| definition/description | What is the mission of FAN (Firefighters Volunteer Association of Nepal)? | Raising public awareness about fire and improving safety |
| causation/reason | Why are the small lakes in the parks emptied before winter? | To clean them of plants and sediments |
| | Yahoo Answers *Comprehensive* | |
| quantity | What is the hourly rate charged by tattoo artists in california? | Anywhere from $15 to $200/ hour |
| list | what religion are most people in America? | 1)Christianity the largest are A) Protestant B) Catholic C) Mormon D) Orthodox . . . |
| navigational/direction | hi are there any kid cartoons online that they can watch for free or really cheap? | go to www.yahooligans.com then click on TV |
| manner/procedure | How to get rid of a beehive? / Who can tell me how to get rid of a beehive with out getting stung? | Call an area apiarist. They should be able to help you and would most likely remove them at no charge in exchange for the hive. The bees have value and they now belong to you. |
| causation/reason | In the NBA, why do the players wear the black tights/leggins? / Dwayne Wade for the Miami Heat wears black tights underneath his uniform, but what are they for? | It keeps them warm so they can run looser and because the NBA overcools arena's. |
| opinion | Role-reversal? / Is it attractive to see women doing men's jobs, and vice versa? | I guess it would depend on the job like what they were doing and how well they were doing it. |
| explanation | how banks manage there databases, what kind of technology it is called, maintaining huge databases.? | Hi, Banks and other organisations mange their databases through softwares known as DBMS . . . |
| | *Stack Overflow* | |
| manner/procedure | Avoid counter increment in Python loop / How can I avoid having to increment the counter manually in the following Python code: . . . | Enumerate! You can refer to the Python docs: `for sequence, value in enumerate(...` |
| explanation | How is a three dimensional array stored in memory? / . . . For example, a two dimensional array is stored in terms of rows and columns in the case of three a dimensional array how are the elements arranged in c? | Multidimensional arrays are laid out contiguously in memory, not as rows and columns. . . . |
| opinion | Silverlight 4 macintosh compatibility / . . . I'd like to believe that any program whatsoever written for Silverlight will magically run the same on Mac and Windows, but I find it very hard to believe that this is the case. | . . . Silverlight 5 has a few features that almost certainly will be windows-only (P/Invoke comes to mind), and early reports are that Silverlight will find a home as the managed-code runtime for Windows 8. |

Table 3.2: Sample question, answers, and question types for three collections (Slashes are used to separate the main question from an optional description; some questions/answers are shortened)

outperforms all other learning-to-rank methods implemented in the library RankLib[1] or matches their performance for all feature combinations of interest.

Learning IBM Model 1 requires a large monolingual parallel corpus. In that, QA data sets seem to be the best publicly available sources of such data. Note that a monolingual corpus can be built from search engine click-through logs [252]. Yet, such data is not readily available for a broad scientific community. Another advantage of the QA data sets that we employ is that they permit a large scale automatic evaluation with sizable training and testing subsets (all our testing subsets have at least 10K queries), where a question (possible extended with a question description) plays a role of the query and answers are considered to be the only relevant documents. Having thousands of queries can be essential to reliable experimentation [44]. A drawback of such data sets is that there are few relevant documents per query, i.e., the pool of relevant documents is shallow. However, there is evidence that increasing the size of a query subset is more useful than increasing the depth of the pool [236, 258].

Specifically, we use the following collections:

- Stanford *WikiSQuAD* questions expanded with Wikipedia passages [249];
- L6 - Yahoo Answers *Comprehensive* version 1.0;[2]
- L5 - Yahoo Answers *Manner* version 2.0: a *subset* of L6 created by Surdeanu al. [280];
- *Stack Overflow* .[3]

Yahoo Answers collections are available through Yahoo WebScope and can be requested by researchers from accredited universities.[4] For each question, there is always an answer (and the best answer is always present). The smaller Yahoo Answers collection *Manner* is a subset of Yahoo Answers *Comprehensive* (created by Surdeanu al. [280]), which includes only *manner/procedure* QA pairs that passed a quality filter. This subset is only used to compare against a previously published result (to ensure that we employ a state-of-the-art similarity model). We typically call these Yahoo collections as simply *Manner* and *Comprehensive*. The *Stack Overflow*

| Question type | Number of occurr. |
|---|---|
| SQuAD | |
| definition/description | 30 |
| person | 18 |
| location | 13 |
| duration | 8 |
| number/amount | 8 |
| date | 6 |
| year | 6 |
| causation/reason | 3 |
| company | 3 |
| opinion | 2 |
| color | 1 |
| country | 1 |
| list | 1 |
| Yahoo Answers *Comprehensive* | |
| opinion | 32 |
| manner/procedure | 16 |
| non-English | 16 |
| navigational/direction | 13 |
| definition/description | 10 |
| explanation | 7 |
| list | 5 |
| number/amount | 4 |
| personal/behavior | 3 |
| causation/reasons | 2 |
| spam | 1 |
| *Stack Overflow* | |
| manner/procedure | 67 |
| explanation | 44 |
| opinion | 9 |

Table 3.3: Answer types for a sample of 100 questions for 3 collections.

[1] https://sourceforge.net/p/lemur/wiki/RankLib/
[2] L6 collection in Yahoo WebScope: https://webscope.sandbox.yahoo.com
[3] https://stackoverflow.com/
[4] https://webscope.sandbox.yahoo.com

collection is freely available for download.[5] While there are 8.8M answered questions, the best answer is not always selected.

The Stanford *WikiSQuAD* data set [249] contains more than 100K questions created by crowdworkers for a subset of curated Wikipedia articles. These articles were further split into paragraphs. In that, paragraphs shorter than 500 characters were discarded. Overall, crowdworkers created questions for 23,215 paragraphs from 536 articles covering a wide range of topics. Each question was on a content of a paragraph. An answer, which was also selected by a crowdworker, had to be a short paragraph span, e.g., a noun-phrase, or a sentence clause. We expanded *WikiSQuAD* by adding paragraphs from articles not included in the *WikiSQuAD* data set. To this end, we used Wikipedia dump dated by 01/01/2017. It was processed and split into a paragraphs using a slightly modified version of `WikiExtractor` [14].[6].

Questions are often classified into two broad groups: factoid and non-factoid questions. A factoid—in the modern sense of the word—is a briefly stated piece of information often of a trivial nature.[7] Therefore, answers to factoid questions are succinct utterances describing simple facts or relationships, which are often expressed via named entities and other noun phrases [247]. For example, an answer to the factoid question "What is the capital of the United States?" is the named entity "Washington, DC".

Although intuitively clear, an *accurate* division of questions into factoids and non-factoids is difficult. Consider, for example question and answers listed in Table 3.2. The first sample question from the *WikiSQuAD* collection is about location. It has a short answer "Western Cape province", which is a noun phrase associated with a named entity. The second and the third sample questions from the *WikiSQuAD* collections are answered by short clauses rather than noun phrases. Thus, we expect some disagreement about whether these questions can be considered factoid. Overall, however, *WikiSQuAD* seems to be largely a factoid collection. *WikiSQuAD* creators find that an answer to 52.4% questions is a named entity, a date, or a amount/number. In addition, 31.8% of answers are expressed as a common-noun phrase, e.g., a *property damage*. This agrees well with our own assessment of randomly sampled 100 QA pairs (see Table 3.3), where 65 (out of 100) answers are named entities (e.g., locations or companies), numbers, dates, colors. There are also 30 *definition/description* questions and answers many of which may be interpreted as factoid.

Compared to *WikiSQuAD* collection, answers in Yahoo Answers *Comprehensive* and *Stack Overflow* collections seem to be more expansive. Occasionally, we can see succinct answers (e.g., in the case of the *quantity* and *navigational* questions in Yahoo Answers), but more frequently answers are long and include several sentences. This is particularly true for *explanation* and *manner/procedure*, i.e., how-to, questions. To obtain additional insight about the nature of these test collections, we sampled 100 QA pairs from each of the collection and manually assessed their types (for some QA pairs more than one type was assigned). Results are presented in Table 3.3.

We can see that there is much less diversity in the *Stack Overflow* collections. This is not surprising, because most users ask how to implement a code for some task (*manner/procedure*

[5]We use a dump from `https://archive.org/download/stackexchange` dated March 10th 2016.

[6]The modification was required, because the original `WikiExtractor` does not properly define paragraph boundaries. The updated extractor is available at `https://github.com/searchivarius/wikiextractor`

[7]See, e.g., the definitions in the Wikipedia `https://en.wikipedia.org/wiki/Factoid#Usage` and Merriam-Webster dictionary `https://www.merriam-webster.com/dictionary/factoid`.

questions), seek help with the code that does not work (*explanation* questions), or occasionally solicit an *opinion* on a somewhat controversial topic. None of such answers and questions can, perhaps, be considered factoid. In contrast, in Yahoo Answers *Comprehensive*, we can occasionally encounter factoid questions, but more than half of the questions are either a *manner/procedure* questions or they ask for an *opinion* or *explanation*. These questions have long non-factoid answers. To sum up, both Yahoo Answers *Comprehensive* and *Stack Overflow* are largely non-factoid collections.

We process collections by removing punctuation, extracting tokens and term lemmas using Stanford CoreNLP [200] (instead, one can use any reasonably accurate tokenizer and lemmatizer). All terms and lemmas are lowercased. We further remove terms that appear in the stopword list of the search engine Indri version 5.10 [279][8]. In addition, we remove the frequent term `n't`, which results from tokenizing words such as `don't` and `can't`. In *Stack Overflow* we remove all the code (the content marked by the tag `code`). Although our main experiments rely on lemmatized text only, in exploratory experiments we use *both* lemmas and original terms as well as additional linguistic annotations such as named entities and semantic role labels (see § 3.2.2).

Using the standard methodology, we split each collection is randomly split into several subsets, in a collection-specific way. For all collections we generate one training, two development (*dev1* and *dev2*), and one testing subset. For all large collections, however, there are additional subsets as we explain below. Each community QA collection is split at the level of individual questions: We randomly assign a question *together* with the associated answers to the same subset, e.g., *dev1*. Each community QA collection answer represents an answer-bearing passage. We do not use cross-validation, because we have enough data for separate training, development, and testing subsets. Unlike the cross-validation setting, we do not employ the test data until after all tuning and exploration is done using development sets.

The *WikiSQuAD* collection is randomly split at the level of a Wikipedia paragraph. Each such paragraph is an answer-bearing passage (because crowdworkers create a question based on a paragraph's content). However, our partition is unrelated to the partition of *WikiSQuAD* generated by its creators, who split at the level of the Wikipedia article. Also note that we use passages only from the development and training subsets, because the official held-out *test* set is not visible to the public.

Furthermore, recall that we expand *WikiSQuAD* with passages from Wikipedia articles not belonging to the set of curated articles from which *WikiSQuAD* was created. These passages do not have answers and are placed deterministically into a separate subset. Answers (but not questions) from *all* subsets are subsequently indexed using various approaches, which include traditional inverted indices as well as indices for $k$-NN search.

Collection statistics is summarized in Table 3.4. We can see that all three main collections: *Comprehensive*, *Stack Overflow*, and *WikiSQuAD* are reasonably large. In that, *Stack Overflow* has the smallest (11.5M) and *WikiSQuAD* has the largest (30.1M) number of answers. In the case of community QA data sets *Stack Overflow* and *Comprehensive*, most questions have answers. In that, *Comprehensive* has more answers per question than *Stack Overflow*. In *Stack Overflow* and *Comprehensive* questions are long and answers (answer-bearing passages) have lengths similar to those of questions. However, in the case of *WikiSQuAD* , on average, an answer-bearing passage

[8] `http://www.lemurproject.org/indri.php`

| | *Manner* | *Comprehensive* | *Stack Overflow* | *WikiSQuAD* |
|---|---|---|---|---|
| **total** | | | | |
| Number of questions | 142627 | 3887111 | 6083371 | 99169 |
| Number of answer-bearing passages | 142627 | 22389010 | 11489831 | 30111896 |
| **train** | | | | |
| Number of questions | 85570 | 50000 | 50000 | 5250 |
| **dev1** | | | | |
| Number of questions | 7119 | 9264 | 9774 | 2448 |
| **dev2** | | | | |
| Number of questions | 21355 | 36606 | 19492 | 2636 |
| **test** | | | | |
| Number of questions | 28583 | 10000 | 10000 | 10570 |
| **Terms per entry in a complete collection** | | | | |
| Questions | 13.9 | 16.2 | 43 | 5 |
| Answer-bearing passages | 40.6 | 21.6 | 26.7 | 34.4 |
| **Terms per entry in train/dev1/dev2/test** | | | | |
| Question | 13.9 | 16.2 | 43.1 | 5 |
| Answer | 40.5 | 21.7 | 26.7 | 65.6 |

Table 3.4: Collection statistics

| | *Comprehensive*\ *Manner* (best answers only) | *Comprehensive* | *Stack Overflow* | *WikiSQuAD* |
|---|---|---|---|---|
| # of QA pairs | 4188046 | 20751778 | 11014341 | 77156 |
| # of terms per question | 17.9 | 17.3 | 42.7 | 5 |
| # of terms per answer | 33.9 | 21.6 | 26.7 | 16.1 |

Table 3.5: Statistics of parallel corpora

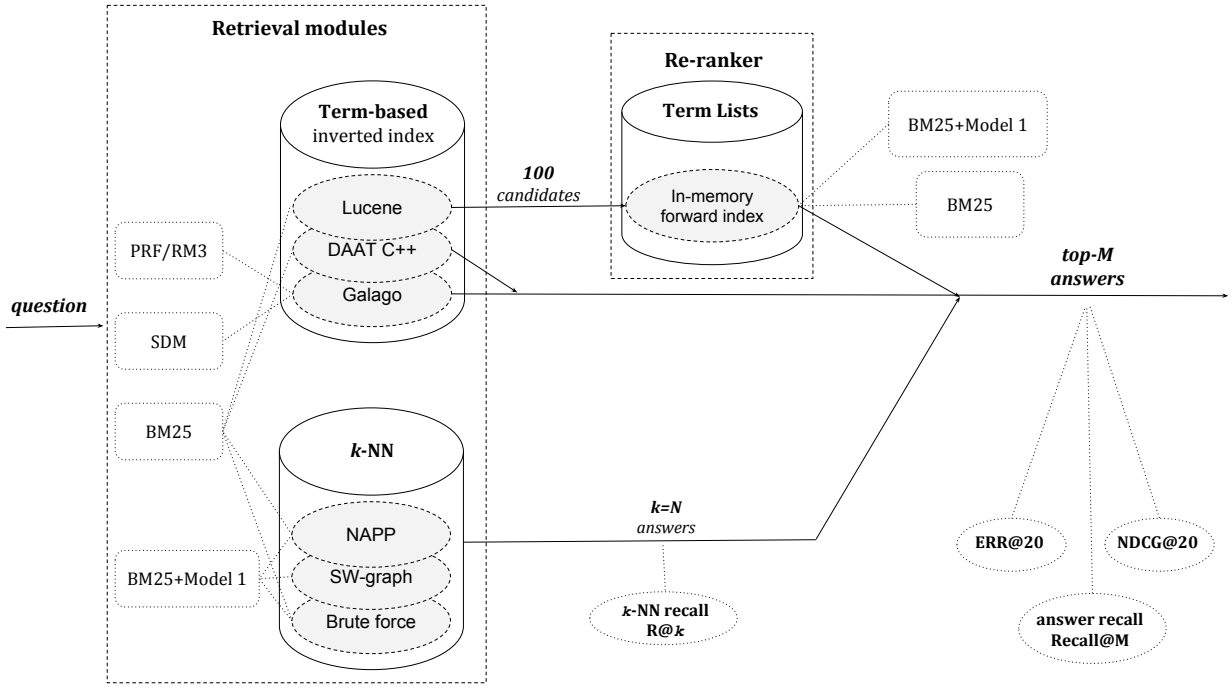



Figure 3.1: Retrieval pipeline architecture. We illustrate the use of evaluation metrics by dotted lines (metric names are placed inside ovals in the bottom right part of the diagram). Dotted lines also indicate which similarity models are used with which retrieval/re-ranking components. Similarity models are shown using rectangles with round corners and dotted boundaries.

is much longer than the respective question. Also note that, unlike community QA data, only about 100K *WikiSQuAD* passages have an associated question.

In addition, to the development, training, and testing subsets, we carve out large portions of collections to create parallel corpora to train IBM Model 1 (see § 3.1.1.2). In the case of *Manner* , the parallel corpus is created from a subset of *Comprehensive* that (1) does not contain questions from *Manner* and (2) employs only best answers. The split of *Manner* mimics the setup of Surdeanu et al. 2011. In the case of community QA data *Comprehensive* and *Stack Overflow*, the parallel corpus is trained on a large subset of a collection that does not overlap with training, testing, and development sets. In the case of *WikiSQuAD* , we use two translation models. One is trained on a subset of *WikiSQuAD* (see § 3.2.1 for details) and another one is the same model that we train on the parallel corpus extracted from *Comprehensive*. Statistics of parallel corpora is given in Table 3.5.

Our task—which belongs to the domain of ad hoc text retrieval—consists in finding answer-bearing passages using questions as queries. Questions are pre-processed in the same way as answer-bearing passages, i.e., we lemmatize terms and remove terms appearing on the stopword list. In the case of community QA data, a question is accompanied by an optional description, which is also pre-processed and appended to the query. The known answer-bearing passages are considered to be the only relevant documents. Thus, in the case of *WikiSQuAD*, the only relevant document for a question is a passage from which the question is created. In the case of

| | BM25 Lucene | BM25 | | BM25+Model 1 | | Relative improvement | |
|---|---|---|---|---|---|---|---|
| | Recall@100 | NDCG@20 | ERR@20 | NDCG@20 | ERR@20 | NDCG@20 | ERR@20 |
| Yahoo Answers *Comprehensive* | | | | | | | |
| best answers only (graded) | 0.1561 | 0.0648 | 0.0492 | 0.0772 | 0.0600 | 19.13% | 21.96% |
| graded/different scores | 0.1029 | 0.0758 | 0.0872 | 0.0898 | 0.1058 | 18.49% | 21.23% |
| graded/same scores | 0.1029 | 0.0793 | 0.1308 | 0.0934 | 0.1561 | 17.89% | 19.36% |
| *Stack Overflow* | | | | | | | |
| best answers only (graded) | 0.1910 | 0.0803 | 0.0619 | 0.1043 | 0.0838 | 29.92% | 35.24% |
| graded/different scores | 0.1623 | 0.0825 | 0.0734 | 0.1073 | 0.0992 | 30.11% | 35.11% |
| graded/same scores | 0.1623 | 0.0844 | 0.0879 | 0.1097 | 0.1182 | 30.05% | 34.47% |

Table 3.6: Effectivenss for different relevance-grading and data-selection strategies (in a *re-ranking* experiment on a *dev1* set).

community QA data, a relevant document for a question is an answer to this question (provided by a community member). Accuracy of retrieval is measured using the following standard IR metrics:

- Answer recall (mostly at rank 100);
- Expected Reciprocal Rank [61] (ERR@20);
- Normalized Discounted Cumulative Gain [148] at rank 20 (NDCG@20).

Answer recall is equal to the fraction of answer-bearing passages among top-$k$ entries. This metric is binary: If an answer-bearing passage is present it gets a score one, if it is absent, the score is zero. NDCG@$k$ and ERR@$k$ are graded metrics, i.e., they can take into account the *degree* of relevance. In particular, we use only grades four (highly relevant document), three (somewhat relevant), and zero (not relevant).

In the case of $k$-NN search, we also measure values of intrinsic metrics: namely $R_{knn}$@20 and extrapolated RBO (see p. 12). In the case of RBO, there is an ad hoc parameter $p$, which models persistence of the user. Webber et al. [310] choose $p = 0.9$, because it assigns a weight of 86% to the set of top ten results. In the spirit of their choice (and because we compare with NDCG@20), we assign a weight of 90% to the set of top 20 results. This corresponds to the value of $p = 0.9372$.

In our preliminary experiments with these community QA collections [45], we discard all answers except best ones. Unfortunately, this reduces the number of answers in data sets by a factor of 3-5. For this reason, in the current evaluation, we include all questions and answers that pass simple a filter test proposed by Surdeanu et al. [280]: a valid question/answer should have at least four non-stop words, one noun, and one verb. An objective is to exclude completely irrelevant questions such as "I don't know".

It is not unreasonable to conjecture, however, that best answers have higher quality and, thus, should be considered more relevant. At the same time, there is evidence that this is not always the case, because the earlier an answer is posted, the more likely it is selected as the best answer [322]. To assess how serious this problem is, we evaluate accuracy of our best models using three grading/evaluation scenarios:

- Only the best answers are considered to be relevant: each relevant document has grade four;
- All answers are considered to be relevant: best answers have grade four, while other answers have grade three;
- All answers are considered to be relevant and have grade four.[9]

In all three cases, we compare the accuracy of BM25 with the accuracy of BM25+Model 1 (a linear combination of BM25 and log-scores of IBM Model 1). This is a re-ranking experiment: we retrieve 100 candidate entries with highest scores using Lucene (with BM25 similarity) and re-rank them using our implementations of BM25 and BM25+Model 1. Further increase in the number of top-$k$ entries retrieved by Lucene leads only to marginal improvement in effectiveness. Even though Lucene has an implementation of BM25 it has a deficiency (see explanation on p. 115), which leads to about 10% loss in accuracy. Hence, we do re-rank Lucene output even in the case of BM25 similarity and measure respective answer recall for different strategies to select indexable answers.

According to results in Table 3.6, the choice of the approach to evaluate relevance does affect raw values of all performance metrics. In particular, the smallest values of NDCG@20 and ERR@20 are achieved when only best answers are considered relevant. In this scenario, the pool of relevant documents contains only one document per query. In the scenario when all answers are considered to be relevant, the pools are deeper and contain nearly two relevant documents per query for *Stack Overflow* and more than five documents for *Comprehensive*. This permits retrieving more relevant documents per query (compared to the scenario where only best answers are considered to be relevant). Consequently, the values of NDCG@20 and ERR@20 are higher.

Importantly, if we consider a *relative* improvement for BM25+Model 1 over BM25, there is very little difference among three evaluation scenarios. In other words, results of the relative comparison are only marginally affected by a decision as to which relevance grade is assigned to non-best answers. In further experiments, we, therefore, grade all answers (that pass through a quality filter) in the same way by assigning the score four. Table 3.6 also provides additional evidence that the use of shallow pools in conjunction with large query sets permits a reliable evaluation. Indeed, we observe only a small difference in relative performance between evaluation scenarios with shallower and deeper pools.

We have implemented multiple retrieval methods and two main similarity models (i.e., similarity functions). In addition, we use two classic IR similarity models as baselines. Retrieval methods, similarity models, and their interactions are summarized in Figure 3.1. All similarity models are computed using *lemmatized* text. Main similarity models include:

- An inner-product similarity BM25, which ranks document using BM25 similarity [254] (see §§3.1.1.2 and 3.1.1.1 for more details);
- The linear combination of BM25 and IBM Model 1 scores, i.e., BM25+Model 1 (see §3.1.1.2 for more details).

Additional IR models include:

- Sequential Dependency Model (SDM) [204];

[9]The grade is the maximum grade using in the evaluation program `gdeval.pl`, which is commonly used in TREC evaluations. The maximum grade is relatively unimportant in this context, because both metrics computed by `gdeval.pl` are normalized metrics.

- The pseudo relevance feedback model RM3 (PRF/RM3), where expansion comes from answers retrieved in response to the query [147].

Parameters of SDM and RM3 are described in § 3.3.2.

In our preliminary experiments we also evaluated dense document representations that are compared using the cosine similarity. These representations can be retrieved quite efficiently, but the underlying similarity model is weak [45]. In this work, we have carried additional experiments (in the re-ranking mode), to ensure that our previous finding still applies to the new data sets (see § 3.2.3 for details). Because the cosine similarity between dense representations is not a competitive baseline, in this thesis this model is not included into main experiments.

The output of our retrieval pipeline is a list of $M$ scored answers. We use $M = 100$. Further increase in the number of top-$M$ entries returned by a retrieval module leads only to marginal improvement in NDCG@20 and ERR@20.

There are two approaches to generating candidates: term-based retrieval (via classic term-based inverted indices) and $k$-NN search. In the case of $k$-NN search, we employ the Non-Metric Space Library (NMSLIB), which is an extendible framework for $k$-NN search in generic spaces [42, 219].[10] Specifically, we use two indexing methods and the brute-force search. The indexing methods are: the Neighborhood APProximation index (NAPP) [284] and the Small-World graph (SW-graph) [196], which we introduced and benchmarked in § 2.2. The original variants of these methods (as proposed by their authors) treat data points as unstructured objects, together with a black-box distance function. In that, the indexing and searching process exploit only values of mutual object distances.

NMSLIB can be extended by implementing new black-box "distance" functions. In particular, we add an implementation for the similarity functions BM25 and BM25+Model1 (see §3.1.1). None of these similarity functions is a metric distance. In fact, both similarities lack symmetry and are not even premetrics (see Table 2.2). For historical reasons, SW-graph is used only with BM25+Model1. NAPP and brute-force search are applied to both BM25 and BM25+Model1. Recall that the brute-force search, which is an exact search procedure, is applicable to any similarity model. Yet, brute-force search is too slow to be practical. SW-graph and NAPP are approximate search methods: They can be a great deal faster than brute-force search at the expense of missing some true nearest neighbors.

Intrinsic evaluations described in § 2.3 demonstrate that SW-graph works well for dense vectorial data where it outstrips NAPP and the multi-probe LSH due to Dong et al. [96], often by an order of magnitude (see Panels 2.11b, 2.11a, 2.11b, 2.11g, 2.11h). Additionally, in a public evaluation[11], SW-graph outperformed two efficient popular libraries: FLANN [215] and Annoy[12]. In this evaluation SW-graph was also mostly faster than a novel LSH algorithm [5]. SW-graph also outperformed NAPP in the case of TF×IDF data, which is compared using the cosine similarity (see Panel 2.11i).

In § 2.3, results for the pivoting method NAPP were a mixed bag, because it was evaluated as a purely black-box distance method. In this setting, the selection of pivots can be sub-optimal, because only data set points can be pivots. However, we can do much better by generating pivots

[10] `https://github.com/searchivarius/nmslib`

[11] `https://github.com/erikbern/ann-benchmarks`, May 2016

[12] `https://github.com/spotify/annoy`

(see § 3.1.2.2 for details). This requires the access to the underlying structure of the space. Hence, the resulting search algorithm is a hybrid rather than purely black-box search method. Likewise, by exploiting the structure of the space, we can modify the indexing algorithm of SW-graph. In addition to improving the search methods, it is often possible to greatly speed up distance computation using various algorithmic tricks. The modified versions of SW-graph and NAPP used in this chapter as well as algorithmic tricks to speed up distance computation are discussed in § 3.1.2.

For term-based retrieval and effectiveness comparison, we use two off-the-shelf search engines: Apache Lucene[13] and Galago [80][14]. Due to its efficiency and ease of use Lucene is employed as the main engine for term-based retrieval, while Galago is employed only to provide SDM and PRF/RM3 baselines. note, however, that Lucene is written in Java. It uses posting list compression and memory mapping instead of fully loading indices into memory. In contrast, our implementations of $k$-NN search algorithms are written in C++. They load indices into memory and do not use compression. Thus, for an apple-to-apple efficiency comparison, we additionally use our own C++ re-implementation of a document-at-a-time (DAAT) posting processing algorithm employed in Lucene. An early reference of DAAT can be found in the paper by Turtle and Flood [292]. However, it is not efficient without a binary heap. The actual implementation used in Lucene seems to match Figure 5.3 of the book by Büttcher et al. [55]. To reiterate, our implementation, akin to Lucene and Büttcher et al. [55] does use the binary heap.

For Lucene, Galago, and C++ implementation of DAAT, we use BM25 as the scoring function with parameters $k_1 = 1.2$ and $b = 0.75$. Galago and our C++ DAAT BM25 have nearly equal accuracy, but Lucene BM25 is inferior (see p. 3.1.1.1). As a reminder, we do not directly use Lucene output and always re-rank it.

Note that our DAAT C++ implementation does not use compression and loads all posting lists into memory. This allows us to further speed up query processing by storing precomputed values of normalized $\text{TF}_i$ values (see Eq. 3.2) in posting lists. At query time, to compute a contribution of term $i$ to the overall value of BM25, we only need to multiply the precomputed normalized frequency $\text{TF}_i$ (extracted from a posting list) by $\text{IDF}_i$. In contrast, a full computation using Eq. 3.2 involves two divisions (which are quite expensive even on newest CPUs[113]) and several other operations. Note that Lucene also avoids the full computation of Eq. 3.2, although this is achieved at the expense of reduced accuracy.

Each retrieval method returns a ranked list of answers, which can be optionally re-ranked. In practice, however, we only re-rank output of Lucene, which is done for two reasons: (1) to produce documents ranked via the similarity function BM25+Model 1, (2) to compensate for the detrimental effect of approximation used in Lucene's implementation of BM25 (see § 3.1.1.1 for details). To re-rank efficiently, we use a *forward index* (also called a *forward file*). Given a document identifier, the forward index allows us to quickly retrieve the list of document terms and their in-document frequencies. Following a recent trend in high throughput in-memory database systems [156], we load forward indices into memory. Consequently, the overall re-ranking time is negligibly small. Likewise, NMSLIB reads contents of the forward index into memory and builds an *additional in-memory* index.

[13] `http://lucene.apache.org`
[14] `http://www.lemurproject.org/galago.php`

### 3.1.1 Similarity Models

#### 3.1.1.1 BM25

We use a variant of BM25 scores [254], which are computed as the sum of global term weights multiplied by respective *normalized* term frequencies. The sum includes only terms appearing in both the query and the answer. The global weight of term $t$ is the inverse document frequency (IDF) value computed using the following formula:

$$\text{IDF}(t) = \ln\left(1 + (N_D - \text{DF}(t) + 0.5)/(d + 0.5)\right), \tag{3.1}$$

where $N_D$ is the number of documents and $\text{DF}(t)$ is a *document frequency*, i.e., the number of documents containing the term $t$. This definition of the global term weight is adopted from Lucene. Note that it slightly differs from the classical document weight proposed by Robertson.

Normalized frequencies of the term $t$ are calculated as follows:

$$\frac{\text{TF}(t) \cdot (k_1 + 1)}{\text{TF}(t) + k_1 \cdot \left(1 - b + b \cdot |D| \cdot |D|^{-1}_{\text{avg}}\right)}, \tag{3.2}$$

where $\text{TF}(t)$ is a raw *in-document* frequency of term $t$ (i.e., a number of occurrences in the document), $k_1$ and $b$ are parameters; $|D|$ is a document length in words; $|D|_{\text{avg}}$ is the average document length.

Lucene's implementation of BM25 uses a *lossy* compression for the document length, which results in reduced effectiveness (about 10% for all three collections). To compensate for this drawback, we obtain 100 top-scored documents using Lucene and re-rank them using our own implementation of BM25. Further increase in the number of re-scored candidates only marginally improves the values of NDCG@20 and ERR@20.

#### 3.1.1.2 IBM Model 1 and BM25+Model1 Similarity

Computing translation probabilities via IBM Model 1 [53] is one common way to quantify the strength of associations among question and answer terms. Logarithms of IBM Model 1 scores are used as input to a learning-to-rank algorithm, which creates a simple yet effective model BM25+Model1. BM25+Model1 is a linear two-feature model, which includes BM25 and IBM Model 1 scores. Optimal feature weights are obtained via a coordinate ascent with 10 random restarts [205]. The model is trained via RankLib 2.7[15] using NDCG@20 as a target optimization metric. To obtain training data, we retrieve $M = 15$ candidates with highest BM25 scores using Lucene. If we do not retrieve a true answer, the query is discarded. Otherwise, we add the true answer to the training pool with the label four (which means relevant), and the remaining retrieved answers with the label zero (which means non-relevant). In § 3.2.1, we demonstrate that BM25+Model1 delivers state of the art performance in a re-ranking task.

In the remainder of this subsection, we describe IBM Model 1, which is a key ingredient of the similarity function BM25+Model1. Let $T(q|a)$ denote a probability that a question term $q$ is a

[15] https://sourceforge.net/p/lemur/wiki/RankLib/

translation of an answer term $a$. Then, a probability that a question $Q$ is a translation of an answer $A$ is equal to:

$$P(Q|A) = \prod_{q \in Q} P(q|A)$$
$$P(q|A) = (1-\lambda)\left[\sum_{a \in A} T(q|a)P(a|A)\right] + \lambda P(q|C) \tag{3.3}$$

$T(q|a)$ is a translation probability learned by the GIZA++ toolkit [233] via the EM algorithm; $P(a|A)$ is a probability that a term $a$ is generated by the answer $A$; $P(q|C)$ is a probability that a term $q$ is generated by the entire collection $C$; $\lambda$ is a smoothing parameter. $P(a|A)$ and $P(q|C)$ are computed using the maximum likelihood estimator. For an out-of-vocabulary term $q$, $P(q|C)$ is set to a small number ($10^{-9}$).

We take several measures to maximize the effectiveness of IBM Model 1. First, we compute translation probabilities on a symmetrized corpus as proposed by Jeon et al. [151]. Formally, for every pair of documents $(A, Q)$ in the parallel corpus, we expand the corpus by adding entry $(Q, A)$.

Second, unlike previous work, which seems to use *complete* translation tables, we discard all translation probabilities $T(q|a)$ below an empirically found threshold of $2.5 \cdot 10^{-3}$. The rationale is that small probabilities are likely to be the result of model overfitting. Pruning of the translation table improves both efficiency and effectiveness. It also reduces memory use.

Third, following prior proposals [151, 280], we set $T(w|w)$, a self-translation probability, to an empirically found positive value and rescale probabilities $T(w'|w)$ so that $\sum_{w'} T(w'|w) = 1$.

Fourth, we make an ad hoc decision to use as many QA pairs as possible to train IBM Model1. A positive impact of this decision has been confirmed by a post hoc assessment (see § 3.2.1 for experimental results).

Fifth, for *WikiSQuAD* we create a parallel corpus by using only sentences *containing answers* rather then complete answer-bearing passages. This modification was implemented after finding out that training with complete answer-bearing passages results in a poor model, which beats BM25 by less than 5% (as verified on a development set). One reason for this poor performance is that there are big differences in lengths between questions and answers. Thus we hit a hard-coded limit of GIZA++ toolkit for the difference in the length of aligned sentences. [16] As a result, GIZA++ simply discards all sentence term beyond a thresholds. Increasing the threshold is possible, but it greatly increases training time. Another likely reason for poor performance is the fact that—unlike sentences containing answers—long answer-bearing passages tend to contain a number of terms poorly related to question terms as well as terms that are completely unrelated.

Finally, we tune parameters on a development set (*dev1* or *dev2*). Rather than evaluating individual performance of IBM Model 1, we aim to maximize performance of the model that linearly combines BM25 and IBM Model 1 scores.

#### 3.1.1.3 Pseudo Relevance Feedback (PRF/RM3)

A pseudo relevance feedback (PRF) is a two-stage retrieval method where top `fbDocs` documents retrieved at the first stage are used to extract `fbTerm` top terms (e.g., terms with highest TF×IDF

[16] Alignment tools are designed primarily for machine translation purposes, where there is only a limited discrepancy in the number of words between the source and the target sentence.

| Feature | Explanation | Galago expression | Mathematical expression |
|---|---|---|---|
| $f_T(t, D)$ | single term $t$ | $t$ | $\log\left[(1-\alpha)\frac{\text{TF}(t)}{\lvert D\rvert}+\alpha\frac{\text{CF}(t)}{N_C}\right]$ |
| $f_O(t_1, t_2)$ | ordered terms $t_1$ and $t_2$ (within a window $N$) | $\#\text{od}N(t_1t_2)$ | $\log\left[(1-\alpha)\frac{\text{TF}(\#\text{od}N(t_1t_2))}{\lvert D\rvert}+\alpha\frac{\text{CF}(\#\text{od}N(t_1t_2))}{N_C}\right]$ |
| $f_U(t_1, t_2)$ | unordered terms $t_1$ and $t_2$ (within a window $N$) | $\#\text{uw}N(t_1t_2)$ | $\log\left[(1-\alpha)\frac{\text{TF}(\#\text{uw}N(t_1t_2))}{\lvert D\rvert}+\alpha\frac{\text{CF}(\#\text{uw}N(t_1t_2))}{N_C}\right]$ |

Table 3.7: Summary of feature functions used in SDM (TF$(t)$ is an in-document term frequency, CF$(t)$ is a collection frequency of term $t$)

values). The weighted combination of these terms is used to expand the original query. The second retrieval stage involves execution of this expanded query.

There exist numerous variants of PRF, which differ in approaches to selecting top terms, weighting them, as well as in ways of mixing PRF scores with documents scores produced by the original query. However, some of the best results are obtained by the model RM3 [147, 193], which is shown to be robust and (mostly) more accurate than other methods [193, 325].

RM3 linearly interpolates scores produced by the original query (weight `fbOrigWeight`) with the scores produced by RM1 [175]. In turn, RM1 first creates a pseudo feedback collection that consists of top `fbDocs` documents obtained in the first stage. Each document in this collection is weighted using the query likelihood (computed at the first stage). Then, the likelihood of each query term is computed as the sum of weighted likelihoods over all documents in the pseudo feedback collection (see the paper of Lv and Zhai [193] and references therein for more details).

Note that we use a Galago's [80][17] implementation of PRF/RM3.

#### 3.1.1.4 Sequential Dependency Model (SDM)

Metzler and Croft [204] proposed a generic framework for incorporating multi-term dependency into an established language modeling framework for IR [132, 246]. To this end, a multi-term configuration of query terms is treated as yet another query term. After obtaining a collection and in-document frequencies of such configuration, they compute a likelihood of encountering such a configuration in a document. These multi-term likelihoods are subsequently interpolated with likelihoods for individual terms.

The number of term combination is exponential in the query size and it is usually too expensive to incorporate them all into the ranking formula. Fortunately, as shown by Metzler and Croft a simpler *Sequential Dependency Model* (SDM) that takes into account only adjacent query words is quite effective [204]. In SDM, the score of a document is computed as a linear combination of terms of three types:

[17] `http://www.lemurproject.org/galago.php`

**Exposition 3.1: An alternative approach to fast retrieval of query-document term similarities**

An alternative but more involved approach to fast computation of all query-document term similarities relies on storing term ids in the form of sorted arrays. One array would contain document term entries sorted by their identifiers. Another array would contain sorted entries resulted from the pseudo-expansion of the query, i.e., $\{a \mid \exists q\ T(q|a) > 0\}$. Then, the key operation would consists in efficient computation of the intersection between these sorted arrays.

One good way of doing so is a galloping search [31], which iterates over a short list and searches entries in the longer list using a variant of the binary search. This approach should work well in our case, because the number of terms in a typically expanded query is much larger than the number of document term ids.[a] We think that this method can work faster for several reasons:

1. Representing the inverted index in the form of plain sorted array results in a smaller memory footprint compared to hashing.
2. Unlike hashing, all elements in the array are accessed sequentially.
3. Galloping search permits effective vectorization, i.e., it can be computed more efficiently on modern CPU with the support for SIMD operations [179].

Overall, we expect better cache utilization due to a better access pattern and a smaller memory footprint.

[a] The number of expanded query entries is often in the dozens of thousands.

- $f_T(t, D)$, which is a likelihood of a single term;
- $f_O(t_1, t_2)$, which is a likelihood of an *ordered* configuration of terms $t_1$ and $t_2$ within a window of a given size. Because the configuration is ordered $t_1$ must precede $t_2$.
- $f_U(t_1, t_2)$: a likelihood of an *unordered* configuration of terms $t_1$ and $t_2$.

We use a Galago's [80][18] implementation of SDM, where it is possible to specify a weight for terms of each three types.

Huston and Croft analyzed the literature and concluded that with respect to more complex dependency models, "none of these techniques have been shown to be consistently better than the simpler adjacent pairs method for identifying good candidate term dependencies." [141] This observation is also supported by their own evaluation, where "...many-term dependency models do not consistently outperform bi-term models." [141] For the largest collections Gov2 and ClueWeb (Category B) used in this evaluation, whenever other methods have higher values of NDCG@20 and ERR@20, the differences is within 4% [141].

[18] `http://www.lemurproject.org/galago.php`

### 3.1.2 Making $k$-NN Search Fast

#### 3.1.2.1 Optimizing Computation of BM25+Model1

We focus on optimizing computation of BM25+Model1, because it represents a major computational bottleneck. A straightforward but slow approach to compute IBM Model 1 scores involves storing $T(q|a)$ in the form of a sparse hash table using pairs $(q, a)$ as keys. Then, computation of Eq. 3.3 entails one hash table lookup for each combination of a question and an answer term. For large translation tables, the performance of the hash table becomes memory-bound, because each hash table lookup requires at least one costly random memory access. In a general case, this seems to be an insurmountable issue. However, in our setting, there are regularities in accessing the translation table, which permit computational shortcuts.

First, we can improve speed in the case when different documents are compared against the same but possibly *non-repeating* query. Before comparison starts, we create an inverted index, which permits retrieving query-specific entries $T(q|a)$ using the identifier of answer term $a$ as a key. This somewhat similar to the query expansion, because we find all $a$ for which $T(q|a) > 0$ for at least one query term $q$. Identifiers are indexed using an efficient hash table (the class *dense_hash_map* from the package *sparsehash*)[19]. The number of expansion entries depends on the specific translation table. However, it is generally large (dozens of thousands of entries).

After the index is created, we need only one lookup per answer term (an alternative approach to fast retrieval of query-document similarities is described in Exposition 3.1). Not taking preprocessing costs into account, this reduces processing times by the factor of $|q|$, where $|q|$ is the number of query terms (additional savings can be achieved if such an index is small enough to fit into L3 or L2 cache). Building such an inverted index is computationally expensive (about $15\,\mathrm{ms}$ for each *Comprehensive* query and $90\,\mathrm{ms}$ for each *Stack Overflow* query). Yet, in most cases, the cost is amortized over multiple comparisons between the query and data points.

Second, we can improve speed when data or query points are compared against the *same* subset of data points, e.g., pivots. This modification is an extension of David Novak's idea to speed up such computation for the case of the inner product between sparse vectors. Imagine that we need to compute an inner-product similarity between a query vector and every pivot $\pi_i$ each of which are represented by a sparse vector. A straightforward approach involves iterating over all pivots. For each pivot, we iterate over a set of non-zero query dimensions. For each non-zero dimension $t$, we multiple the values of the query's dimension $t$ by the value of the pivot's dimension $t$. The product value is added to the result variable.

Because sparse vectors are stored in some compact form, retrieving the value of dimension $t$ is costly. It can be sped up if we somehow *batch* these lookup operations. First, we would zero-initialize accumulator variables, one for each pivot $\pi_i$. Then, we would iterate over all non-zero dimension of the query. At each iteration step, we would identify all pivots for which dimension $t$ is non-zero. We would then multiply each of these values by the value of the query's dimension $t$ and add the result to the respective accumulator value. When the computation is finished, accumulator $i$ contains a dot-product between the query and $\pi_i$. To support efficient batch, we create an inverted index of all pivots non-zero dimensions using dimension indices as

[19] The code, originally written by Craig Silverstein, is now hosted at `https://github.com/sparsehash/sparsehash`

keys (as proposed by David Novak).

To extend this approach to BM25+Model 1, we create an inverted index for all non-zero values $T(q|a)P(a|\pi_i)$ using $q$ as a key. When the distances are computed in a batch mode, for each query term $q$ we retrieve values $T(q|a)P(a|\pi_i)$. Each value $T(q|a)P(a|\pi_i)$ is added to the value of accumulator $i$. When this computation is finished, we need to interpolate accumulator values with collection probabilities (see Eq. 3.3). A potentially useful additional optimization is to merge all inverted index entries corresponding to the same $q$ and $\pi_i$ by simply summing up their values. However, it is not implemented yet.

Although the description of these tricks is specific to BM25+Model 1, we believe that a similar approach can be used in other scenarios as well. In particular, a similar efficiency issue arises when word embeddings are used in the framework of translation language modeling [119, 125, 331]. There, we need to compute the cosine similarity (or an inner product) between each pair of a query and document term vector, which represents a substantial cost. If we precompute these cosine similarities for all pairs of sufficiently similar terms, this costs can be reduced in the same fashion as in the case BM25+Model 1. Pruning pairs with low similarity does introduce an approximation, but it is likely to have a negligible effect on performance.

#### 3.1.2.2 Improving Methods for $k$-NN Search

Recall that NAPP is a *pivoting* method that arranges points based on their distances to pivots [284]. This is a filtering method: All candidate points should share at least `numPivotSearch` closest pivots with the query point. The search algorithm employs an inverted index. Unlike term-based indices, which memorize which terms appear in which document, for each pivot the index keeps references to data points for which this pivot is among `numPivotIndex` closest pivots. During indexing, we have to compute the distances between a data point and every pivot. As described in § 3.1.2.1, this operations is sped up by batching.

Tellez et al. [284] use pivots randomly sampled from the data set, but David Novak has discovered that for *sparse* data such as TF×IDF vectors substantially shorter retrieval times—sometimes by orders of magnitude—can be obtained by using a special pivot generation algorithm (this finding is published in our joint paper [45]). Specifically, pivots are generated as pseudo-documents containing $K$ equal-value entries sampled from the set of $M$ most frequent words *without* replacement (in our setup $K = 1000$ and $M = 50000$). This surprisingly simple modification makes NAPP a competitive search method, which can sometimes outperform SW-graph. This modification was originally proposed for the cosine similarity, but we have discovered it works for BM25+Model 1 too.

SW-graph is a variant of a proximity graph, i.e., it is a data structure, where data points are nodes and sufficiently close nodes, i.e., *neighbors*, are connected by edges. Compared to the version we used in § 2.3, we implemented two important modifications. First, recall that in SW-graph, both indexing and retrieval employ the same search algorithm, which consists in a series of greedy sub-searches. A sub-search starts at some, e.g., random node and proceeds to expanding the set of traversed nodes in a best-first fashion by following neighboring links. The depth of the search is controlled by the parameter `efSearch`. As explained in § 2.2.2.3, in § 2.3, we set `efSearch` equal to the value of the parameter `NN`, which is the (maximum) number of neighbors for a newly added data point (note that as we add more data points, the

**Exposition 3.2: Tight clusters do not always exist**

To illustrate that tight clusters do not necessarily exist, we experiment with the Wiki-sparse data set introduced in § 2.3.1.1. It contains about four million sparse TF×IDF vectors generated from Wikipedia. The distance function is the angular distance, which is a true metric distance. There is a monotonic transformation that turns the cosine distance into the angular distance and vice versa. Hence, $k$-NN search with the (metric) angular distance produces the same neighbors as $k$-NN search with the (non-metric) cosine distance.

We note that the distance from a cluster center to its $k$-th nearest neighbor is a lower bound for the radius of the cluster with $k+1$ members. We then estimate the distance to a $k$-th nearest neighbor via sampling. Additionally, we estimate a distance to a randomly selected data point (which is nearly always is *not* a nearest neighbor). More specifically, we do the following:

- We randomly and uniformly select 5K data points and find their $k$-nearest neighbors using brute-force search (for $k \leq 100$);
- We randomly and uniformly select 10M pairs of data points and compute 10M pairwise distances.

Because the space is highly dimensional, the histogram of distances is quite concentrated. In more than 99% of all cases the distance between two randomly selected points is between 1.49 and 1.63. What is worse, in 50% of all cases the distance to the first nearest neighbor is more than 0.9. This means that it is hard to distinguish between near and far points because they are located at similar distances from the query.

This observation has two important consequences. First, when a query is compared to a cluster center, in half of all cases, the distance is larger than 0.9. Note that this is a data-dependent but strict lower bound, which applies to any clustering algorithms, including k-means clustering [190].[a] Second, in 50% of all cases the radius of the cluster is larger than 0.9.

It is not unreasonable to model the radius of the cluster and the distance from the query to the cluster center as random variables. It is also quite reasonable to assume that these variables are nearly independent. Then, in 25% of all cases, the distance from the query to the cluster center *and* the cluster radius would be *both* in excess of 0.9. It can be seen that the location of the cluster center is a very poor predictor of locations of cluster data points. For example, if the distance to the cluster center is 1, then, according to the triangle inequality, the range of possible distances between the query and a cluster data point can be from 0.1 to 1.9. Yet, this is likely to comprise nearly all data set points (as noted above in 99% of all cases the distance does not exceed 1.63).

[a] In fact, this lower bound is quite optimistic for two reasons. First, to make clustering efficient, clusters need to be large, e.g., they should contain dozens or hundreds of data points. But even for clusters of size 10, the radius of the cluster would exceed 0.95 (using 10-NN distance as the lower bound). Second, real clustering algorithms are unlikely to produce tight clusters that consist solely of its center's nearest neighbors.

number of neighbors for many data points can become (and often becomes) larger than `NN`). The rationale behind using multiple (e.g., random) restarts is to increase recall. However, in this

section, following a suggestion in [197], we achieve various recall-efficiency trade-offs by varying the value of the parameter `efSearch` and use a *single* sub-search.

Recall that in the case of NAPP, we compute BM25+Model 1 distance only to a set of fixed pivots, which permits an efficient batching and, consequently, BM25+Model1 can be used to create an index. However, batching is hardly possible in SW-graph, because the set of data points with which the query is compared is not static and depends on the query itself. Because we do not know how to efficiently construct a graph using BM25+Model 1, we resort to building the proximity graph using BM25 similarity. Technically, we construct the proximity graph using BM25 with a natural symmetrization and sharing of `IDF` values. The respective similarity computation is described by Eq (2.22) in § 2.3.2.4.

The resulting graph is likely to be sub-optimal, because the indexing algorithm does not bridge the vocabulary gap. Yet, as we show in § 3.3, the resulting method for $k$-NN search can outperform NAPP, despite where the indexing procedure does take into account subtle word associations mined from parallel monolingual corpora.

In addition to improving NAPP and SW-graph, we experiment with several distance-based clustering algorithms, which include a canopy-like clustering due to Chávez and Navarro [64], the iterative variant of canopy-like clustering and iterative variant of CLARANS [226]. In the iterative variant, we cluster using a few centers and recursively apply the same procedure to data points that are not placed sufficiently close to their respective cluster centers. At search time, the clusters are visited in the order of increasing distance from the query to a cluster center. The search process stops after vising a certain number of clusters (a method parameter). There are a number of papers that exploited this search algorithm in both metric and non-metric setting, which differ mostly in the way of building clusters (see, e.g., [121, 181]). Unfortunately, none of the methods work on our data and we further investigated if our data set had good clusters. As we illustrate in Exposition 3.2, in high-dimensional data sets of the medium size—containing a few million entries—it is likely tight clusters do not exist, i.e., the data is not sufficiently dense.

It is possible that scaling data sets to billions of data points substantially improves the density (because the probability of *not* having a nearby point should decrease quite substantially). In this case, the clustering algorithms should become more efficient with respect to the brute-force search. However, it is very likely that for denser data the relative performance of other methods (including SW-graph and NAPP) would improve as well.

## 3.2 Exploratory Experiments

As noted in the beginning of § 3.1, to demonstrate the usefulness of $k$-NN search in text retrieval, we need to design a similarity function that outstrips the baseline method BM25 by a good margin. To construct such a similarity, we have carried out a large number of exploratory experiments, which are described in this section.

First, we assess performance of our BM25+Model 1 implementation by comparing it with a previously published result (see § 3.2.1). In doing so, we (1) verify correctness of our implementation and (2) ensure that BM25+Model 1 outperforms BM25 by a good margin. We then demonstrate that using a lot of training data is essential to achieving the latter objective.

In one scenario, we use only a lexical Model 1, which estimates the strength of associations

| Prior art [280] | | | | |
|---|---|---|---|---|
| | $M = 15$ | $M = 25$ | $M = 50$ | $M = 100$ |
| Recall | 0.290 | 0.328 | 0.381 | 0.434 |
| P@1 | 0.499 | 0.445 | 0.385 | 0.337 |
| MRR | 0.642 | 0.582 | 0.512 | 0.453 |
| This work | | | | |
| | $M = 10$ | $M = 17$ | $M = 36$ | $M = 72$ |
| Recall | 0.285-0.302 | 0.322-0.341 | 0.376-0.395 | 0.428-0.448 |
| P@1 | 0.553-0.588 | 0.493-0.527 | 0.426-0.458 | 0.377-0.406 |
| MRR | 0.695-0.720 | 0.632-0.658 | 0.557-0.582 | 0.498-0.522 |
| Gain in P@1 | 11-18% | 11-18% | 11-19% | 12-20% |
| Gain in MRR | 8-12% | 9-13% | 9-14% | 10-15% |

Table 3.8: Comparison of BM25+Model 1 against prior art on *Manner*. Accuracy is computed for several result set sizes $M$ using the methodology of Surdeanu et al. [280]. Each column corresponds to a different subset of queries. Values obtained in our experiment are represented by respective 99.9% confidence intervals.

only among terms. In § 3.2.2, we explore scenarios where Model 1 is computed for a combination of lexical items and linguistic annotations such as semantic role labels and named entities. Finally, in § 3.2.3, we present our evaluation results for dense document representations.

### 3.2.1 Evaluating Lexical IBM Model 1

To demonstrate that BM25+Model 1 is a correct implementation that delivers state of the art performance, we compare our result on *Manner* against a previously published result [280].[20] We mimic the setup of Surdeanu et al. [280] in several ways: First, we split *Manner* in the same proportions into the training, development and testing sets. The exact split used by Surdeanu et al. [280] is unfortunately not known. To ensure that the differences are substantial and significant, we compute 99.9% confidence intervals for all measured performance metrics. Second, we discard all but the best answers.

Third, we measure P@1 (instead of NDCG@20) and the Mean Reciprocal Rank (MRR) at various answer recall levels by varying the result sizes $M$ (during training we also maximize P@1 instead of NDCG@20 and ERR@20). The mean reciprocal rank is the inverse rank of the first relevant answer averaged among all queries. We select values $M$ in such a way that respective recall values are as close as possible to answer recall values obtained by Surdeanu et al. for $M = 15$, $M = 25$, $M = 50$, and $M = 100$ (in fact, all our recall values are slightly higher). Note our values of $M$ are different from those used by Surdeanu et al.) [280], because we use a different search engine (Lucene instead of Terrier [194]), a different lemmatization/tokenization software, and a possibly different stoplist.

[20] We have carried out a literature review, but we have not found papers improving upon this result.

| *Comprehensive* | | | | | *Stack Overflow* | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Number of QA pairs | NDCG@20 | ERR@20 | Gain in NDCG | Gain in ERR | Number of QA pairs | NDCG@20 | ERR@20 | Gain in NDCG | Gain in ERR |
| **BM25** | | | | | | | | | |
| N/A | 0.0793 | 0.1308 | 0.0% | 0.0% | N/A | 0.0844 | 0.0879 | 0.0% | 0.0% |
| **BM25+Model1** | | | | | | | | | |
| 20752K | 0.0940 | 0.1571 | 18.6% | 20.1% | 11014K | 0.1063 | 0.1136 | 26.0% | 29.2% |
| 10376K | 0.0933 | 0.1562 | 17.7% | 19.4% | 5507K | 0.1055 | 0.1125 | 25.0% | 28.0% |
| 5188K | 0.0920 | 0.1548 | 16.1% | 18.4% | 2754K | 0.1044 | 0.1115 | 23.7% | 26.7% |
| 2594K | 0.0915 | 0.1536 | 15.4% | 17.4% | 1377K | 0.1034 | 0.1109 | 22.6% | 26.1% |
| 1297K | 0.0895 | 0.1503 | 12.9% | 14.9% | 688K | 0.1025 | 0.1089 | 21.5% | 23.9% |
| 648K | 0.0882 | 0.1482 | 11.3% | 13.3% | 344K | 0.1006 | 0.1073 | 19.3% | 22.0% |
| 324K | 0.0870 | 0.1462 | 9.7% | 11.8% | 172K | 0.0984 | 0.1048 | 16.6% | 19.2% |
| 162K | 0.0859 | 0.1437 | 8.4% | 9.9% | 86K | 0.0974 | 0.1033 | 15.4% | 17.5% |
| 81K | 0.0836 | 0.1406 | 5.5% | 7.5% | 43K | 0.0943 | 0.0999 | 11.7% | 13.6% |

Table 3.9: Effectiveness of BM25+Model 1 for training data of varying size (in a re-ranking experiment on *dev1* for $M = 100$)

Importantly, we use only questions for which a relevant answer is found, but not necessarily ranked number one. Filtering out queries that do not retrieve a relevant answer is done to reproduce the setup of Surdeanu et al. [280]. As a result of this filtering, different $M$ correspond to different subsets of documents. Therefore, it is not surprising that the values of P@1 decrease as $M$ increases. Indeed, for $M = 1$, we would include only queries where the relevant answer is ranked highest. Hence, P@1 will have a perfect value of 1. For $M = 2$, however, we may include queries where the relevant answer is ranked second. For these queries, P@1 is zero. Consequently, inclusion of these queries decreases the overall P@1. By further increasing $M$ we increase the fraction of queries with zero P@1 and, thus, decrease the overall value of P@1. The same reasoning explains why MRR decreases as $M$ grows.

According to results Table 3.8, our method surpasses the previously published result by at least 11% in P@1, and by at least 10% in MRR despite using only two features (we have found no other papers that improve over the original result of Surdeanu et al. [280]). This may be explained by two factors. First, we use a 50× larger corpus to train IBM Model 1. Second, Terrier [194] seems to deliver a less accurate ranking algorithm compared to our Lucene-based implementation. In particular, Lucene achieves a higher recall using a smaller $M$ (see Table 3.8). This conjecture is also supported by a recent evaluation [187]. It is not clear to us why this is the case. Note that Surdeanu et al. also use a different learning algorithm to combine BM25 and BM25+Model 1 scores. However, we do not think this makes a difference, because both learning algorithms combine features using a linear function and train on equally large data sets containing about 30K queries.

Now we demonstrate that BM25+Model 1 outperforms BM25 by a good margin, yet, this requires quite a bit of data. To assess the effect of the training data availability, we carry out two experiments with three collections (on a *development* set *dev1*). In the first experiment,

| Parallel corpus | NDCG@20 | ERR@20 | Gain in NDCG | Gain in ERR |
|---|---|---|---|---|
| **BM25** | | | | |
| N/A | 0.3196 | 0.2542 | 0.0% | 0.0% |
| **BM25+Model1** | | | | |
| *WikiSQuAD* | 0.3697 | 0.3035 | 15.7% | 19.4% |
| *Comprehensive* | 0.3783 | 0.3115 | 18.3% | 22.6% |
| *WikiSQuAD +Comprehensive* | 0.3968 | 0.3318 | 24.1% | 30.5% |

Table 3.10: Effectiveness of BM25+Model1 for two models and their combination (in a re-ranking experiment on *WikiSQuAD dev1* for $M = 100$)

we use collections *Comprehensive* and *Stack Overflow*. In this case, we evaluate effectiveness of BM25+Model 1 for training sets of exponentially decreasing sizes. Note that this is a re-ranking experiment using $M = 100$ candidates. First, we train a model using a complete parallel corpus, then we experiment with a half of a collection, and so on so forth. The complete parallel corpora contain about 21M QA pairs (for *Comprehensive*) and 11M QA pairs (for *Stack Overflow*). The smallest samples of parallel data contain only 81K (for *Comprehensive*) and 43K (for *Stack Overflow*) QA pairs. According to results in Table 3.9, there is quite a performance gap between training using the smallest and the largest parallel corpora. For example, in the case of *Comprehensive*the improvement over BM25 grows from 5.5% to 18.6% as measured in NDCG@20. Clearly, there are diminishing returns with respect to increasing the data size. Yet, by using millions rather than hundreds of thousands QA pairs, one can obtain substantial gains in accuracy (especially for *Comprehensive*).

In the second experiment (which also uses a development set *dev1*), we measure relative performance of BM25+Model1 on *WikiSQuAD*. The parallel corpus extracted from *WikiSQuAD* alone contains only about 77K QA pairs. According to results in Table 3.10, the relative improvement of BM25+Model1 over BM25 is only 15.7% in NDCG@20 and 19.4% in ERR@20. Quite surprisingly, bigger gains can be achieved by simply re-using the model trained on a parallel corpus for *Comprehensive*. To further boost accuracy on *WikiSQuAD*, we decided to combine both models using the simple linear model with three features: a BM25 score and two Model 1 scores (for two models trained on different parallel corpora). This combination improves over BM25 by 24.1% in NDCG@20 and by 30.5% in ERR@20.

### 3.2.2 Evaluating BM25+Model1 for Linguistically-Motivated Features

We explore two approaches to construct a linguistically-motivated similarity function. In the first approach, we generate linguistically-motivated textual representations of questions and answers as proposed by Surdeanu et al. [280]. We segment text into sentences, remove punctuation, extract word tokens, obtain token lemmas, and obtain Penn Tree bank part-of-speech (POS) tags [201] using Stanford CoreNLP [200]. All words and lemmas are lowercased. We further parse sentences

| Description | Mnemonic code | Sample |
|---|---|---|
| Original text | N/A | How does the President nominate Supreme court justices? |
| Lemmas | lemmas | president nominate supreme court justice |
| Original non-stop words | words | president nominate supreme court justices |
| Bigrams of lemmas | bigrams | president_nominate nominate_supreme supreme_court court_justice |
| Semantic Roles (unlabeled) | srl | nominate_president nominate_justice |
| Semantic Roles (labeled) | srl_lab | nominate_A0_president nominate_A1_justice |
| Dependencies (unlabeled) | dep | nominate_president court_supreme justice_court nominate_justice |
| WordNet super-senses | wnss | verb.social noun.person verb.communication noun.group noun.attribute |

Table 3.11: Types of textual representations for *Manner*

by constructing dependency trees and obtaining PropBank style Semantic Role Labels (SRLs) [160] using the package ClearNLP [73]. [21] The annotating pipeline is implemented in Java. It employs the framework Apache UIMA as well as the collection of UIMA components from DKPro Core [102].[22]

We also extract WordNet super-senses [110]. Surdeanu et al. employed a super-sense tagger authored by Ciaramita and Altun and written in C++.[23] We instead took a simpler approach by selecting the most frequent super-sense corresponding to a word's POS tag. Despite simplicity it is quite a strong and high-accuracy baseline [224], which was we also confirmed by comparing two approaches on a small set of sentences.

Parsing the text allows us to create multiple text representations. Sample text representation for a question "How does the President nominate Supreme court justices?" are given in Table 3.11. There are three purely lexical text representations: words, lemmas, and lemmas' bigrams. In all three cases, we remove words and lemmas appearing on the stop word list. There are two representations that employ ProbBank style semantic role labels. In the first (unlabeled) representation, we simply concatenate verbs with words from the arguments of the verbal predicate. In the second (labeled) representation, we additionally insert an argument label. In the case of syntactic dependencies we concatenate lemmas adjacent in the dependency tree without using the labels of dependency relations.

For each text representation, there is a parallel corpus that we use to train Model 1. Similar to previously described experiments on *Manner*, the parallel corpora are created from a subset of *Comprehensive* that (1) does not contain questions or answers from *Manner* and (2) employs only best answers. The main difference here is that, in addition to the lemma-based parallel corpus, we use corpora for several textual representations. We compare accuracy in a re-ranking experiment on the second development set *dev2* where $M = 72$ candidate entries are re-ranked using similarity BM25+Model1.

According to results in Table 3.12, the best performing text representations are the lexical

[21] We used a patched version of ClearNLP 2.0.2, which we made available on GitHub. We had to patch the semantic role labeler because it crashed with a null-pointer exception for some inputs. Patching only affected a few outcomes when the unpatched parser would crash otherwise.

[22] `https://dkpro.github.io/dkpro-core/`

[23] The parser is available on SourceForge.

| Features used to compute Model 1 | NDCG@20 | ERR@20 | Gain in NDCG@20 | Gain in ERR@20 |
|---|---|---|---|---|
| **BM25** | | | | |
| N/A | 0.2262 | 0.1806 | 0.0% | 0.0% |
| **BM25+Model1** | | | | |
| lemmas | 0.2531 | 0.2056 | 11.9% | 13.8% |
| words | 0.2557 | 0.2081 | 13.0% | 15.2% |
| bigrams | 0.2323 | 0.1863 | 2.7% | 3.2% |
| dep | 0.2305 | 0.1844 | 1.9% | 2.1% |
| srl | 0.2272 | 0.1814 | 0.4% | 0.5% |
| srl_lab | 0.2266 | 0.1808 | 0.2% | 0.1% |
| wnss | 0.2264 | 0.1807 | 0.1% | 0.1% |
| lemmas+words | 0.2562 | 0.2082 | 13.2% | 15.3% |
| lemmas+bigrams | 0.2580 | 0.2105 | 14.0% | 16.6% |
| lemmas+dep | 0.2559 | 0.2082 | 13.1% | 15.3% |
| lemmas+srl | 0.2538 | 0.2062 | 12.2% | 14.2% |
| lemmas+srl_lab | 0.2546 | 0.2071 | 12.5% | 14.7% |
| lemmas+wnss | 0.2530 | 0.2055 | 11.8% | 13.8% |

Table 3.12: Effectiveness of BM25+Model1 for lexical and linguistic features as well as for their combination (in a re-ranking experiment on *Manner dev2* for $M = 72$)

ones. Combining BM25 with Model 1 for original words, i.e., unlemmatized words, boosts performance by 13% in NDCG@20 compared to 11.9% for lemmas and 2.7% for bigrams. Translation models for linguistically-motivated text representations deliver only small performance gains with the best improvement of 1.9% for dependencies. Furthermore, combining linguistically-motivated models with the BM25+Model 1 for lemmas is always worse than combining BM25+Model 1 for lemmas with BM25+Model 1 for original words. Combining BM25+Model1 for lemmas with BM25+Model1 for unlemmatized words improves NDCG@20 by 1.3%. Combining BM25+Model1 for lemmas with BM25+Model1 for bigrams improves NDCG@20 by 2.1%. However, we have decided not to use the bigram or original words model for efficiency reasons. Bigram model is especially large and it requires quite bit a of memory during training: We had to rent a 256 GB AWS instance to do this.

The second approach to construct a linguistically-motivated similarity function relies on answer-typing commonly used in factoid QA [247]. It is therefore applied only to *WikiSQuAD*, which—as explained previously—is mostly a factoid QA collection. We extract different sets of linguistically-motivated features from questions and answers using question-processing modules of OpenEphyra [264] and YodaQA [25].

Consider, an example of the question "In which city was the first Starbucks coffee shop opened?". This question is about location, which is indicated by the word *city*, which is a focus word of this factoid question. For a more nuanced discussion of question typing, please, see the

| Description | Mnemonic code | Sample |
|---|---|---|
| | | **Question** |
| Original text | N/A | What year did Croatia pass a law regarding polytechnic education? |
| Lemmas | lemmas | croatium pass law regard polytechnic education |
| Ephyra question type | ephyra | NEyear |
| Question and focus words | qwords | what year |
| | | **Answer-bearing passage** |
| Original text | N/A | In Croatia there are many polytechnic institutes and colleges that offer a polytechnic education. The law about polytechnic education in Croatia was passed in 1997. |
| Lemmas | lemmas | croatium polytechnic institute college offer polytechnic education law polytechnic education croatium pass 1997 |
| Spacy entity categories | spacy | DATE |
| DBPedia entity categories | dbpedia | Location Country Location Country |

Table 3.13: Sample textual representations for *WikiSQuAD*

survey by Prager [247].

OpenEphyra is used to extract focus words and phrases as well as the question type. YodaQA is used only to extract a focus words. Focus words from YodaQA and OpenEphyra are merged and duplicates are removed.

OpenEphyra often generates several question types, but we select the one with the highest confidence score. The selected question type may contain several descriptors with increasing specificity. For example, OpenEphyra may indicate that an answer type is both location and city. In such a case, we select the least specific type.

From answer-bearing passages, we extract named entities of two types. First, we employ an additional Python script to generate coarse-grained entities using spaCy [135][24], which recognizes 17 general categories such as locations, names, and dates. Second, we obtain fine-grained entities using DbPedia Spotlight [81]. DbPedia Spotlight links entities in the text with entities present in DbPedia. DbPedia has an ontology with 771 categories such as *WrittenWork*, *Song*, *AcademicSubject*. This ontology is hierarchical, but we select the most specific category. An example of textual representations constructed for one QA pair is given in Table 3.13.

A manual examination of question-typing results for a small sample of questions revealed that our pipeline (powered by OpenEphyra and YodaQA) generates focus words quite accurately. There are clearly more mistakes in determining answer types. This is in part because OpenEphyra was designed to perform well in a TREC QA task, where a set of question types is quite limited. However, the hope is that these errors are systematic and respective biases may be successfully learned by an alignment model. In other words, we believe that the alignment model should be

[24]Available at `https://spacy.io`

| OpenEphyra answer types | | | |
|---|---|---|---|
| location | dbpedia:Location/0.264 | dbpedia:Country/0.079 | spacy:GPE/0.07 |
| person | dbpedia:Location/0.126 | spacy:PERSON/0.116 | spacy:DATE/0.108 |
| organization | spacy:ORG/0.220 | dbpedia:Location/0.109 | spacy:DATE/0.106 |
| song | spacy:PERSON/0.138 | dbpedia:Single/0.115 | spacy:DATE/0.107 |
| time | spacy:DATE/0.196 | dbpedia:Location/0.094 | spacy:GPE/0.085 |
| language | dbpedia:Language/0.403 | spacy:NORP/0.193 | spacy:LANGUAGE/0.087 |
| **Focus words** | | | |
| attitude | dbpedia:Island/0.197 | spacy:GPE/0.172 | dbpedia:Settlement/0.162 |
| cemetery | dbpedia:Engineer/0.188 | dbpedia:OfficeHolder/0.160 | dbpedia:CollegeCoach/0.094 |
| company | dbpedia:Company/0.597 | spacy:DATE/0.043 | spacy:MONEY/0.037 |
| language | dbpedia:Language/0.505 | spacy:NORP/0.212 | spacy:LANGUAGE/0.101 |
| mathematician | spacy:GPE/0.309 | spacy:NORP/0.181 | dbpedia:Scientist/0.143 |

Table 3.14: Sample top-3 associations mined between question features (or focus words) and answer annotations.

The first column contains Ephyra answer types or focus words. The remaining columns show top-3 associated words with respective probabilities (separated by the slash).

| Features used to compute Model 1 | NDCG@20 | ERR@20 | Gain in NDCG@20 | Gain in ERR@20 |
|---|---|---|---|---|
| **BM25** | | | | |
| N/A | 0.3196 | 0.2542 | 0.0% | 0.0% |
| **BM25+Model1** | | | | |
| ephyra+spacy | 0.3224 | 0.2564 | 0.9% | 0.9% |
| ephyra+dbpedia | 0.3223 | 0.2565 | 0.8% | 0.9% |
| ephyra+spacy+dbpedia | 0.3251 | 0.2592 | 1.7% | 2.0% |
| lemmas | 0.3683 | 0.3016 | 15.2% | 18.6% |
| lemmas+all linguistic features | 0.3710 | 0.3029 | 16.1% | 19.1% |

Table 3.15: Effectiveness of BM25+Model1 for lexical and linguistic features as well as for their combination (in a re-ranking experiment on *WikiSQuAD dev1* for $M = 100$).

Note the following: (1) we train BM25+Model1 only on *WikiSQuAD* data, (2) the results for `lemmas` are slightly different from results in Table 3.10, because in this experiment we use slightly different BM25+Model1 parameters (and the resulting BM25+Model1 model is slightly less effective).

capable of finding *useful* associations between focus words in questions and types of entities in answers even if there are question typing and entity extraction errors. In doing so, an alignment model may be able to remedy some errors made by our tagging tools.

After building BM25+Model1 for textual representation with QA-specific features, we manually examined associations for a sample of question types and focus words. Some of these

associations along with respective association probabilities are presented in Table 3.14. We can see that some of the learned associations are quite good. For example, both the focus word `language` and the OpenEphyra type `language` are strongly associated with the DbPedia and spaCy entities representing a language as well as with the spaCy entity `NORP`, which represents nationalities, religious, and political groups. Clearly, good associations are also learned for the OpenEphyra answer type `location`, which is associated strongly with DbPedia entities for Locations and Countries as well as with spaCy entiy `GPE`, which marks cities, countries, and states. However, the OpenEphyra answer type `person` is most strongly associated with DbPedia locations. This association looks suspicious. Some associations for focus words are outright weird. Consider, for example, a focus word `attitude`, which appears in questions such as "What are the anti-Catalan attitudes in Valencia and Ibiza?". It is strongly associated with location entities, but there seems to be no good reason for this. It is likely, however, that many attitude questions have a geographical context, but it is not clear how well such association may generalize, in particular, because location entities are quite common.

We further assess the effectiveness of linguistic features in a series of ablation experiments on a development set *dev1*. In this case, we build the alignment model using solely *WikiSQuAD* data. First, we carry out three experiments, where BM25+Model1 is constructed without using lexical information, i.e., we use only answer types and/or entity labels. We can see that using OpenEphyra answer types and entities results in minuscule improvements: The biggest gain is only 2%, which is quite small compared to 18.6% achieved when training BM25+Model1 purely on lexical data. Furthermore, combining lexical data, i.e., lemmas with all linguistic features (including question and focus words), results in only 0.5% gain over the lexical model.

In summary, using linguistic features did not result in substantially better alignment models.

### 3.2.3 Evaluating Cosine Similarity between Dense Document Representations

In addition to lexical Model 1 and Model 1 built with linguistically-motivated features, we evaluate effectiveness of dense document representations, which are compared via cosine similarity. We focus on dense document representations computed via averaging of document word embeddings. As explained below, we also tested `doc2vec` model [176], but it did not result in higher accuracy. Word embeddings are usually constructed in an unsupervised manner from large unstructured corpora via artificial neural networks [77, 206]. Because embeddings can capture syntactic and semantic regularities in language [207, 291], embedding-based similarity can be useful in retrieval and re-ranking tasks [115, 320]. The hope here is that comparing embeddings instead of original words would help to bridge the vocabulary gap.

One popular embedding-based similarity measure is the average cosine similarity computed for all pairs of question and answer terms. The average pairwise cosine similarity is equal to the cosine similarity between averaged word embeddings of questions and answers (we use zero-element vectors for out-of-vocabulary terms). In our work we use the cosine similarity between *IDF-weighted* averaged embeddings. This method works nearly equally well for both lemmas and original words, but using original words results in 1-2 percent gain in accuracy.

In our preliminary experiments [45], we carry out extensive experiments with various embed-

| Training data | NDCG@20 | ERR@20 | Gain in NDCG@20 | Gain in ERR@20 |
|---|---|---|---|---|
| ***Comprehensive*** | | | | |
| **BM25** | | | | |
| | 0.0793 | 0.1308 | 0.0% | 0.0% |
| **Cosine similarity between averaged embeddings for original words** | | | | |
| *Comprehensive* | 0.0521 | 0.0821 | -34.3% | -37.2% |
| Google news | 0.0477 | 0.0760 | -39.8% | -41.9% |
| Google news (unweighted retrofitting) | 0.0498 | 0.0795 | -37.2% | -39.2% |
| ***Stack Overflow*** | | | | |
| **BM25** | | | | |
| | 0.0844 | 0.0879 | 0.00% | 0.00% |
| **Cosine similarity between averaged embeddings for original words** | | | | |
| *Stack Overflow* | 0.0549 | 0.0528 | -34.90% | -39.92% |
| Google news | 0.0442 | 0.0421 | -47.67% | -52.12% |
| Google news (weighted retrofitting) | 0.0448 | 0.0426 | -46.92% | -51.55% |
| ***WikiSQuAD*** | | | | |
| **BM25** | | | | |
| | 0.3196 | 0.2542 | 0.00% | 0.00% |
| **Cosine similarity between averaged embeddings for lemmas** | | | | |
| *WikiSQuAD* | 0.1773 | 0.1230 | -44.52% | -51.60% |

Table 3.16: Effectiveness of the cosine similarity between averaged embeddings (in a re-ranking experiment on *WikiSQuAD dev1* for $M = 100$).

ding types to understand which embeddings work better for our data (for *Comprehensive*). We first evaluated performance of `word2vec` [206] and `doc2vec` [176] implemented in gensim [250]. The latter model computes vector representations for complete documents. We further compared these models with several sets of pre-trained embeddings [206, 238, 315], which were improved by retrofitting [109].

In the case of retrofitting, we use BM25+Model 1 translation tables $T(q|a)$ (see Eq. 3.3) as a relational lexicon. The relational lexicon indicates which words are related (e.g., synonyms are related). The retrofitting procedure attempts to modify word embeddings to increase the cosine similarity between related words. In our cases, it increases similarity between words that tend to co-occur in questions and answer-bearing passages. For these words $T(q|a) > 0$. Retrofitting can be weighted and non-weighted. A weighted variant takes into account a strength of relationship.

In our previous experiments on *Comprehensive*, retrofitting word embeddings trained on 100B token Google News corpus resulted in better models compared to models produced by Gensim. In this work, we carry out an additional comparison with embeddings computed using the original `word2vec` implementation with default settings [206].[25] These embeddings are trained on the complete corpora. In the case of *Comprehensive* and *Stack Overflow* we create embeddings for original lower-cased words (with punctuation removed). In the case of *WikiSQuAD* we use lemmas. We then evaluate embeddings in a re-ranking experiment, where we first retrieve 100 candidates using Lucene and re-rank them using the cosine similarity between averaged word embeddings.

According to results in Table 3.16, using the original `word2vec` implementation resulted in a 5% gain for *Comprehensive* and 22% gain for *Stack Overflow* in terms of NDCG@20. Yet, the overall accuracy is still too low to carry out retrieval using word embeddings alone (i.e., without fusing cosine similarity scores with BM25 and other relevance signals). For this reason, this thesis does not include experiments with dense word representations.

## 3.3 Main Experiments

### 3.3.1 Setup

We carry out experiments on Amazon EC2 instances of two different types. For data sets *Comprehensive* and *Stack Overflow* we use *r3.4xlarge*, which has 16 virtual cores and 122 GB of memory. The main retrieval pipeline uses 16 search threads. For *WikiSQuAD* we use *r3.8xlarge*, which has 32 virtual cores and 244 GB of memory. The main retrieval pipeline uses 32 search threads. The main reason why we need so much memory is inefficiencies of our implementation, which nevertheless are easy to fix if necessary. For a more detailed discussion of minimal memory requirements, please see § 3.3.4.

The retrieval architecture (see § 3.1) is outlined in Figure 3.1. There is a client application that may process requests on its own (with a help of Lucene or Galago) or submit queries to NMSLIB, which works as a standalone server. The client application is implemented in Java (1.8.0_11). The pipeline can work with various term-based retrieval or $k$-NN search methods. Term-based retrieval engines include Lucene (6.0.0), Galago (3.10), and our own C++ implementation of the inverted index that is implemented as a part of NMSLIB. All methods of $k$-NN search are also a part of NMSLIB. Note that we use a modified version of NMSLIB 1.5,[26] which operates as a server processing queries via TCP/IP. NMSLIB is written in C++ and compiled using GCC 4.8.4 with optimization flags `-O3` and `-march=native`.

Retrieval times are measured by the client application that submits search requests to Lucene, Galago, or NMSLIB. In the case of Lucene and Galago, we first "warm up" the index by executing the whole set of queries (which forces OS to cache disk-resident index data in RAM). Run-times are measured only for the second run. Effectiveness of *retrieval runs* is measured using two

[25] Available at `https://code.google.com/archive/p/word2vec/`.

[26] `https://github.com/searchivarius/nmslib/tree/nmslib4a_bigger_reruns`

standard applications provided by NIST: trec_eval 9.0.4[27] and gdeval.[28] Statistical significance of main experimental results is evaluated using the t-test with a subsequent Bonferroni adjustment for multiple testing [40, 100]. This adjustment for multiple testing consists in multiplying p-values by the total number of runs for $M = 100$ (although some of the runs are not shown in the tables to save space). The significance level is 0.01. Significance is computed for comparison against two baselines: the brute-force search using BM25 and a simple Lucene-based fusion pipeline that re-ranks Lucene's output using BM25+Model1.

The values of intrinsic metrics $R_{knn}$@20 and RBO are measured using NMSLIB. The original version of NMSLIB does not support computation of RBO, so it was extended using the code by William Webber [310].[29] The value of parameter $p$ was set to $0.9372$, which means that the first 20 results receive a 90% weight. The value of $p$ was computed by our own script.[30]

### 3.3.2 Tuning

First of all, as explained in § 3.1.1.2, we tune parameters of BM25+Model1, which include the self-translation probability and the smoothing parameter $\lambda$ (see Eq. 3.1.1.2), on a development set (*dev1* or *dev2*). Rather than evaluating individual performance of IBM Model1, we aim to maximize performance of the model that linearly combines BM25 and IBM Model1 scores. In addition to this we pruned the translation table by using entries $T(q|a)$ that exceed a threshold $2.5 \cdot 10^{-3}$. The primary objective of this pruning is to improve efficiency. However, for some data sets (this is true for *Comprehensive* in our preliminary experiments [45]), this can also improve accuracy. We attempted to tune BM25 for *Comprehensive*, but this did not result in substantial improvements. However, we did not tune BM25 for *WikiSQuAD* and *Stack Overflow* .

For each *approximate* $k$-NN pipeline, we execute several runs with different parameters. In the case of SW-graph, we vary the parameter `efSearch`. In the case of NAPP, we vary the number of indexed pivots (parameter `numPivotIndex`) and the number of pivots that should be shared between the query and an answer (`numPivotSearch`). Additionally we have to chose pivot-generation parameters (pivots are generated as pseudo-documents containing $K = 1000$ entries sampled from the set of $M = 50000$ most frequent words *without* replacement), but the set of pivots does not change between runs. All parameters have been selected based on preliminary experiments on development sets. Furthermore, in these experiments, we do not use any extrinsic metric values such as NDCG@20. We rather select a set of $k$-NN recall values in the range of 0.8 to 1 and obtain parameters that maximize efficiency at a given value of $k$-NN recall.

In the case of SDM and PRF/RM3, tuning is difficult, because computation of these similarities is quite slow. For this reason, tuning is carried out on a subset of *dev1*, which contains only 1K queries. Furthermore, we tune parameters in a greedy fashion rather than doing a grid search, because the grid search is impractically slow. For example, in some cases, evaluating a set of 1K queries takes longer than two hours on c4.8 AWS instance with 36 virtual cores.

[27] `https://github.com/usnistgov/trec_eval`

[28] `http://trec.nist.gov/data/web/10/gdeval.pl`

[29] Available at `http://www.williamwebber.com/research/downloads/rbo-0.2.1.tar.gz`.

[30] We verify that this script produces correct results for two cases mentioned in [310]. It is available at `https://github.com/oaqa/knn4qa/blob/bigger_reruns/scripts/nmslib/rbo_weight.py`.

For PRF/RM3, we learn that `fbDocs=5` is optimal for all three collections, which is much smaller than the default value. The default value `fbTerm=100` is good for all three collections, but slightly better results for *WikiSQuAD* are achieved with `fbTerm=140`. The optimal weight of the original query `fbOrigWeight` is slightly different among collections. For *Comprehensive* and *Stack Overflow*, `fbOrigWeight` is 0.9, while for *WikiSQuAD* `fbOrigWeight` is 0.75.

Tuning of PRF/RM3 lead to double digit improvements in NDCG@20 compared to default values (on the tuning subset). For *Stack Overflow*, the improvement is as large as 77%. In contrast, for SDM there is virtually no improvement compared to default values (we have tried to vary only parameters `uniw`, `odw`, and `uww`). This is in line with observation of Metzler and Croft who note that "... with the exception of $\mu$, the SDM parameters are relatively stable. The optimal parameters are similar for all collections and query types, and the parameters are close to the suggested default parameters. " [204]

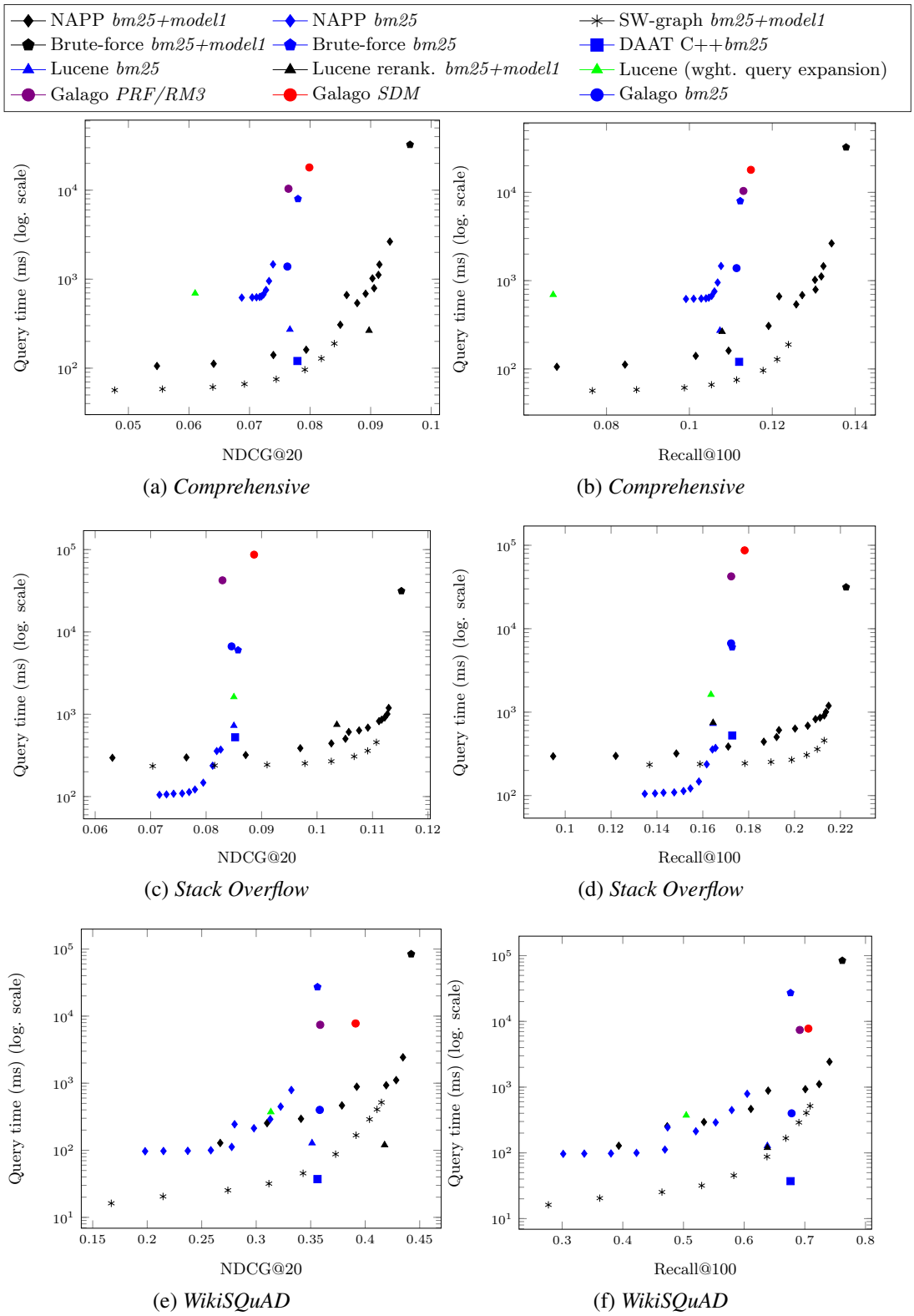
## 3.3.3 Results

### 3.3.3.1 Efficiency-Effectiveness Trade-offs

In this subsection, we discuss **efficiency-effectiveness** trade-offs for three collections and four similarity models: BM25 , BM25+Model 1, SDM, and PRF/RM3. Effectiveness is measured using NDCG@20, ERR@20, and Recall@100 (i.e., answer recall at 100). Efficiency is measured in terms of the average retrieval time. For methods of approximate $k$-NN search, we additionally compute improvement in efficiency (compared to the *exact* brute-force search). These experiments use 10K queries from the test set (see Table 3.4). Detailed experimental results are presented in Tables 3.17, 3.18, and 3.19). Because, in a vast majority of cases the differences are statistically significant, we only mark the cases where we fail to reject the null hypothesis (i.e., when we cannot find statistically significant differences between two runs, because the difference are small and can be attributed to random factors). For all retrieval times, a respective 99.9% confidence interval is within 12% of the reported average values. For *Stack Overflow*, there is less of a variance a respective 99.9% confidence interval is within 4% of the reported average values. An easier-to-digest presentation is given in Figure 3.2, where we present efficiency-effectiveness plot for three collections and two intrinsic metrics: NDCG@20 and Recall@100. In each plot, the average retrieval time is on the Y-axis, while effectiveness (either answer recall or NDCG@20) is on the X-axis. Lower and to the right is better. We omit a plot for ERR@20, because it is quite similar to the plot for NDCG@20.

Note that for *each* method of approximate $k$-NN search, there are multiple runs corresponding to different similarities and search parameters. In that, approximate $k$-NN runs are compared against multiple baselines. First, of all we compare approximate $k$-NN against exact brute-force $k$-NN search (represented by pentagons). Second, we compare against three Lucene-based baselines:

- The Lucene's implementation of BM25 (with subsequent re-ranking using a more accurate BM25 implementation), which is denoted by a blue triangle;
- A simple Lucene-based fusion pipeline (black triangles), where we generate 100 candidates using Lucene's BM25 similarity and further re-rank them using BM25+Model 1;




(a) *Comprehensive* (b) *Comprehensive*

(c) *Stack Overflow* (d) *Stack Overflow*

(e) *WikiSQuAD* (f) *WikiSQuAD*

Figure 3.2: Efficiency-effectiveness trade-offs of top-100 retrieval (lower and to the right is better). Effectiveness metrics include NDCG@20 and answer recall. Best viewed in color: **black** denotes BM25+Model1; **blue** denotes BM25; **violet** denotes PRF/RM3; **red** denotes SDM; **green** triangles denote a weighted query-expansion that relies on BM25.

| Query time (sec) | Speed-up over BF | NDCG@20 | ERR@20 | Answer recall | Relative NDCG@20 | Relative ERR@20 | Relative answer recall | $R_{knn}$@20 | RBO@20 |
|---|---|---|---|---|---|---|---|---|---|
| **Brute-force** *bm25+model1* | | | | | | | | | |
| 32.45 | 1 | 0.097 | 0.134 | 0.138 | 1 | 1 | 1 | 1 | 1 |
| **SW-graph** *bm25+model1* | | | | | | | | | |
| 0.19 | 172 | 0.084 | 0.120 | 0.124 | 0.870 | 0.892 | 0.899 | 0.888 | 0.891 |
| 0.13 | 253 | 0.082$^a$ | 0.117 | 0.121 | 0.848 | 0.874 | 0.879 | 0.867 | 0.870 |
| 0.10 | 338 | 0.079$^a$ | 0.114$^a$ | 0.118 | 0.820 | 0.846 | 0.855 | 0.840 | 0.843 |
| 0.08 | 432 | 0.074$^a$ | 0.107$^a$ | 0.111 | 0.771 | 0.801 | 0.808 | 0.793 | 0.797 |
| 0.07 | 490 | 0.069 | 0.101 | 0.105 | 0.717 | 0.751 | 0.764 | 0.745 | 0.750 |
| 0.06 | 530 | 0.064 | 0.093 | 0.099 | 0.663 | 0.695 | 0.717 | 0.688 | 0.694 |
| 0.06 | 558 | 0.056 | 0.082 | 0.087 | 0.576 | 0.608 | 0.633 | 0.595 | 0.602 |
| 0.06 | 572 | 0.048 | 0.071 | 0.077 | 0.495 | 0.531 | 0.556 | 0.509 | 0.519 |
| **NAPP** *bm25+model1* | | | | | | | | | |
| 2.64 | 12 | 0.093 | 0.131$^b$ | 0.134 | 0.966 | 0.977 | 0.975 | 0.959 | 0.961 |
| 1.46 | 22 | 0.091$^b$ | 0.130$^b$ | 0.132 | 0.948 | 0.965 | 0.960 | 0.938 | 0.943 |
| 1.12 | 29 | 0.091$^b$ | 0.129$^b$ | 0.132 | 0.946 | 0.962 | 0.957 | 0.927 | 0.933 |
| 1.02 | 32 | 0.090$^b$ | 0.128$^b$ | 0.130 | 0.936 | 0.956 | 0.945 | 0.911 | 0.918 |
| 0.79 | 41 | 0.091$^b$ | 0.129$^b$ | 0.130 | 0.939 | 0.958 | 0.947 | 0.900 | 0.908 |
| 0.69 | 47 | 0.089$^b$ | 0.127$^b$ | 0.127 | 0.924 | 0.945 | 0.923 | 0.873 | 0.884 |
| 0.66 | 49 | 0.086 | 0.124 | 0.122 | 0.892 | 0.920 | 0.883 | 0.815 | 0.832 |
| 0.54 | 60 | 0.088$^b$ | 0.125$^b$ | 0.126 | 0.910 | 0.932 | 0.913 | 0.848 | 0.860 |
| 0.31 | 106 | 0.085 | 0.122 | 0.119 | 0.881 | 0.912 | 0.864 | 0.787 | 0.805 |
| 0.16 | 202 | 0.079$^a$ | 0.116$^a$ | 0.109 | 0.822 | 0.864 | 0.794 | 0.712 | 0.735 |
| 0.14 | 231 | 0.074$^a$ | 0.109$^a$ | 0.102 | 0.766 | 0.810 | 0.737 | 0.649 | 0.676 |
| 0.11 | 290 | 0.064 | 0.097 | 0.084 | 0.664 | 0.723 | 0.613 | 0.535 | 0.567 |
| 0.11 | 307 | 0.055 | 0.085 | 0.068 | 0.567 | 0.636 | 0.494 | 0.434 | 0.469 |
| **Brute-force** *bm25* | | | | | | | | | |
| 7.99 | 1 | 0.078 | 0.110 | 0.112 | 1 | 1 | 1 | 1 | 1 |
| **NAPP** *bm25* | | | | | | | | | |
| 1.47 | 5 | 0.074 | 0.106 | 0.108 | 0.947 | 0.967 | 0.959 | 0.962 | 0.963 |
| 0.95 | 8 | 0.073 | 0.106 | 0.107 | 0.939 | 0.961 | 0.951 | 0.953 | 0.954 |
| 0.76 | 11 | 0.073 | 0.105 | 0.106 | 0.932 | 0.957 | 0.945 | 0.944 | 0.947 |
| 0.67 | 12 | 0.072 | 0.105 | 0.105 | 0.927 | 0.953 | 0.939 | 0.936 | 0.939 |
| 0.64 | 12 | 0.072 | 0.104 | 0.105 | 0.923 | 0.949 | 0.932 | 0.927 | 0.931 |
| 0.63 | 13 | 0.072 | 0.104 | 0.104 | 0.919 | 0.947 | 0.926 | 0.917 | 0.923 |
| 0.27 | | 0.077 | 0.108 | 0.107 | Lucene *bm25* | | | | |
| 0.69 | | 0.061 | 0.090 | 0.067 | Lucene (wght. query expansion) | | | | |
| 0.26 | | 0.090 | 0.128 | 0.108 | Lucene rerank. *bm25+model1* | | | | |
| 0.12 | | 0.078$^a$ | 0.110$^a$ | 0.112 | DAAT C++*bm25* | | | | |
| 1.39 | | 0.076$^a$ | 0.107 | 0.111 | Galago *bm25* | | | | |
| 18.00 | | 0.080$^a$ | 0.114$^a$ | 0.115 | Galago *SDM* | | | | |
| 10.37 | | 0.076$^a$ | 0.107 | 0.113 | Galago *PRF/RM3* | | | | |

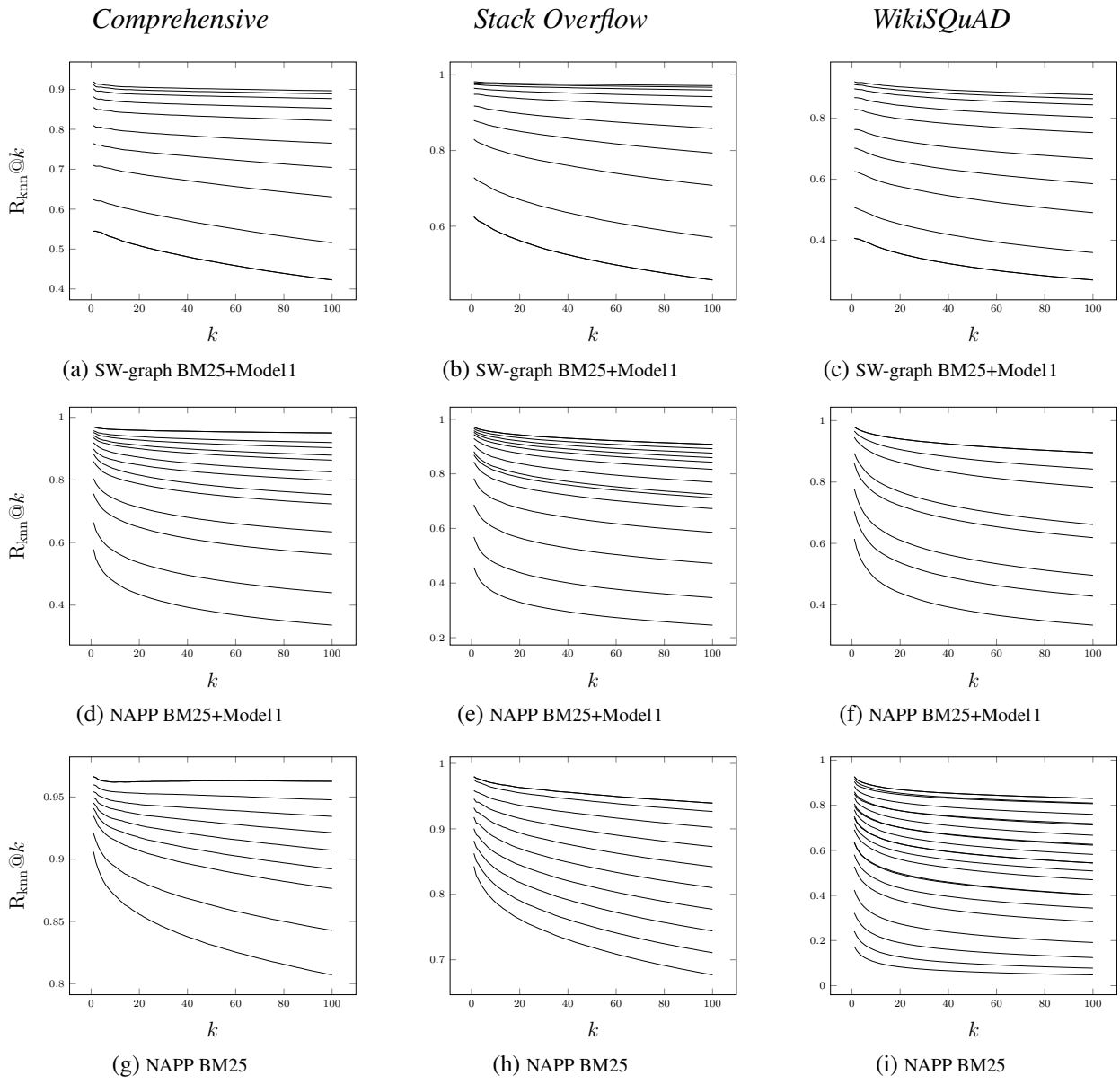
Table 3.17: Detailed experimental results for *Comprehensive*. **Lacking** statistically significant difference is indicated by a superscript: $a$ for brute-force BM25, $b$ for learning-to-rank with BM25+Model1. There are no significance tests for recall.

| Query time (sec) | Speed-up over BF | NDCG@20 | ERR@20 | Answer recall | Relative NDCG@20 | Relative ERR@20 | Relative answer recall | $R_{knn}$@20 | RBO@20 |
|---|---|---|---|---|---|---|---|---|---|
| **Brute-force** *bm25+model1* | | | | | | | | | |
| 31.46 | 1 | 0.1152 | 0.1283 | 0.2225 | 1 | 1 | 1 | 1 | 1 |
| **SW-graph** *bm25+model1* | | | | | | | | | |
| 0.46 | 69 | 0.1107 | 0.1238 | 0.2129 | 0.961 | 0.965 | 0.957 | 0.969 | 0.971 |
| 0.36 | 87 | 0.1091 | $0.1223^{b}$ | 0.2099 | 0.947 | 0.954 | 0.944 | 0.956 | 0.959 |
| 0.31 | 103 | $0.1067^{b}$ | $0.1198^{b}$ | 0.2053 | 0.926 | 0.934 | 0.923 | 0.938 | 0.941 |
| 0.27 | 117 | $0.1026^{b}$ | $0.1154^{b}$ | 0.1987 | 0.890 | 0.899 | 0.893 | 0.898 | 0.904 |
| 0.25 | 124 | 0.0978 | 0.1106 | 0.1897 | 0.849 | 0.862 | 0.853 | 0.851 | 0.858 |
| 0.24 | 129 | $0.0910^{a}$ | $0.1030^{a}$ | 0.1783 | 0.790 | 0.803 | 0.802 | 0.785 | 0.797 |
| 0.24 | 132 | $0.0816^{a}$ | $0.0931^{a}$ | 0.1588 | 0.708 | 0.726 | 0.714 | 0.670 | 0.686 |
| 0.23 | 134 | 0.0704 | 0.0798 | 0.1368 | 0.611 | 0.622 | 0.615 | 0.562 | 0.579 |
| **NAPP** *bm25+model1* | | | | | | | | | |
| 1.19 | 26 | 0.1129 | 0.1265 | 0.2149 | 0.980 | 0.986 | 0.966 | 0.943 | 0.950 |
| 1.01 | 31 | 0.1127 | 0.1262 | 0.2137 | 0.978 | 0.984 | 0.961 | 0.932 | 0.941 |
| 0.91 | 34 | 0.1122 | 0.1258 | 0.2130 | 0.974 | 0.980 | 0.957 | 0.920 | 0.930 |
| 0.86 | 37 | 0.1116 | 0.1254 | 0.2111 | 0.969 | 0.977 | 0.949 | 0.909 | 0.920 |
| 0.82 | 38 | 0.1112 | 0.1251 | 0.2091 | 0.965 | 0.975 | 0.940 | 0.896 | 0.909 |
| 0.69 | 46 | 0.1092 | $0.1229^{b}$ | 0.2058 | 0.948 | 0.958 | 0.925 | 0.875 | 0.889 |
| 0.64 | 50 | 0.1076 | $0.1216^{b}$ | 0.2002 | 0.934 | 0.948 | 0.900 | 0.838 | 0.855 |
| 0.61 | 52 | $0.1057^{b}$ | $0.1196^{b}$ | 0.1932 | 0.918 | 0.932 | 0.868 | 0.801 | 0.821 |
| 0.50 | 63 | $0.1051^{b}$ | $0.1195^{b}$ | 0.1922 | 0.912 | 0.931 | 0.864 | 0.787 | 0.808 |
| 0.44 | 71 | $0.1026^{b}$ | $0.1172^{b}$ | 0.1866 | 0.890 | 0.913 | 0.839 | 0.752 | 0.775 |
| 0.39 | 81 | 0.0970 | 0.1115 | 0.1711 | 0.842 | 0.869 | 0.769 | 0.674 | 0.701 |
| 0.32 | 98 | $0.0871^{a}$ | $0.1009^{a}$ | 0.1485 | 0.756 | 0.787 | 0.667 | 0.565 | 0.596 |
| **Brute-force** *bm25* | | | | | | | | | |
| 6.01 | 1 | 0.0858 | 0.0964 | 0.1728 | 1 | 1 | 1 | 1 | 1.000 |
| **NAPP** *bm25* | | | | | | | | | |
| 0.37 | 16 | 0.0826 | 0.0933 | 0.1655 | 0.964 | 0.968 | 0.958 | 0.963 | 0.968 |
| 0.36 | 17 | 0.0819 | 0.0927 | 0.1642 | 0.955 | 0.962 | 0.950 | 0.955 | 0.960 |
| 0.11 | 56 | 0.0741 | 0.0851 | 0.1429 | 0.865 | 0.883 | 0.827 | 0.818 | 0.835 |
| 0.11 | 57 | 0.0729 | 0.0840 | 0.1392 | 0.850 | 0.871 | 0.805 | 0.790 | 0.809 |
| 0.11 | 57 | 0.0716 | 0.0826 | 0.1347 | 0.835 | 0.857 | 0.780 | 0.762 | 0.783 |
| 0.24 | 25 | 0.0812 | 0.0918 | 0.1617 | 0.947 | 0.952 | 0.936 | 0.936 | 0.942 |
| 0.15 | 41 | 0.0795 | 0.0901 | 0.1583 | 0.927 | 0.935 | 0.916 | 0.916 | 0.924 |
| 0.12 | 49 | 0.0780 | 0.0887 | 0.1546 | 0.909 | 0.921 | 0.895 | 0.894 | 0.904 |
| 0.73 | | 0.0850 | $0.0957^{a}$ | 0.1645 | Lucene *bm25* | | | | |
| 1.62 | | 0.0850 | $0.0957^{a}$ | 0.1636 | Lucene (wght. query expansion) | | | | |
| 0.75 | | 0.1036 | 0.1198 | 0.1645 | Lucene rerank. *bm25+model1* | | | | |
| 0.53 | | $0.0852^{a}$ | $0.0958^{a}$ | 0.1728 | DAAT C++*bm25* | | | | |
| 6.68 | | $0.0846^{a}$ | $0.0947^{a}$ | 0.1723 | Galago *bm25* | | | | |
| 86.93 | | 0.0887 | $0.0999^{a}$ | 0.1782 | Galago *SDM* | | | | |
| 42.37 | | 0.0830 | 0.0921 | 0.1724 | Galago *PRF/RM3* | | | | |

Table 3.18: Detailed experimental results for *Stack Overflow*. **Lacking** statistically significant difference is indicated by a superscript: $a$ for brute-force BM25, $b$ for learning-to-rank with BM25+Model1. There are no significance tests for recall.

| Query time (sec) | Speed-up over BF | NDCG@20 | ERR@20 | Answer recall | Relative NDCG@20 | Relative ERR@20 | Relative answer recall | $R_{knn}$@20 | RBO@20 |
|---|---|---|---|---|---|---|---|---|---|
| **Brute-force** *bm25+model1* | | | | | | | | | |
| 84.18 | 1 | 0.4422 | 0.3620 | 0.7618 | 1 | 1 | 1 | 1 | 1 |
| **SW-graph** *bm25+model1* | | | | | | | | | |
| 0.52 | 163 | 0.4149$^b$ | 0.3402 | 0.7087 | 0.938 | 0.940 | 0.930 | 0.9030 | 0.9079 |
| 0.41 | 207 | 0.4109$^b$ | 0.3368 | 0.7025 | 0.929 | 0.930 | 0.922 | 0.8927 | 0.8980 |
| 0.29 | 290 | 0.4040 | 0.3309 | 0.6903 | 0.914 | 0.914 | 0.906 | 0.8759 | 0.8820 |
| 0.17 | 503 | 0.3916 | 0.3207 | 0.6687 | 0.886 | 0.886 | 0.878 | 0.8422 | 0.8501 |
| 0.09 | 963 | 0.3729 | 0.3050 | 0.6377 | 0.843 | 0.843 | 0.837 | 0.7985 | 0.8081 |
| 0.05 | 1855 | 0.3430 | 0.2813$^a$ | 0.5830 | 0.776 | 0.777 | 0.765 | 0.7266 | 0.7378 |
| 0.03 | 2647 | 0.3117 | 0.2553 | 0.5300 | 0.705 | 0.705 | 0.696 | 0.6576 | 0.6705 |
| 0.03 | 3322 | 0.2740 | 0.2238 | 0.4645 | 0.620 | 0.618 | 0.610 | 0.5760 | 0.5903 |
| 0.02 | 4111 | 0.2146 | 0.1745 | 0.3621 | 0.485 | 0.482 | 0.475 | 0.4525 | 0.4683 |
| 0.02 | 5196 | 0.1671 | 0.1358 | 0.2774 | 0.378 | 0.375 | 0.364 | 0.3558 | 0.3711 |
| **NAPP** *bm25+model1* | | | | | | | | | |
| 2.43 | 35 | 0.4347 | 0.3567 | 0.7407 | 0.983 | 0.985 | 0.972 | 0.9402 | 0.9503 |
| 1.11 | 76 | 0.4284 | 0.3522 | 0.7237 | 0.969 | 0.973 | 0.950 | 0.9057 | 0.9213 |
| 0.93 | 90 | 0.4193$^b$ | 0.3454$^b$ | 0.7009 | 0.948 | 0.954 | 0.920 | 0.8637 | 0.8850 |
| 0.89 | 95 | 0.3923 | 0.3245 | 0.6395 | 0.887 | 0.896 | 0.840 | 0.7681 | 0.8007 |
| 0.47 | 181 | 0.3787 | 0.3135 | 0.6112 | 0.856 | 0.866 | 0.802 | 0.7233 | 0.7575 |
| 0.29 | 286 | 0.3411 | 0.2841$^a$ | 0.5338 | 0.771 | 0.785 | 0.701 | 0.6106 | 0.6528 |
| 0.25 | 332 | 0.3100 | 0.2596 | 0.4732 | 0.701 | 0.717 | 0.621 | 0.5374 | 0.5798 |
| 0.13 | 656 | 0.2670 | 0.2249 | 0.3934 | 0.604 | 0.621 | 0.516 | 0.4387 | 0.4832 |
| **Brute-force** *bm25* | | | | | | | | | |
| 27.16 | 3 | 0.3563 | 0.2841 | 0.6764 | 1 | 1 | 1 | 1 | 1 |
| **NAPP** *bm25* | | | | | | | | | |
| 0.79 | 106 | 0.3323 | 0.2670 | 0.6050 | 0.933 | 0.940 | 0.894 | 0.8695 | 0.8837 |
| 0.45 | 188 | 0.3224 | 0.2596 | 0.5796 | 0.905 | 0.914 | 0.857 | 0.8476 | 0.8615 |
| 0.29 | 290 | 0.3129 | 0.2527 | 0.5531 | 0.878 | 0.890 | 0.818 | 0.8123 | 0.8300 |
| 0.24 | 344 | 0.2802 | 0.2280 | 0.4739 | 0.786 | 0.803 | 0.701 | 0.7010 | 0.7281 |
| 0.21 | 395 | 0.2979 | 0.2413 | 0.5204 | 0.836 | 0.849 | 0.769 | 0.7756 | 0.7948 |
| 0.11 | 750 | 0.2776 | 0.2257 | 0.4694 | 0.779 | 0.794 | 0.694 | 0.7014 | 0.7262 |
| 0.10 | 842 | 0.2584 | 0.2118 | 0.4226 | 0.725 | 0.746 | 0.625 | 0.6292 | 0.6587 |
| 0.10 | 858 | 0.2372 | 0.1950 | 0.3805 | 0.666 | 0.687 | 0.563 | 0.5598 | 0.5935 |
| 0.10 | 864 | 0.2149 | 0.1778 | 0.3365 | 0.603 | 0.626 | 0.497 | 0.4939 | 0.5298 |
| 0.13 | | 0.3512 | 0.2806 | 0.6383 | Lucene *bm25* | | | | |
| 0.37 | | 0.3134 | 0.2551 | 0.5045 | Lucene (wght. query expansion) | | | | |
| 0.12 | | 0.4178 | 0.3469 | 0.6383 | Lucene rerank. *bm25+model1* | | | | |
| 0.04 | | 0.3563 | 0.2841 | 0.6764 | DAAT C++*bm25* | | | | |
| 0.40 | | 0.3583$^a$ | 0.2857$^a$ | 0.6782 | Galago *bm25* | | | | |
| 7.77 | | 0.3912 | 0.3173 | 0.7059 | Galago *SDM* | | | | |
| 7.42 | | 0.3587$^a$ | 0.2822$^a$ | 0.6917 | Galago *PRF/RM3* | | | | |

Table 3.19: Detailed experimental results for *WikiSQuAD*. **Lacking** statistically significant difference is indicated by a superscript: $a$ for brute-force BM25, $b$ for learning-to-rank with BM25+Model1. There are no significance tests for recall.




(a) SW-graph BM25+Model1 (b) SW-graph BM25+Model1 (c) SW-graph BM25+Model1

(d) NAPP BM25+Model1 (e) NAPP BM25+Model1 (f) NAPP BM25+Model1

(g) NAPP BM25 (h) NAPP BM25 (i) NAPP BM25

Figure 3.3: Values of $\mathrm{R_{knn}}$@k as a function of $k$. Each line corresponds to a unique combination of parameters. We merely show a family of results without focusing on individual parameter configurations.

- A Lucene-based query expansion (green triangles), where for each query term $q$ we add five of its Model 1 "synonyms", i.e., terms $w$ such that $T(q|w) > 0$. We weigh each expansion term $w$ using the respective translation probability $T(q|w)$, because the weighted expansion is more accurate (sometimes by a wide margin) than non-weighted one. This baseline is used to demonstrate that query expansion is expensive.

Lucene is a strong baseline, which fares well against optimized C++ code, especially for disjunctive queries [187, 297]. However, for a fairer efficiency comparison, we reimplement Lucene's algorithm for posting processing in C++. As explained previously, this method (labeled *DAAT C++*) does not use any compression and does not explicitly compute BM25. It also uses the index fully loaded into memory. Respective runs are denoted by blue squares.

First, we can see that exact brute-force $k$-NN search is, indeed, quite slow, but corresponding approximate runs can be one-four orders of magnitude faster, sometimes at the expense of only a small loss in effectiveness. More importantly, these plots seem to support our conjecture that incorporation of sophisticated similarity into retrieval can be beneficial. This can be seen by comparing black diamond runs with blue diamond runs. In both cases, we use the same pivoting method NAPP and the same set of pivots. The difference is that the distance to pivots is computed using different similarity models: BM25 and BM25+Model 1. Although computing BM25+Model 1is an order of magnitude more expensive (compare retrieval times of brute-force runs represented by blue and black diamonds), it is sometimes possible to obtain a black diamond run that outstrips a blue diamond run in NDCG@20 while being equally or more efficient. This effect is especially prominent in the case of *Comprehensive* where for each blue diamond (i.e., BM25 ) run there is a clearly superior black diamond (i.e., BM25+Model 1) run.

In the case of *Stack Overflow*, only for two slowest blue diamond runs there are clearly superior black diamond ones. Quite interestingly though, these black runs are also more accurate and efficient than the DAAT C++ baseline, which ranks results using BM25. In terms of NDCG@20 these black diamond runs do not beat another important baseline: a simple Lucene-based BM25+Model 1 fusion baseline (black diamond). However, there is one black diamond run that is equally efficient to DAAT C++ and slightly more accurate than the fusion baseline. Furthermore, there are two runs for SW-graph (black stars) that are substantially more efficient than DAAT C++ baseline are slightly more accurate than the fusion BM25+Model 1. However, the differences in accuracy are small. There is also lack of statistical significance with respect to ERR@20 (see Table 3.18): formally we cannot reject a hypothesis that these runs are different. One SW-graph run is nearly 2× faster than DAAT C++ and is as accurate as the fusion pipeline BM25+Model 1.

Note that none of the query-expansion baselines is more effective than BM25. In that, they are both less efficient. Compared to Lucene BM25, the query expansion on top of Lucene is 2-3× slower than Lucene without expansion. Galago PRF/RM3 is 7-19× slower compared to Galago BM25. Both query-expansion methods add several additional terms for each original query term, which leads to noticeably longer retrieval times. A more precise query expansion is possible (see, e.g., [155]) but—to the best of our knowledge—only if user behavior data (i.e., query sessions and click logs) is available. This user behavior data may still have poor coverage for tail queries.

In contrast to query expansion methods, SDM is always more effective than BM25: in terms of NDCG@20 it is roughly 5% more effective for *Comprehensive* and *Stack Overflow*. For

*WikiSQuAD*, SDM is 9% more effective. However, SDM is not more efficient than PRF/RM3. The difference is particularly large for *WikiSQuAD* where SDM is 19× slower than Galago's implementation of BM25. It is not clear to us why this is the case.

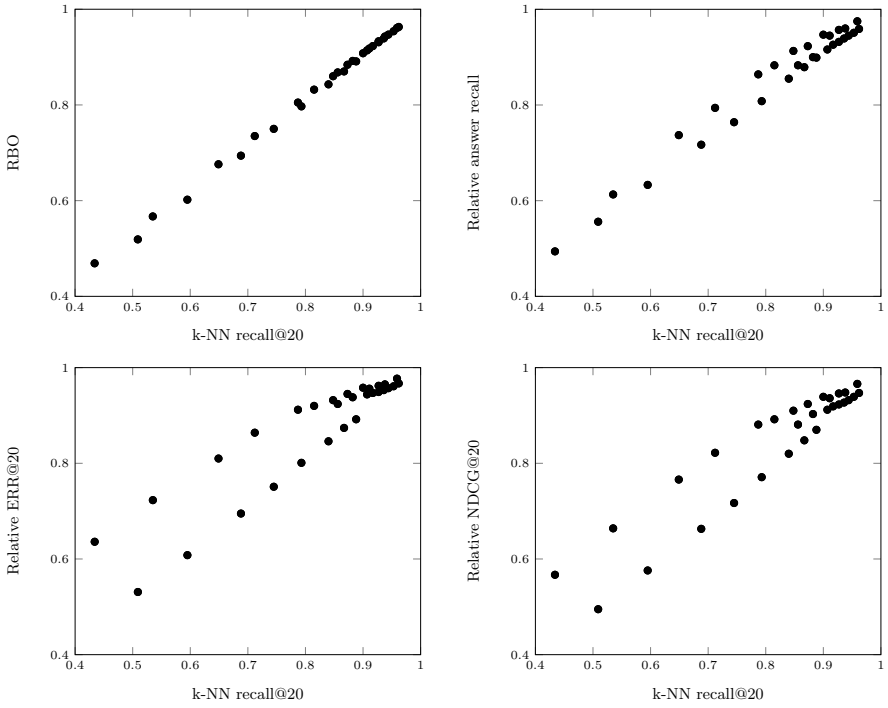



Figure 3.4: Relationship between $R_{knn}@20$ and other accuracy metrics for *Comprehensive*

#### 3.3.3.2 Relationship Between Intrinsic and Extrinsic Metrics

In this section, we study the relationship between values of intrinsic and extrinsic metrics. To this end, we measure the relative values of IR, i.e., extrinsic, metrics. For example, a relative value of NDCG@20 is equal to the actual value of NDCG@20 obtained for an approximate $k$-NN run divided by the value of NDCG@20 for corresponding exact brute-force run. Detailed results can be found in Tables 3.4-3.6.

First, as we can see in all agreement plots, there is a nearly ideal agreement in values of RBO and $R_{knn}@20$. This might seem surprising, but we think that an explanation is simple. RBO is a weighted value of $R_{knn}@k$ for various values of $k$. Because we place a 90% weight on the set of top 20 results, the $R_{knn}@k$ values for $k \leq 20$ largely define the value of RBO. In Figure 3.3, we plot an empirically obtained relationship between $R_{knn}@k$ and $k$. Note that each line in the plot corresponds to a unique configuration of parameters. Because specific details about these configurations are not relevant to the discussion, We merely show a family of results without focusing on individual parameter configurations. As we can see from the plots, the value of recall does decrease with $k$, but the decrease is typically moderate for small $k$. Because, 90% of the value of RBO is computed as an average of 20 recall values each of which is close to $R_{knn}@20$, the value of RBO should be quite close to the value $R_{knn}@20$.

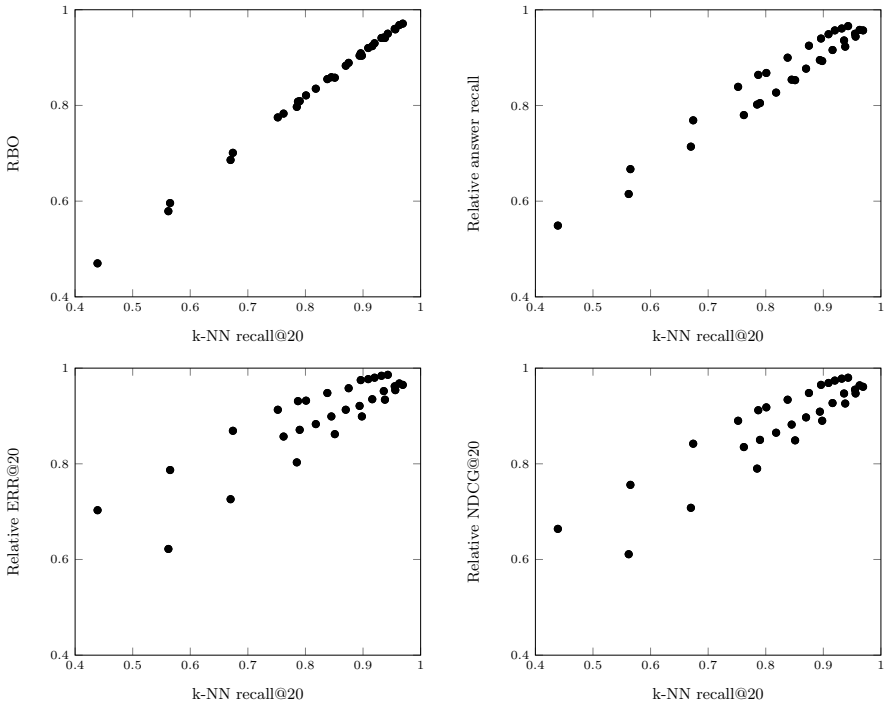



Figure 3.5: Relationship between $R_{knn}@20$ and other accuracy metrics for *Stack Overflow*

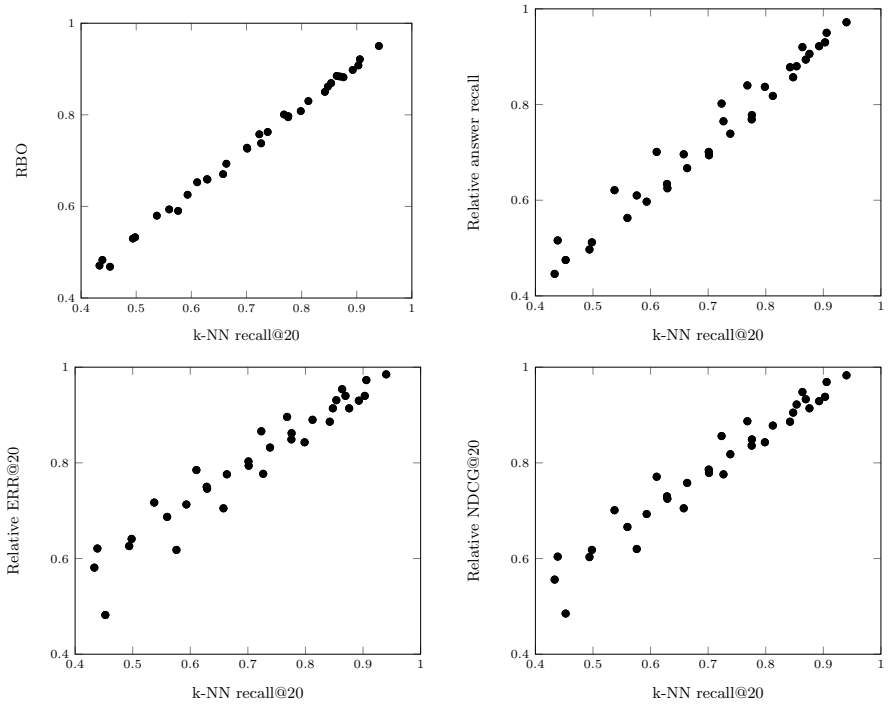



Figure 3.6: Relationship between $R_{knn}@20$ and other accuracy metrics for *WikiSQuAD*

Second, we can see that applying approximate search gives us no free pass to efficiency, because there is a nearly monotonic relationship between the intrinsic and extrinsic effectiveness values. The exact degree of degradation does depend on the method of $k$-NN search, its parameters, and the data set. We can that the values of NDCG@20 and ERR@20 may decline somewhat slower than the values of $R_{knn}$@20. In particular, from Tables 3.17, 3.18, and 3.19), we can see that this is the case of NAPP with BM25+Model1 similarity. Still, we can see that retaining 90% of the original value of NDCG@20 requires $R_{knn}$@from 70% to 90%. SW-graph with BM25+Model1 similarity is less forgiving and retaining 90% of the original value of NDCG@20 requires a nearly 90% recall for all three data sets.

Imagine now a situation where we have a simple retrieval baseline model, which we want to replace with an approximate $k$-NN search method that employs a more sophisticated similarity model. Let the latter model outperform the simple baseline by 20% in answer recall at rank 20. Let us also assume that there are no substantial retrieval biases so that both extrinsically relevant and non-relevant data points are dropped from the top-20 result set randomly and independently with the same probability $r$ (i.e., as the number of queries increases the value of $R_{knn}$@20 converges to $r$).

Because 20-NN search is supposed to always retrieve 20 data points, the relevant data point missed by an approximate $k$-NN search point might be replaced by a relevant entry that did not make it to the exact top-20 result set. However, if the overall number of relevant entries is small, which is not an unreasonable assumption, the probability of such a replacement is small as well. Thus, due to approximate nature of the search, we expect that the average number of relevant top-10 entries retrieved decreases roughly by a factor of $r^{-1}$. Consequently, if $r \approx 0.83$ ($R_{knn}$@20$\approx 83\%$), the accuracy of the $k$-NN search with a more sophisticated model is no better than the accuracy of the baseline model. If we want to outperform the baseline by 10%, the value of $R_{knn}$@20 should be more than $91\%$.

### 3.3.4 Index Size and Creation Times

All our methods for $k$-NN search have longer indexing times than classic term-based indices. In particular, on a *r3.4xlarge* AWS instance with 16 virtual cores, it takes only 2 minutes to create an uncompressed term-based index for *WikiSQuAD*, which is our largest collection. In contrast, on the same hardware, creating a NAPP index with BM25 similarity takes 0.5 hour, creating an SW-graph index (using BM25 similarity to build a graph) takes 5.1 hour, and creating a NAPP index with BM25+Model1 similarity takes 27.6 hours.

These indexing times are clearly long, but they are not impractical. In particular, a "deep" parsing using existing off-the-shelf NLP tools can be even more expensive. Named entity linkers and recognizers are some of the fastest NLP tools. Yet, it takes more than 24 hours to fully annotate Wikipedia using fast multi-threaded tools such as spaCy [135] and DbPedia Spotlight [81] (we count total execution time for both NLP tools). Obtaining word embeddings is also a slow process compared to building a term-based inverted file. For example, it takes half an hour for `word2vec` [206] to train word embeddings on a lemmatized and stopped Wikipedia corpus on a 16 core instance *r3.4xlarge* (`word2vec` uses all available cores).

Compared to term-based inverted indices, distance-based methods of $k$-NN search are less memory-efficient. This is primarily because they rely on a horizontal model of storage (also

| *Comprehensive* | | | | |
|---|---|---|---|---|
| Uncompressed format | Uncompressed size (GB) | Compression algorithm | Compressed size (GB) | Compression ratio |
| Document ids + Frequencies | 2.9 | Variable Byte | 1.0 | 0.35 |
| Text (w/o stopwords) | 2.3 | varint-G8IU | 1.3 | 0.44 |
| *Stack Overflow* | | | | |
| Uncompressed format | Uncompressed size (GB) | Compression algorithm | Compressed size (GB) | Compression ratio |
| Document ids + Frequencies | 1.7 | Variable Byte | 0.5 | 0.32 |
| Text (w/o stopwords) | 1.5 | varint-G8IU | 0.7 | 0.42 |
| *WikiSQuAD* | | | | |
| Uncompressed format | Uncompressed size (GB) | Compression algorithm | Compressed size (GB) | Compression ratio |
| Document ids + Frequencies | 6.4 | Variable Byte | 2.2 | 0.35 |
| Text (w/o stopwords) | 5.7 | varint-G8IU | 2.7 | 0.43 |

Table 3.20: The sizes of compressed and uncompressed forward indices

known as a forward index/file), where data point attributes (belonging to the same data point) are stored jointly, often in the same memory region. Data point attributes are also retrieved jointly using data point identifiers as keys. In contrast, term-based inverted indices rely on a vertical model of storage, where term occurrence information is stored in the form of posting lists, where postings of the same term are stored jointly. The postings can also be ordered by document identifiers and/or in-document term offsets, which permits efficient data differencing and compression. Posting list compression is a well-studied topic with many good results. In fact, the use of modern light-weight compression entails virtually no slowdown in query processing [179, 185].

Yet, it is not clear to us if forward indices can be compressed as effectively and decompressed as efficiently. The main approach to compression of term-based inverted files, sometimes called a d-gap compression, consists in encoding differences between document identifiers in adjacent postings. These differences tend to be distributed rather uniformly (thus, performance results obtained for randomly generated gaps generally match results obtained on real data sets [178]). However, the distribution of terms in a collection is approximately Zipfian [18]. Consequently, the differences between term identifiers are much less uniform (they tend to be larger towards the end of the list), which makes traditional d-gap techniques less effective. More generic compression techniques such as Huffman coding [140], arithmetic coding (see [316] for historical references), or Lempel-Ziv algorithms [330] seem to be much less efficient than byte- or word-aligned coding schemes currently used for compression of term-based inverted files, in particular, because they often require manipulations with individual bits. An efficient library Snappy[31] implements a variant of Lempel-Ziv algorithm using only byte-aligned codes. Yet, it is still nearly an order of magnitude slower than word-aligned bitpacking [178]. In any case, we are not aware of other compression techniques that are competitive against word-aligned compression methods currently used in IR.

Although we have not carried out experiments using compressed forward files, we have nevertheless computed their size for two variable-byte compression algorithms: the classic *Variable Byte* compression algorithm [285] and *Varint-G8IU* [277]. These algorithms do not rely on grouping of integers into large blocks (and, hence, have small overhead per compressed entry), which fits well to the task of encoding each forward index entry separately.

For better compression, we sort all word identifiers in the increasing order and apply d-gap coding. However, we do not transform frequency values as they are already small numbers. The index sizes are estimated by reading serialized indices, compressing respective index entries, and memorizing the resulting sizes. We use Python wrappers for the FastPFOR library.[32]

Variable Byte, which is known under many names (v-byte, variable-byte, var-byte, vbyte, varint, VInt, VB, or even Escaping) is a fairly slow compression algorithm capable of decoding less than one billing integers per second [178]. Varint-G8IU is a much faster modification of Variable Byte accelerated using SIMD instructions: It can decode roughly two billion integers per second. According to the results in Table 3.20, we obtain a roughly three-fold compression with the slower Variable Byte (32-35% compression rate). For the much faster algorithm Varint-G8IU, however, the compression rates are somewhat worse: 42-44%.

[31] `https://github.com/google/snappy`
[32] `https://github.com/searchivarius/PyFastPFor`

| Uncompressed (size in GB) | Compressed (size in GB) | Compression ratio | Index parameters |
|---|---|---|---|
| **SW-graph** *BM25+Model1* | | | |
| 8.3 | 5.7 | 0.69 | NN=50,efConstruction=100 |
| **NAPP** *BM25+Model1* | | | |
| 2.1 | 0.9 | 0.41 | numPivot=8000,numPivotIndex=25 |
| 4.2 | 1.4 | 0.35 | numPivot=8000,numPivotIndex=50 |
| 8.3 | 2.5 | 0.30 | numPivot=8000,numPivotIndex=100 |
| 12.5 | 3.5 | 0.28 | numPivot=8000,numPivotIndex=150 |
| 16.7 | 4.4 | 0.27 | numPivot=8000,numPivotIndex=200 |
| 20.9 | 5.3 | 0.26 | numPivot=8000,numPivotIndex=250 |
| 25.0 | 6.2 | 0.25 | numPivot=8000,numPivotIndex=300 |
| **NAPP** *BM25* | | | |
| 16.7 | 4.4 | 0.27 | numPivot=8000,numPivotIndex=200 |
| 19.0 | 4.9 | 0.26 | numPivot=8000,numPivotIndex=228 |
| 19.9 | 5.1 | 0.26 | numPivot=8000,numPivotIndex=238 |

Table 3.21: Estimates for compressed and uncompressed index sizes for *Comprehensive*. NAPP is compressed using SIMD-accelerated binary packing. SW-graph is compressed using Varint-G8IU [178]. Compression is applied to differences between sorted adjacent identifiers.

| | Compressed (size in GB) | Compression ratio | Index parameters |
|---|---|---|---|
| **SW-graph** *bm25+model1* | | | |
| 4.3 | 2.9 | 0.67 | NN=50,efConstruction=100 |
| **NAPP** *bm25+model1* | | | |
| 1.1 | 0.4 | 0.41 | numPivot=8000,numPivotIndex=25 |
| 2.1 | 0.7 | 0.35 | numPivot=8000,numPivotIndex=50 |
| 4.3 | 1.3 | 0.30 | numPivot=8000,numPivotIndex=100 |
| 6.4 | 1.8 | 0.28 | numPivot=8000,numPivotIndex=150 |
| 8.6 | 2.3 | 0.26 | numPivot=8000,numPivotIndex=200 |
| 10.7 | 2.7 | 0.25 | numPivot=8000,numPivotIndex=250 |
| **NAPP** *bm25* | | | |
| 4.3 | 1.3 | 0.30 | numPivot=8000,numPivotIndex=100 |
| 8.6 | 2.2 | 0.26 | numPivot=8000,numPivotIndex=200 |

Table 3.22: Estimates for compressed and uncompressed index sizes for *Stack Overflow*. NAPP is compressed using SIMD-accelerated binary packing. SW-graph is compressed using Varint-G8IU [178]. Compression is applied to differences between sorted adjacent identifiers.

| | Compressed (size in GB) | Compression ratio | Index parameters |
|---|---|---|---|
| **SW-graph** *bm25+model1* | | | |
| 11.2 | 7.1 | 0.63 | NN=50,efConstruction=100 |
| **NAPP** *bm25+model1* | | | |
| 5.6 | 2.0 | 0.35 | numPivot=8000,numPivotIndex=50 |
| 11.2 | 3.4 | 0.31 | numPivot=8000,numPivotIndex=100 |
| 22.4 | 6.0 | 0.27 | numPivot=8000,numPivotIndex=200 |
| **NAPP** *bm25* | | | |
| 2.8 | 1.1 | 0.41 | numPivot=8000,numPivotIndex=25 |
| 5.6 | 2.0 | 0.35 | numPivot=8000,numPivotIndex=50 |
| 11.2 | 3.4 | 0.31 | numPivot=8000,numPivotIndex=100 |
| 22.4 | 6.1 | 0.27 | numPivot=8000,numPivotIndex=200 |

Table 3.23: Estimates for compressed and uncompressed index sizes for *WikiSQuAD*. NAPP is compressed using SIMD-accelerated binary packing. SW-graph is compressed using Varint-G8IU [178]. Compression is applied to differences between sorted adjacent identifiers.

Beside the forward file, a method of $k$-NN search requires an additional index. In the case of NAPP, it consists of an inverted file that keeps `numPivotIndex` closest pivots of each data point. To match performance of our Lucene-based fusion pipeline in NDCG@20, we need `numPivotIndex`$\geq 100$ for *Comprehensive* and *Stack Overflow* and `numPivotIndex`$\geq 200$ for *WikiSQuAD*. In the case of SW-graph, insertion of each data point entails creation of $2\times$`NN` graph edges, each of which is represented by an integer number. In our experiments, we use `NN`=50 for all collections. Hence, in addition to the forward index, we also need to store 100 integers per data point.

In contrast, for a term-based inverted file (without positions), the inverted file keeps $|D|$ document identifiers and $|D|$ term frequencies ($|D|$ is the number of document terms). In total we need to store $2|D|$ integers for documents. For our collections (see Table 3.4), we, on average, need to store 40-50 integers per answer for *Comprehensive* or *Stack Overflow* and 70 integers per answer-bearing passage for *WikiSQuAD*.

Overall, assuming that (1) the size of the compressed forward file is $2\times$ the size of the term-based inverted files and (2) $k$-NN indices can be compressed as effectively as term-based indices, in our scenario, a $k$-NN index is 2.5-7$\times$ as large as the term-based index. However, the latter is likely true only for NAPP. In the case of SW-graph, we need to compress small lists of neighbor identifiers, where gaps between nearest identifiers are typically quite large. Hence, the d-gap compression is unlikely to be effective.

To verify our guess, we estimate $k$-NN index sizes using methodology similar to the estimation of the forward file size. Namely, index sizes are estimated by reading serialized indices, compressing respective index entries, and memorizing the resulting sizes. We again use Python wrappers for the FastPFOR library.[33] In the case of SW-graph, we compress each list of neighbors separately. Because these lists are relatively short, we use a fast-to-decode algorithm, namely Varint-G8IU, which does not require grouping neighbor identifiers into large blocks [178] and has a small overhead per each compressed entry. However, the index of NAPP is a classic inverted index, where each pivot is represented by a long list of integers. Here we use an extremely efficient SIMD-accelerated binary packing algorithm, which can decompress four billion integers per second (which is as fast as copying integers using the C function `memcpy`) [179]. This algorithm groups integers into blocks containing 128 elements. It has a noticeable overhead. Hence, we apply it only to long integer lists. For both NAPP and SW-graph we use d-gap coding before applying a compression algorithm.

Estimates of the compressed indices are given in Tables 3.21-3.23. We can see that the index of SW-graph is, indeed, poorly compressible: The compression ratio is in the range 63-69%. Moreover, for all three collections, the size of the compressed SW-graph index is at least $3\times$ of the size of the compressed forward index. The index of NAPP is more compressible: The compression ratio can be as low as 25%. NAPP indices are typically more compact than SW-graph indices. However, their size varies greatly depending on the number of indexed pivots (parameter `numPivotIndex`). In the case of *Comprehensive* and `numPivotIndex`=300, the size of the NAPP index exceeds the size of the SW-graph index (see Table 3.21).

We conjecture that in some cases, it may be possible to reduce memory overhead. First, we suppose that the overhead would be smaller if documents were longer. For high accuracy, we

[33] `https://github.com/searchivarius/PyFastPFor`

would still need to index a large number of pivots in NAPP (e.g., `numPivotIndex`=100) and create proximity graphs with a large number of neighbors (e.g., `NN`=50). However, the relative size of the additional $k$-NN index would be smaller compared to the data set size.

Second, in the case of NAPP or another filter-and-refine method, e.g, LSH, it may be possible to share a single instance of the forward index among several servers. For example, in some cases, NAPP can spend as much as 90% of the time on processing the inverted file and only 10% on computing BM25+Model 1 similarity between candidate documents and the query. Imagine that we designate a separate pool of servers to generate candidates. These candidates can be sent to a much smaller pool of servers, which actually compute BM25+Model1. Because only servers from the second pool need a copy of the forward index, this approach could greatly reduce memory requirements. Such splitting of the index, however, seems impossible for graph-based approaches such as SW-graph.

Third, in the case of NAPP, it may be possible to carry out an efficient refinement step using only the inverted index. This can be done by employing a well-known folklore procedure that *reconstructs* forward file entries using only the inverted file and the set of terms $\{t_i\}$ that participate in the computation of the query-document similarity. In the case of BM25, the set $\{t_i\}$ includes only query terms. In the case of BM25+Model 1, the set $\{t_i\}$ includes both the query terms and other collection terms with sufficiently large translation probabilities (with respect to the query terms). The efficiency of such procedure is roughly equivalent to the efficiency of the BM25+Model1-based query expansion (see p. 134 for details). Although the overall number of expanded terms can be quite large (in the order of thousands), a vast majority of expansion terms are quite rate. Thus, this massive expansion approach can still be reasonably efficient. In fact, we verified that query expansion using about one thousand terms from BM25+Model 1 translation dictionary is only marginally slower compared to the same expansion using five terms.

During verification step, we carry out an intersection between the array of sorted document identifiers $\{d_i\}$ obtained during the filtering step and posting lists of terms $\{t_i\}$. Such reconstruction can be quite cheap for an inner-product similarity. Yet, it is more expensive for BM25+Model 1, because each query term can be potentially translated to hundreds and thousands different document terms. Verifying the practicality of this approach remains for future work.

The studied methods of $k$-NN search have higher memory requirements and longer indexing times. Yet, this alone does not disqualify them from practical use. A search method with higher memory requirements and longer indexing times can still be useful if it reduces latency, increases diversity of results, and/or finds answers that are hard to find using the term-based index. A case in point is extreme sharding of Indri indices employed in IBM Watson QA system [104]. IBM Watson system crucially relied on the passage retrieval algorithm implemented in Indri [279]. However, for the complete 50 GB collection passage retrieval was too slow, sometimes taking longer than 100 seconds. Consequently, the complete index was split into small chunks each requiring approximately one GB of memory [104], which was much less than the amount of memory available on individual IBM servers [104, 111].

Although compressed "sparse" forward-indices may be larger than compressed term-based inverted indices, they are still reasonably small. For example, in our experiments (see Table 3.20), they can be compressed to 30-40% of their uncompressed size. We speculate that dense vectorial representations are likely to have higher memory requirements. Let us assume that we obtain dense document representations by averaging individual word embeddings. Because

the dimensionality of embeddings used in experiments is typically not less than 100 [238], each document would be represented by at least 100 *floating-point* values, which are likely hard to compress. However, often times, good results require even larger vectors. For example, Le and Mikolov experiment with 400-dimensional [176] document embeddings. Kiros et al. generate paragraph embeddings that have 1200 (or more) dimensions [161].

## 3.4 Summary

Most importantly, we have shown that on some data sets, a generic $k$-NN search method can be a replacement for term-based retrieval on top of an inverted file. We cannot yet conclude that $k$-NN search finds substantially different results compared to a classic term-based retrieval pipeline. Yet, we show that it is possible for $k$-NN search to be more efficient and equally effective.

In particular, for a *Stack Overflow* collection, $k$-NN search with SW-graph can be more efficient than the DAAT C++ baseline and equally effective to the fusion pipeline (see black-star runs in Panel 3.2c). For *Comprehensive* (see Panel 3.2a, the results are less impressive: the SW-graph run that is equally efficient to the DAAT C++ baseline is more effective than this baseline, but less effective than the fusion pipeline. For *WikiSQuAD*, unfortunately, all SW-graph runs that are more efficient than the DAAT C++ baseline are less effective than BM25 (see Panel 3.2e). It seems that the longer are the queries (see Table 3.4), the more impressive is performance of $k$-NN runs. However, this conjecture requires a more rigorous experiment to be confirmed.

Whenever NAPP and SW-graph runs are equally effective, SW-graph runs are always more efficient (although NAPP runs are sometimes close to SW-graph in terms of efficiency). Note that we cannot compare NAPP and SW-graph for a full range of values of an accuracy metric such as NDCG@20, because there are high-accuracy NAPP runs without SW-graph runs that match NAPP in accuracy.

Note that both the DAAT C++ baseline and our code keep indices in memory and do not employ any index compression. Hence, this comparison is equitable. Size of the $k$-NN index is larger than the size of the classic inverted term-based index, in particular, because we need to keep the forward index in memory. In a distributed setting and the NAPP method, the memory requirements for the forward index can be reduced by sharing a single forward index among instances (see § 3.3.4 for a more detailed discussion). Yet, this seems to be impossible for SW-graph (which is more efficient in our experiments).

We explore several approaches to construct an effective similarity, which are based on either dense or sparse representations. We find that dense representations are substantially less effective than BM25 (§ 3.2.3). In contrast, a combination of BM25 and IBM Model 1 scores trained on purely lexical representations is 18-29% in NDCG20 and 20-30% in ERR20 more effective than BM25 alone (see § 3.2.1. However, using linguistic features does not result in substantially better models (§ 3.2.2).

We also note that using a lot of training data is crucial to good performance of IBM Model 1. The fact that the amount of training data is quite important is not novel. In fact, Banko and Brill have famously shown that a simple model with more data can substantially outperform a better model with less data [23]. However, the degree of improvement by using millions of training pairs instead of dozens of thousands is quite remarkable (see Tables 3.9 and 3.10). To our knowledge,

this a novel observation with respect to performance of IBM Model 1 in the text-retrieval domain.

We conclude the summary with a list of practical insights learned from our extrinsic evaluations:

- The accuracy in the intrinsic evaluation is tightly coupled with an accuracy in the extrinsic evaluation: Thus, there is little hope of getting good extrinsic results without obtaining high $k$-NN recall.
- Furthermore, the accuracy and efficiency of approximate $k$-NN search decreases with $k$. Because we are likely to miss a substantial fraction of data points that are not especially close to the query, it may be infeasible to apply approximate $k$-NN search when $k$ is high. In other words, approximate $k$-NN search may not be applicable to the problem of retrieving a large number of candidate entries.
- In our experiments, we observe that there is—unsurprisingly—a difficult trade-off between accuracy and efficiency. Computation of the distance can be a major computational bottleneck. In particular, good results reported in Panel 3.2c would have been impossible to obtain without a novel trick of reducing computational complexity of Model 1 .
- Somewhat preliminarily, the longer are the queries (see Table 3.4), the more impressive is performance of $k$-NN search. Yet, a more rigorous verification of this conjecture remains for future work.

# Chapter 4

# Conclusion

In this concluding chapter we present a discussion of findings, hypotheses, and related open questions (§ 4.1), which is followed by a brief summary of thesis results (§ 4.2). We also discuss potential challenges facing a retrieval system designer.

## 4.1 Lessons Learned and Lessons to be Learned

In the following discussion we wear a hat of a practitioner tasked with a design/implementation of a top-$k$ retrieval system, which—as noted in § 2.1—is essentially a system for $k$-NN search. To produce a system that is reasonably accurate and efficient, this practitioner needs to make several fundamental design decisions listed in Table 4.1. The major decisions are related to choosing data representation, a type of a distance function, and a search algorithm/data structure.

Two vital components of $k$-NN search are data representation and a distance function that is computed over these representations. These components are closely related and the choice of the distance function is often motivated by the choice of the data representation. For reasons of convenience and efficiency, we employ mostly vectorial data which is either sparse or dense. In the case of sparse representations, the number of dimensions can be in the order of millions. However, in a single sparse vector most elements are zero. For example in the case of Wiki-sparse collection (§ 2.3.1.1), an average fraction of zero elements is nearly 99.9%. A representation can also have an arbitrarily complex structure, e.g., we can use strings or parse trees.

It is not yet clear to us if one should use sparse representations directly or, instead, convert them to dense vectors, which are compared using the Euclidean distance. Having dense representation of moderate dimensionality (dozens or hundreds of elements) with a simple Euclidean or cosine distance could be an ideal scenario for $k$-NN search for two main reasons:

- The Euclidean and cosine distance between dense vectors can be computed quite efficiently on modern hardware using SIMD instructions;
- Most research effort has been focused on the Euclidean distance: answering Euclidean $k$-NN queries can be done quite efficiently (see § 2.3.1).

However, we suppose that that coercion of non-metric data to the *low-dimensional* Euclidean space leads to a loss of information. A case in point are dense representations for text retrieval. Historically, text retrieval applications has been relying on sparse bag-of-words vectors, where

| Design problem | Notes |
|---|---|
| Data representation | Possible options:<br>• Dense vectors<br>• Sparse vectors<br>• Heterogeneous and/or complex data structures (e.g., trees) |
| Distance function type | Possible options:<br>• Euclidean<br>• Inner product/cosine distance<br>• Statistical divergence<br>• Premetric black-box distance |
| Cheaper inaccurate or slower accurate distances? | $k$-NN search using cheaper but less accurate function may be carried out more efficiently with higher $R_{knn}$@k. Using a more complex and computationally more expensive function might allow us to find relevant entries that are hard or impossible to retrieve using the cheaper baseline distance. |
| Usage pattern | • candidate generation (for production and prototyping)<br>• evidencing |
| Search algorithm and the data structure | Possible options:<br>• Brute-force search<br>• Term-based inverted files<br>• Clustering and/or quantization<br>• Pivoting methods<br>• Graph-based search (e.g., SW-graph)<br>• LSH<br>• Hierarchical space partitioning (a tree or a forest) |

Table 4.1: Fundamental top-$k$ retrieval design problems

vector values may incorporate, e.g., a scaled term frequency, an IDF, or their combination. In contrast, dense vectors are typically generated via artificial neural networks in an unsupervised or supervised manner [77, 161, 176, 206, 318] (older embedding-generation methods typically relied on PCA, see, e.g., [291]). In one scenario, document representations are generated directly by a neural network [161, 176]. In another scenario, we generate a separate dense vector for each word: A document representation would be a weighted average of individual word embeddings (one approach is to use IDFs as weights). Despite simplicity, average embeddings (possibly dimensionality reduced) are a tough baseline [8].

There are promising results related to dense representations. In particular, they are shown to be effective in text classification [161, 260], QA [144], and in many NLP tasks, notably in machine translation [21, 72]. Furthermore, a well-known DSSM model that re-ranks a short top-$k$ list produced by a search engine using the cosine similarity between averaged word embeddings has been shown to outperform BM25 [139].

However, as we learn from our experiments (§ 3.2.3 and [45]), performance of purely dense representations in ad hoc retrieval task can be quite poor compared to BM25. Mitra et al. [208] have come to the same conclusion. They further note that effectiveness of dense representations in many prior publications (in particular, in the paper that proposed DSSM [139]) is obtained in the so called "telescopic" setting, where the cosine similarity is used to re-rank a short list of candidates. We in turn note that the telescopic evaluation implicitly combines the cosine similarity signals with other (potentially stronger) signals. An implicit combination that linearly mix the BM25 score with the cosine similarity values achieves a similar effect. However, this is quite different from evaluating the effectiveness of the cosine similarity between dense representations alone.

Several neural ranking models for ad hoc text retrieval are substantially more effective than the cosine similarity between averaged word embeddings: The published gains over BM25 are as high as 60% [89, 318]. Potentially these models may result in effective $k$-NN solutions. However, these models do not reduce evaluation of similarity to computation of a distance between simple vectorial representations. Furthermore, if we use neural ranking models as black-bock distances, we will likely face computational challenges, because these models go well beyond computing simple functions—such as the cosine similarity—over dense representations.

For example, the model of Xiong et al. [318] involves computing $|q| \cdot |D|$ cosine similarities among query and term embedding vectors, where $|q|$ and $|D|$ are the number of terms in the query and the document, respectively. Xiong et al. [318] use 300-dimensional embeddings vectors. The widely precomputed Glove vectors [238] have a varying dimensionality, but the minimum one is 50.[1] Thus, computing the cosine similarity (or an inner product) between each pair of a query and document term vector represents a substantial cost. A similar issue arises when word embeddings are used in the framework of translation language modelling [119, 125, 331] and in the well-known Word Mover's distance [174]. Although some of these computations can be sped up using our efficiency trick as discussed in § 3.1.2.1, whether computational challenges concerned with the use of neural IR models can be overcome remains an open research question.

Averaged word embeddings can be useful in classification or when a retrieval problem can be recast as a classification problem. For example, Iyyer et al. [144] use averaged word embeddings

[1] See `https://nlp.stanford.edu/projects/glove/`.

as an input to a neural network classifier that selects an answer to a quiz ball question from a list of several thousand commonly occurring answers. However, this approach seems to require a large training set with several question-answer pairs for each answer. It is therefore not clear if it can be extended, e.g., for problems with a long tail distribution of answer types and rarely (if ever) repeating answers as in the case of, e.g., Jeopardy! [217].

Neural machine translation is another example of a classification task, where reasonable results can be achieved by "squishing" a short document (a sentence) into a dense vector. The translation can be in principle carried out by first encoding an input sentence into a dense vector of a fixed dimensionality using one recurrent neural network (RNN). Then an output sentence would be produced by a second RNN which uses only the input sentence dense vector (i.e., the second RNN does not have access to the original sentence words) [72]. Despite this, in a state-of-the-art machine translation system the decoding RNN has access to an input sequence via the attention mechanism [21]. In other words, we do not decode using merely a sentence produced by the encoding RNN. All in all, there seems to be good evidence that in the case of the natural language "squishing" long sequences of words into a single dense vector can be a suboptimal approach. However, we have a lot faith in *hybrid* representations, which combine, e.g., sparse TF×IDF vectors, sparse metadata and learned dense representations.

To conclude with the discussion of data representation, we note that in some cases we may have no luxury in choosing between dense and sparse representations because sparse representations are non-existent or impractical. For example, in a recent study Soska et al. [276] cluster software packages using rich file meta-data. They note that there is a long tail of software products where files lack consistent names and signatures. In other words, the same file (i.e., a file with the same SHA1 code) can be named quite differently in various installations or product versions. Keeping and comparing complete sets of file attributes is impractical. Instead, attribute sets are converted into dense vectors via count-min sketching [78]. Dense sketches are then compared using an $L_1$-normalized inner product.

A similar situation is in the image and video domains which lack "natural" sparse features such as words and characters. Instead, image search engines commonly rely on indexing of local and global image descriptors represented by dense vectors [16, 17, 24, 149, 150, 153]. Quite often images are represented and retrieved using bag-of-words features. However, such bag-of-words features are artificial constructs obtained by quantizing image/video descriptors [270].

It is also noteworthy that—similar to the text retrieval domain, where an inner product between sparse textual representations is a strong baseline—the Euclidean distance between local or global image descriptors is quite effective [17, 170]. There have been numerous attempts to devise a better distance function (see, e. g., [26, 168, 212]) as well as to integrate the distance function with the locality sensitive hashing scheme ([123, 211, 306]). One interesting attempt, is the work by Kulis and Grauman where they compare the non-metric kernelized LSH with the $L_2$-based retrieval [170]. Please, consider Figure 2 (from their paper) for the Photo Tourism data set. One can clearly see that for the same number of candidate entries a learned kernelized distance finds more relevant entries. For example, for a 70% recall it is sufficient to retrieve two hundred entries using the kernelized distance vs. three hundred entries using $L_2$. However, the difference in the number of candidates (for a given recall level) is small: It can be easily compensated by retrieving a slightly larger number of candidates and re-ranking them. In the text retrieval domain, Diaz obtained a similar result by applying pseudo-relevance feedback ranking only at a re-ranking step

[92]. This is quite similar to our experiments where using a more expressive distance function directly can possibly increase efficiency, yet it does not allow us to find substantially different results.

This brings up another fundamental question for the designer of the top-$k$ retrieval system. Which distance function should we use at a candidate generation phase: A cheap inexpensive distance or a more accurate but computationally more demanding? There are certainly scenarios where we have distance functions that are both cheap and accurate. Yet, in most cases, there is a trade-off between the accuracy and efficiency. We still believe that using a more complex and computationally more expensive function holds a potential of finding relevant entries that are hard or impossible to retrieve using the cheaper and less accurate distance. However, it has now become clear to us that this potential can be hard to realize.

First, efficient $k$-NN search is often impossible without trading accuracy for efficiency: From our evaluations we learn that retaining a good accuracy (which corresponds to 90% $k$-NN recall) requires the speed-up over the brute-force search to be in the range 100-300$\times$ (see Tables 3.17-3.19 and Figure 3.2). We also observe that applying approximate search gives us no free pass to efficiency, because there is a nearly monotonic relationship between the intrinsic and extrinsic effectiveness values such as $k$-NN recall and ERR@20 (or answer recall). Simply speaking, to retain 90% accuracy with respect to the extrinsic metric, we alos need to have $k$-NN recall close to 90% (which is clearly hard to do efficiently).

We hypothesize that this is a fundamental issue present in other domains such as image, video, or audio retrieval. Thus, if the distance function outperforms the baseline by a small margin, this effectiveness gain can be easily invalidated by the inaccuracy of the search procedure. In other words, the maximum achievable speed-up over the brute-force search still retaining high accuracy imposes a limit on computational complexity of the similarity. In particular, obtaining good result in this thesis was hardly possible without efficient computation of IBM Model 1 (see § 3.1.2 for more details).

Second, for a simpler, cheaper, and less accurate function it is often feasible to carry out $k$-NN search using much larger values of $k$ compared to the fancier, expensive, and more accurate distance. In particular, for sufficiently cheap distance functions it may be even feasible to carry out an exact brute force search. Therefore, it can be much easier to achieve higher recall by simply increasing the value of $k$. In contrast, we observe that the accuracy and/or efficiency of $k$-NN search decreases with $k$. It is, therefore, most useful when there is a sufficiently accurate similarity that we expect to generate a short list of high-quality candidates.

Having fewer top-$k$ candidates is typically associated with higher precision but lower recall. We still believe, however, that it is possible to achieve high recall for small $k$, but we have to be clever about the design of the distance function. Designing clever-yet-efficiently computable distances is a hard open research question. What is possible to achieve clearly depends a lot on the domain. We believe that the most promising are domains with rich meta-data and abundance of strong relevance signals. In particular, question answering seems to be a particularly promising domain with a variety of known syntactic and semantic clues. In addition to this thesis, there have been a number of attempts to incorporate these features into retrieval [36, 321]. However, this direction remains largely unexplored.

In addition to these strong known relevance signals, the QA domain has a number of parallel corpora. This makes it possible to design effective similarities using Model 1 or neural networks.

These similarities use parallel corpora to capture associations between query/question and document/answer terms. We think that existence of direct (e.g., expert annotations) or indirect (parallel corpora, user clicks, etc) supervision is crucial to success of such models. However, we are more skeptical about purely unsupervised approaches such as pseudo-relevance feedback. As we see from our experiments, the pseudo-relevance feedback where queries are expanded by terms from retrieved answers does not improve accuracy. An interesting alternative to consider is to do pseudo-relevance feedback on questions: Expansion term would come from questions similar to a given one or from answers provided to these similar questions. Yet we think that purely unsupervised approaches—even if effective on average—is destined to suffer from a poison pill problem [309]. In contrast, we believe that transfer learning can help sometimes when training data is lacking or small. For example, in our experiments we observe that a translation model trained on *Comprehensive* is applicable to *WikiSQuAD* (see Table 3.10).

Although in this work we focus on the scenario of replacing a cheaper baseline distance function with an accurate $k$-NN search, there are scenarios where approximate $k$-NN search can be complementary. First, there is a trend in using retrieval algorithms to provide additional evidence for a statistical classifier rather than to generate a *primary* list of candidates for further re-ranking [124, 152, 216]. Second, we hypothesize that efficient black-box retrieval methods would be extremely useful for experimentation, because they place fewer restrictions on the type of similarities with which a data scientist can experiment. After the data scientist done with experimentation, software engineers would optimize pipeline for efficiency possibly using the filter-and-refine framework.

The last but not the least design question is a choice of suitable retrieval algorithms and data structures. For domains with purely dense representations, it is often natural to compare dense vectors using the Euclidean distance. Many methods work well in this case. In particular, we know that proximity-graph search methods deliver great performance. In § 2.3 we demonstrate this for SW-graph: There are also a number of proximity-graph methods proposed that offer further improvements for *symmetric* distances [117, 130, 184, 196]. We, in turn, show that SW-graph works well for a number of *non-symmetric* premetric distances, which are non-negative distance functions with zero dissimilarity and identity of indiscernibles (see § 2.3.2.4). We nevertheless do not truly know the limits of applicability of the graph-based search algorithms. Obtaining a better understanding by developing theory and carrying out additional experiments remains an important direction of future work.

Another disadvantage of graph-based retrieval algorithms is their high memory consumption. As we show in § 3.3.4, memory requirements of graph-based search algorithms are higher than memory requirements of the classic methods (term-based inverted files). Clearly, there should be some research effort to reduce memory footprint. However, we speculate that for text retrieval, these index sizes are still quite reasonable. The situation is different in the image and video domain in part because a single image is often represented by multiple local descriptors each of which is a vector of moderate dimensionality (e.g., uncompressed SIFT descriptors require 128 bytes) [99]. For this reason, image retrieval has been heavily relying on lossy compression of descriptors achieved via quantization. As mentioned previously, quantization produces a bag-of-words style features that are indexed via inverted files. However, a recent trend is to combine quantization with graph-based retrieval [24, 99].

In domains with purely sparse bag-of-words representations, the similarity is typically com-

puted using the inner product or the cosine similarity. Despite the inner product and the cosine similarity search can be reduced to the Euclidean search (see Exposition 2.5), in practice the sparse bag-of-words retrieval landscape has been dominated by inverted files. An enormous effort was put into designing optimal computation strategies. In particular, a number of methods are designed to prune unpromising parts of an inverted file without full evaluation, sometimes in an unsafe manner [51, 94, 114, 186, 198].

Thanks to the sparsity of query-document matrix, inverted file based solutions can be quite efficient. However, they have their limits too. In our opinion, they are most useful when:

- the similarity is decomposable into the sum of term-specific similarity values;
- term-document matrix is sparse;
- queries are short.

When these conditions are not met, it is not easy to prune away documents without fully evaluating the query-document similarity score. Pruning is especially hard when the similarity does not decompose into the sum of term-specific similarity values, because each pair of a query and a document term can potentially contribute to the similarity score (see, e.g., the Eq. (3.3) for IBM Model 1 in § 3.1.1.2). However as shown in this thesis, generic methods of $k$-NN search can work in the case of the non-decomposable similarity that is a combination of BM25 and BM25+Model1 scores (see § 3.3).

Furthermore, pruning may be difficult even for a decomposable inner-product similarity. In particular, Crane et al. [79] experiment with three state-of-the art approaches to top-$k$ score evaluation strategies for inner-product similarities: WAND [51], block-max WAND (BMW) [94], and JASS [186]. These strategies are designed to prune unpromising parts of an inverted file without full evaluation, sometimes in an unsafe manner.

Let us examine retrieval times posted in Figure 6 (in [79]), which are computed for approximate pruning strategies that do not guarantee accurate retrieval. These are computed for a ClueWeb12-B13 collection containing 52 million Web documents.[2] First, we can see that retrieval time increases as the number of query terms increases and in some cases this increase is super-linear [79].

Furthermore, top-$k$ evaluation times often increase as we increase $k$. In particular, for $k = 1000$ and 10-term queries, the average retrieval times for WAND and BMW is nearly one second, whereas for $k = 10$, these two methods answer queries in roughly 100 milliseconds. Performance of approximate JASS is nearly unaffected by the choice of $k$ and for long queries JASS is more efficient than WAND and BMW. However, from Figure 4c (in [79]) we can see that JASS is 5-15% less accurate compared to WAND and BMW on ClueWeb12-B13. In fact, it is always slower at a given level of accuracy. To sum up our observations, *long* queries entail substantial latencies even with the state-of-the-art *approximate* pruning strategies for inner-product similarities.

Another potential issue with the classic term-based retrieval methods is their sensitivity to the query length. As we can see from Tables 3.17, 3.18, and 3.19), expanding the query makes retrieval 2-10× slower (see also a note on p. 140). Because adding rare terms has much smaller effect on overall performance, retrieval times do not linearly depend on the size of the query. For

[2] `https://lemurproject.org/clueweb12/specs.php`

example, we experimented with expanding the query using Model 1 translation tables: expanding one term with five synonyms has nearly the same cost as the expansion with one thousand despite not using any pruning algorithms such as WAND [51] and Block-Max WAND (BMW) [94]. Query expansion is nevertheless a substantial overhead.

Although we do not directly evaluate the effect of the query length on performance, we believe that performance of generic methods for $k$-NN search such as NAPP and SW-graph is much less affected by the query length: Our best results are obtained for the *Stack Overflow* collection, where we have the longest queries. We hypothesize that this happens because the employed methods for $k$-NN search implicitly rely on matching relatively small pieces of query and document vectors. This can be seen as a variant of pattern partitioning, which is a recurrent topic in the filed of approximate matching [41, 50, 129, 231].

In a nutshell, given two similar objects, we expect parts of these objects to have a much higher degree of similarity than complete objects. The "larger" are the objects to be compared, the easier it is to find these highly similar object parts. Therefore, although the increase in the length of queries and documents does lead to increase in the distance computation time, this increase is typically offset by improved pruning effectiveness, which, in turn, is a consequence of object part comparison being more informative.

For example, in the case of the pivoting method NAPP, the longer are the documents and queries, the larger is the overlap among non-zero dimensions in sparse query and document vectors. It is possible to further increase the amount of "overlap" by using the distance function that produces non-zero cross-dimensional similarity scores. For example, in the case of a simple BM25 model, the overlap is non-empty only if the pivot, the query, and the answer have at least one common term. In contrast, for the model BM25+Model 1—used extensively in our experiments in § 3—information regarding proximity of answers and queries may be obtained if pivots, queries, and answers share only related but not necessarily identical terms.

We also note that retrieval times of inverted file based solutions grows linearly with the collection size. However, for SW-graphs (and other graph-based algorithms for $k$-NN search) there seems to be a sub-linear dependency of the retrieval time on the collection size. In other words, empirically we observe that the relative efficiency of these methods increases with the collection size, but it is like to remain the same for term-based inverted files. Given that there is a relatively small difference between the classic approach and graph-based retrieval in this work, we expect graph-based retrieval to be at least as efficient as term-based inverted files for collections that are $10\times$ larger (than collections used in this thesis). Sublinear retrieval times of graph-based algorithm, however, comes at a price of a super-linear dependency in the indexing time. Therefore, optimizing the indexing stage of graph-based retrieval methods is an open research question.

To conclude the efficiency section, we note that there is a need for more definitive comparison between the classic retrieval methods and methods of generic $k$-NN search. First, our DAAT baseline may be insufficiently strong. Somewhat preliminarily, while preparing the final version of this manuscript, we have also compared against NMSLIB implementation with the following methods WAND [51] and BMW [94] (provided by David Novak). WAND was about 15% more efficient, but BMW was actually about $2\times$ slower. A more definitive comparison using third-party implementations of the pruning approaches remains for future work. In this comparison one should also study the effect of query length with more care.

## 4.2 Summary

In this thesis we advance state-of-the-art of the non-metric $k$-NN search by carrying out an extensive empirical evaluation (both extrinsic and intrinsic) of generic methods for $k$-NN search, thus, contributing to establishing a collection of strong benchmarks for data sets with generic distances.

We make a step towards replacing/complementing classic term-based retrieval with a generic $k$-NN search algorithm. To this end we use a similarity function that takes into account subtle term associations, which are learned from a parallel monolingual corpus. An *exact* brute-force $k$-NN search using this similarity function is quite slow, but an *approximate* search algorithm can be 100-300 times faster at the expense of only a small loss in accuracy (10%).

On *Stack Overflow*, a retrieval pipeline using an approximate $k$-NN search is comparably accurate to the Lucene-based fusion pipeline and is twice as efficient as the C++ DAAT baseline. We note, however, that it is necessary to compare our methods against more recent ranking algorithms [51, 94, 114, 186, 198].

In § 4.1, we discuss fundamental questions facing a designer of a top-$k$ retrieval system as well as related directions for future research.

A detailed list of thesis contributions is given in § 1.1.

In conclusion, we note that nearly all the code used in this thesis is available on-line:

- `https://github.com/oaqa/knn4qa/tree/bigger_reruns`
- `https://github.com/searchivarius/nmslib/tree/nmslib4a_bigger_reruns`

# Bibliography


[1] Amirali Abdullah, John Moeller, and Suresh Venkatasubramanian. Approximate bregman near neighbors in sublinear time: beyond the triangle inequality. In Tamal K. Dey and Sue Whitesides, editors, *Symposuim on Computational Geometry 2012, SoCG '12, Chapel Hill, NC, USA, June 17-20, 2012*, pages 31–40. ACM, 2012. doi: 10.1145/2261250.2261255. 15, 24, 67, 97

[2] Giuseppe Amato and Pasquale Savino. Approximate similarity search in metric spaces using inverted files. In *3rd International ICST Conference on Scalable Information Systems, INFOSCALE 2008, Vico Equense, Italy, June 4-6, 2008*, page 28, 2008. doi: 10.4108/ICST.INFOSCALE2008.3486. 42, 55

[3] Giuseppe Amato, Claudio Gennaro, and Pasquale Savino. MI-File: using inverted files for scalable approximate similarity search. *Multimedia Tools Appl.*, 71(3):1333–1362, 2014. doi: 10.1007/s11042-012-1271-1. 42, 54, 55

[4] Laurent Amsaleg, Oussama Chelly, Teddy Furon, Stéphane Girard, Michael E. Houle, Ken-ichi Kawarabayashi, and Michael Nett. Estimating local intrinsic dimensionality. In Longbing Cao, Chengqi Zhang, Thorsten Joachims, Geoffrey I. Webb, Dragos D. Margineantu, and Graham Williams, editors, *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 29–38. ACM, 2015. doi: 10.1145/2783258.2783405. 15

[5] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya P. Razenshteyn, and Ludwig Schmidt. Practical and optimal LSH for angular distance. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1225–1233, 2015. 20, 35, 113

[6] Vo Ngoc Anh and Alistair Moffat. Pruned query evaluation using pre-computed impacts. In Efthimis N. Efthimiadis, Susan T. Dumais, David Hawking, and Kalervo Järvelin, editors, *SIGIR 2006: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, Washington, USA, August 6-11, 2006*, pages 372–379. ACM, 2006. doi: 10.1145/1148170.1148235. 20

[7] Kazuo Aoyama, Kazumi Saito, Hiroshi Sawada, and Naonori Ueda. Fast approximate similarity search based on degree-reduced neighborhood graphs. In Chid Apté, Joydeep Ghosh, and Padhraic Smyth, editors, *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24,*

*2011*, pages 1055–1063. ACM, 2011. doi: 10.1145/2020408.2020576. 37

[8] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. In *Proceedings of the ICLR 2017 conference*, 2016. 155

[9] Sunil Arya and David M. Mount. Algorithms for fast vector quantization. In *Proceedings of the IEEE Data Compression Conference, DCC 1993, Snowbird, Utah, March 30 - April 1, 1993.*, pages 381–390, 1993. doi: 10.1109/DCC.1993.253111. 32

[10] Sunil Arya and David M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas.*, pages 271–280, 1993. 37

[11] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, November 1998. ISSN 0004-5411. doi: 10.1145/293347.293348. 33

[12] Vassilis Athitsos, Jonathan Alon, Stan Sclaroff, and George Kollios. Boostmap: A method for efficient approximate similarity rankings. In *2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2004), with CD-ROM, 27 June - 2 July 2004, Washington, DC, USA*, pages 268–275, 2004. doi: 10.1109/CVPR.2004.52. 23, 25

[13] Vassilis Athitsos, Michalis Potamias, Panagiotis Papapetrou, and George Kollios. Nearest neighbor retrieval using distance-based hashing. In Gustavo Alonso, José A. Blakeley, and Arbee L. P. Chen, editors, *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*, pages 327–336. IEEE Computer Society, 2008. doi: 10.1109/ICDE.2008.4497441. 35

[14] Giuseppe Attardi. WikiExtractor: A tool for extracting plain text from Wikipedia dumps. `https://github.com/attardi/wikiextractor`, [Last Checked Jan 2017], 2015. 107

[15] Artem Babenko and Victor S. Lempitsky. The inverted multi-index. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 3069–3076. IEEE Computer Society, 2012. doi: 10.1109/CVPR.2012.6248038. 17, 29, 31, 57

[16] Artem Babenko and Victor S. Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2055–2063. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.226. 156

[17] Artem Babenko, Anton Slesarev, Alexander Chigorin, and Victor S. Lempitsky. Neural codes for image retrieval. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, pages 584–599, 2014. doi: 10.1007/978-3-319-10590-1_38. URL `https://doi.org/10.1007/978-3-319-10590-1_38`. 156

[18] Ricardo A. Baeza-Yates and Gonzalo Navarro. Block addressing indices for approximate text retrieval. *JASIS*, 51(1):69–82, 2000. doi: 10.1002/(SICI)1097-4571(2000)51:1⟨69::

AID-ASI10⟩3.0.CO;2-C. 145

[19] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval - the concepts and technology behind search, Second edition*. Pearson Education Ltd., Harlow, England, 2011. ISBN 978-0-321-41691-9. 14, 19, 101

[20] Ricardo A. Baeza-Yates, Walter Cunto, Udi Manber, and Sun Wu. Proximity matching using fixed-queries trees. In *Combinatorial Pattern Matching, 5th Annual Symposium, CPM 94, Asilomar, California, USA, June 5-8, 1994, Proceedings*, pages 198–212, 1994. doi: 10.1007/3-540-58094-8_18. 53

[21] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. 155, 156

[22] Arindam Banerjee, Srujana Merugu, Inderjit S. Dhillon, and Joydeep Ghosh. Clustering with bregman divergences. *Journal of Machine Learning Research*, 6:1705–1749, 2005. 17

[23] Michele Banko and Eric Brill. Scaling to very very large corpora for natural language disambiguation. In *Association for Computational Linguistic, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference, July 9-11, 2001, Toulouse, France.*, pages 26–33. Morgan Kaufmann Publishers, 2001. 150

[24] Dmitry Baranchuk, Artem Babenko, and Yury Malkov. Revisiting the inverted indices for billion-scale approximate nearest neighbors. *CoRR*, abs/1802.02422, 2018. URL `http://arxiv.org/abs/1802.02422`. 156, 158

[25] Petr Baudis and Jan Sedivý. Modeling of the question answering task in the yodaqa system. In *Experimental IR Meets Multilinguality, Multimodality, and Interaction - 6th International Conference of the CLEF Association, CLEF 2015, Toulouse, France, September 8-11, 2015, Proceedings*, pages 222–228, 2015. doi: 10.1007/978-3-319-24027-5_20. 127

[26] Christian Beecks. *Distance based similarity models for content based multimedia retrieval*. PhD thesis, RWTH Aachen University, 2013. URL `http://dme.rwth-aachen.de/de/publications/3162`. 57, 156

[27] Christian Beecks, Merih Seran Uysal, and Thomas Seidl. Signature quadratic form distance. In *Proceedings of the ACM International Conference on Image and Video Retrieval*, CIVR '10, pages 438–445, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0117-6. 56, 57

[28] Aurélien Bellet, Amaury Habrard, and Marc Sebban. A survey on metric learning for feature vectors and structured data. *CoRR*, abs/1306.6709, 2013. 1

[29] Richard Bellman. *Adaptive control processes: a guided tour*. Princeton University Press, Princeton, New Jersey, USA, 1961. 2

[30] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975. doi: 10.1145/361002.361007. 27, 30, 31

[31] Jon Louis Bentley and Andrew Chi-Chih Yao. An almost optimal algorithm for unbounded searching. *Inf. Process. Lett.*, 5(3):82–87, 1976. doi: 10.1016/0020-0190(76)90071-5. 118

[32] Adam L. Berger, Rich Caruana, David Cohn, Dayne Freitag, and Vibhu O. Mittal. Bridging the lexical chasm: statistical approaches to answer-finding. In *SIGIR 2000: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development*

*in Information Retrieval, July 24-28, 2000, Athens, Greece*, pages 192–199, 2000. doi: 10.1145/345508.345576. 102, 104

[33] Erik Bernhardsson. Music discovery at spotify. `https://www.slideshare.net/erikbern/music-recommendations-mlconf-2014` [Last Checked March 2017], 2014. Presentation about Spotify's music discovery system at MLConf NYC 2014. 102

[34] Kevin S. Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In Catriel Beeri and Peter Buneman, editors, *Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings.*, volume 1540 of *Lecture Notes in Computer Science*, pages 217–235. Springer, 1999. doi: 10.1007/3-540-49257-7_15. 2, 18, 33, 44

[35] Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In William W. Cohen and Andrew Moore, editors, *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, volume 148 of *ACM International Conference Proceeding Series*, pages 97–104. ACM, 2006. doi: 10.1145/1143844.1143857. 99

[36] Matthew W. Bilotti, Paul Ogilvie, Jamie Callan, and Eric Nyberg. Structured retrieval for question answering. In Wessel Kraaij, Arjen P. de Vries, Charles L. A. Clarke, Norbert Fuhr, and Noriko Kando, editors, *SIGIR 2007: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Amsterdam, The Netherlands, July 23-27, 2007*, pages 351–358. ACM, 2007. doi: 10.1145/1277741.1277802. 157

[37] Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In Doheon Lee, Mario Schkolnick, Foster J. Provost, and Ramakrishnan Srikant, editors, *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, CA, USA, August 26-29, 2001*, pages 245–250. ACM, 2001. 16, 26, 64

[38] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003. 50, 56, 58, 68

[39] Paolo Bolettieri, Andrea Esuli, Fabrizio Falchi, Claudio Lucchese, Raffaele Perego, Tommaso Piccioli, and Fausto Rabitti. Cophir: a test collection for content-based image retrieval. *CoRR*, abs/0905.4627, 2009. 56, 57

[40] C. E. Bonferroni. Teoria statistica delle classi e calcolo delle probabilità. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze*, 8:3–62, 1936. 133

[41] Leonid Boytsov. Indexing methods for approximate dictionary searching: Comparative analysis. *ACM Journal of Experimental Algorithmics*, 16(1), 2011. doi: 10.1145/1963190.1963191. 21, 160

[42] Leonid Boytsov and Bilegsaikhan Naidan. Engineering efficient and effective non-metric space library. In Nieves R. Brisaboa, Oscar Pedreira, and Pavel Zezula, editors, *Similarity Search and Applications - 6th International Conference, SISAP 2013, A Coruña, Spain, October 2-4, 2013, Proceedings*, volume 8199 of *Lecture Notes in Computer Science*, pages

280–293. Springer, 2013. doi: 10.1007/978-3-642-41062-8_28. 59, 62, 113

[43] Leonid Boytsov and Bilegsaikhan Naidan. Learning to prune in metric and non-metric spaces. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 1574–1582, 2013. 1, 18, 21, 35, 48, 49, 50, 52, 54, 55, 59, 77

[44] Leonid Boytsov, Anna Belova, and Peter Westfall. Deciding on an adjustment for multiplicity in IR experiments. In *The 36th International ACM SIGIR conference on research and development in Information Retrieval, SIGIR '13, Dublin, Ireland - July 28 - August 01, 2013*, pages 403–412, 2013. doi: 10.1145/2484028.2484034. 106

[45] Leonid Boytsov, David Novak, Yury Malkov, and Eric Nyberg. Off the beaten path: Let's replace term-based retrieval with k-nn search. volume abs/1610.10001, 2016. v, 101, 102, 104, 111, 113, 120, 130, 133, 155

[46] Tolga Bozkaya and Z. Meral Özsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Trans. Database Syst.*, 24(3):361–404, 1999. doi: 10.1145/328939.328959. 32, 99

[47] L.M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7(3):200 – 217, 1967. ISSN 0041-5553. doi: http://dx.doi.org/10.1016/0041-5553(67)90040-7. 16

[48] Sergey Brin. Near neighbor search in large metric spaces. In Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, editors, *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland.*, pages 574–584. Morgan Kaufmann, 1995. 31

[49] A. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, SEQUENCES '97, pages 21–, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-8132-2. 34

[50] Andrei Z. Broder. Identifying and filtering near-duplicate documents. In Raffaele Giancarlo and David Sankoff, editors, *Combinatorial Pattern Matching, 11th Annual Symposium, CPM 2000, Montreal, Canada, June 21-23, 2000, Proceedings*, volume 1848 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2000. doi: 10.1007/3-540-45123-4_1. 21, 102, 160

[51] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Y. Zien. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management, New Orleans, Louisiana, USA, November 2-8, 2003*, pages 426–434. ACM, 2003. doi: 10.1145/956863.956944. 5, 159, 160, 161

[52] Georgios-Ioannis Brokos, Prodromos Malakasiotis, and Ion Androutsopoulos. Using centroids of word embeddings and word mover's distance for biomedical document retrieval in question answering. *CoRR*, abs/1608.03905, 2016. 104

[53] Peter F. Brown, Stephen Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993. 14, 104, 115

[54] Cristian Bustos, Gonzalo Navarro, Nora Reyes, and Rodrigo Paredes. An empirical evaluation of intrinsic dimension estimators. In Giuseppe Amato, Richard C. H. Connor, Fabrizio Falchi, and Claudio Gennaro, editors, *Similarity Search and Applications - 8th International Conference, SISAP 2015, Glasgow, UK, October 12-14, 2015, Proceedings*, volume 9371 of *Lecture Notes in Computer Science*, pages 125–137. Springer, 2015. doi: 10.1007/978-3-319-25087-8_12. 15

[55] Stefan Büttcher, Charles L. A. Clarke, and Gordon V. Cormack. *Information Retrieval - Implementing and Evaluating Search Engines*. MIT Press, 2010. ISBN 978-0-262-02651-2. 114

[56] CJ Carey, Yuan Tang, et al. metric-learn: : Metric Learning in Python. `http://all-umass.github.io/metric-learn/` [Last Checked June 2017], 2013–. 75

[57] Claudio Carpineto and Giovanni Romano. A survey of automatic query expansion in information retrieval. *ACM Comput. Surv.*, 44(1):1, 2012. doi: 10.1145/2071389.2071390. 3

[58] Lawrence Cayton. Fast nearest neighbor retrieval for Bregman divergences. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, volume 307 of *ACM International Conference Proceeding Series*, pages 112–119. ACM, 2008. doi: 10.1145/1390156.1390171. 14, 17, 18, 21, 32, 33, 44, 50, 54, 58

[59] Lawrence Cayton. *Bregman Proximity Search*. PhD thesis, University of California, San Diego, 2009. URL `http://escholarship.org/uc/item/3q28g2vj`. 30, 32

[60] Lawrence Cayton and Sanjoy Dasgupta. A learning framework for nearest neighbor search. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 233–240, 2007. 44

[61] Olivier Chapelle, Donald Metlzer, Ya Zhang, and Pierre Grinspan. Expected reciprocal rank for graded relevance. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009*, pages 621–630, 2009. doi: 10.1145/1645953.1646033. 13, 103, 111

[62] Moses Charikar. Similarity estimation techniques from rounding algorithms. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 380–388. ACM, 2002. doi: 10.1145/509907.509965. 34, 35

[63] Edgar Chávez and Gonzalo Navarro. Probabilistic proximity search: Fighting the curse of dimensionality in metric spaces. *Inf. Process. Lett.*, 85(1):39–46, 2003. doi: 10.1016/S0020-0190(02)00344-7. 33, 51

[64] Edgar Chávez and Gonzalo Navarro. A compact space decomposition for effective metric

indexing. *Pattern Recognition Letters*, 26(9):1363–1376, 2005. doi: 10.1016/j.patrec.2004.11.014. 31, 122

[65] Edgar Chávez, Gonzalo Navarro, Ricardo A. Baeza-Yates, and José L. Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001. doi: 10.1145/502807.502808. 1, 2, 21, 31

[66] Edgar Chávez, Karina Figueroa, and Gonzalo Navarro. Proximity searching in high dimensional spaces with a proximity preserving order. In Alexander F. Gelbukh, Alvaro de Albornoz, and Hugo Terashima-Marín, editors, *MICAI 2005: Advances in Artificial Intelligence, 4th Mexican International Conference on Artificial Intelligence, Monterrey, Mexico, November 14-18, 2005, Proceedings*, volume 3789 of *Lecture Notes in Computer Science*, pages 405–414. Springer, 2005. doi: 10.1007/11579427_41. 54

[67] Edgar Chávez, Karina Figueroa, and Gonzalo Navarro. Effective proximity retrieval by ordering permutations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(9):1647–1658, 2008. doi: 10.1109/TPAMI.2007.70815. 40, 41, 54, 55

[68] Edgar Chávez, Mario Graff, Gonzalo Navarro, and Eric Sadit Téllez. Near neighbor searching with K nearest references. *Inf. Syst.*, 51:43–61, 2015. doi: 10.1016/j.is.2015.02.001. 44

[69] Edgar Chávez, Mario Graff, Gonzalo Navarro, and Eric Sadit Téllez. Near neighbor searching with K nearest references. *Inf. Syst.*, 51:43–61, 2015. doi: 10.1016/j.is.2015.02.001. 55, 60, 61, 67

[70] Gal Chechik, Varun Sharma, Uri Shalit, and Samy Bengio. Large scale online learning of image similarity through ranking. *Journal of Machine Learning Research*, 11:1109–1135, 2010. doi: 10.1145/1756006.1756042. 1

[71] Lei Chen and Xiang Lian. Efficient similarity search in nonmetric spaces with local constant embedding. *IEEE Trans. Knowl. Data Eng.*, 20(3):321–336, 2008. doi: 10.1109/TKDE.2007.190700. 21, 22

[72] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734. ACL, 2014. 155, 156

[73] Jinho D. Choi and Andrew McCallum. Transition-based dependency parsing with selectional branching. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 1: Long Papers*, pages 1052–1062, 2013. 126

[74] André-Louis Cholesky. Sur la résolution numérique des systèmes d'équations linéaires, 1910. URL `https://sabix.revues.org/529`. 24

[75] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively,

with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*, pages 539–546, 2005. doi: 10.1109/CVPR.2005.202. 24, 26

[76] Massimiliano Ciaramita and Yasemin Altun. Broad-coverage sense disambiguation and information extraction with a supersense sequence tagger. In *EMNLP 2007, Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing, 22-23 July 2006, Sydney, Australia*, pages 594–602, 2006.

[77] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011. 130, 155

[78] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005. doi: 10.1016/j.jalgor.2003.12.001. 156

[79] Matt Crane, J. Shane Culpepper, Jimmy J. Lin, Joel Mackenzie, and Andrew Trotman. A comparison of document-at-a-time and score-at-a-time query evaluation. In Maarten de Rijke, Milad Shokouhi, Andrew Tomkins, and Min Zhang, editors, *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, WSDM 2017, Cambridge, United Kingdom, February 6-10, 2017*, pages 201–210. ACM, 2017. 159

[80] W. Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines - Information Retrieval in Practice*. Pearson Education, 2009. ISBN 978-0-13-136489-9. 114, 117, 118

[81] Joachim Daiber, Max Jakob, Chris Hokamp, and Pablo N. Mendes. Improving efficiency and accuracy in multilingual entity extraction. In *I-SEMANTICS 2013 - 9th International Conference on Semantic Systems, ISEM '13, Graz, Austria, September 4-6, 2013*, pages 121–124, 2013. doi: 10.1145/2506182.2506198. 128, 143

[82] Sanjoy Dasgupta. Experiments with random projection. In Craig Boutilier and Moisés Goldszmidt, editors, *UAI '00: Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence, Stanford University, Stanford, California, USA, June 30 - July 3, 2000*, pages 143–151. Morgan Kaufmann, 2000. 26

[83] Sanjoy Dasgupta and Yoav Freund. Random projection trees and low dimensional manifolds. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 537–546. ACM, 2008. doi: 10.1145/1374376.1374452. 16, 26

[84] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In Jack Snoeyink and Jean-Daniel Boissonnat, editors, *Proceedings of the 20th ACM Symposium on Computational Geometry, Brooklyn, New York, USA, June 8-11, 2004*, pages 253–262. ACM, 2004. doi: 10.1145/997817.997857. 16, 25, 35

[85] John Daugman. How iris recognition works. *IEEE Trans. Circuits Syst. Video Techn.*, 14 (1):21–30, 2004. doi: 10.1109/TCSVT.2003.818350. 18

[86] John Daugman and Cathryn Downing. Epigenetic randomness, complexity and singularity

of human iris patterns. *Proceedings of the Royal Society of London B: Biological Sciences*, 268(1477):1737–1740, 2001. ISSN 0962-8452. doi: 10.1098/rspb.2001.1696. 18

[87] Jason V. Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S. Dhillon. Information-theoretic metric learning. In Zoubin Ghahramani, editor, *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, volume 227 of *ACM International Conference Proceeding Series*, pages 209–216. ACM, 2007. doi: 10.1145/1273496.1273523. 73

[88] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *JASIS*, 41(6):391–407, 1990. doi: 10.1002/(SICI)1097-4571(199009)41:6⟨391::AID-ASI1⟩3.0.CO;2-9. 102

[89] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W. Bruce Croft. Neural ranking models with weak supervision. *CoRR*, abs/1704.08803, 2017. 155

[90] Michel Marie Deza and Elena Deza. *Encyclopedia of Distances*. Springer Berlin Heidelberg, 2009. doi: 10.1007/978-3-642-00234-2_1. 13

[91] Persi Diaconis. Group representations in probability and statistics. *Lecture Notes-Monograph Series*, pages i–192, 1988. 40

[92] Fernando Diaz. Condensed list relevance models. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval, ICTIR 2015, Northampton, Massachusetts, USA, September 27-30, 2015*, pages 313–316, 2015. doi: 10.1145/2808194.2809491. 157

[93] Constantinos Dimopoulos, Sergey Nepomnyachiy, and Torsten Suel. Optimizing top-k document retrieval strategies for block-max indexes. In Stefano Leonardi, Alessandro Panconesi, Paolo Ferragina, and Aristides Gionis, editors, *Sixth ACM International Conference on Web Search and Data Mining, WSDM 2013, Rome, Italy, February 4-8, 2013*, pages 113–122. ACM, 2013. doi: 10.1145/2433396.2433412. 20

[94] Shuai Ding and Torsten Suel. Faster top-k document retrieval using block-max indexes. In Wei-Ying Ma, Jian-Yun Nie, Ricardo A. Baeza-Yates, Tat-Seng Chua, and W. Bruce Croft, editors, *Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2011, Beijing, China, July 25-29, 2011*, pages 993–1002. ACM, 2011. doi: 10.1145/2009916.2010048. 5, 102, 159, 160, 161

[95] Wei Dong. kgraph: A library for k-nearest neighbor search. `https://github.com/aaalgo/kgraph` [Last Checked April 2018]. 37, 39

[96] Wei Dong, Zhe Wang, William Josephson, Moses Charikar, and Kai Li. Modeling LSH for performance tuning. In James G. Shanahan, Sihem Amer-Yahia, Ioana Manolescu, Yi Zhang, David A. Evans, Aleksander Kolcz, Key-Sun Choi, and Abdur Chowdhury, editors, *Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM 2008, Napa Valley, California, USA, October 26-30, 2008*, pages 669–678. ACM, 2008. doi: 10.1145/1458082.1458172. 35, 113

[97] Wei Dong, Zhe Wang, William Josephson, Moses Charikar, and Kai Li. Modeling LSH for performance tuning. In James G. Shanahan, Sihem Amer-Yahia, Ioana Manolescu,

Yi Zhang, David A. Evans, Aleksander Kolcz, Key-Sun Choi, and Abdur Chowdhury, editors, *Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM 2008, Napa Valley, California, USA, October 26-30, 2008*, pages 669–678. ACM, 2008. doi: 10.1145/1458082.1458172. 59

[98] Wei Dong, Moses Charikar, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011*, pages 577–586. ACM, 2011. doi: 10.1145/1963405.1963487. 36, 37, 58, 59

[99] Matthijs Douze, Alexandre Sablayrolles, and Hervé Jégou. Link and code: Fast indexing with graphs and compact regression codes. *CoRR*, abs/1804.09996, 2018. URL `http://arxiv.org/abs/1804.09996`. 158

[100] Olive Jean Dunn. Multiple comparisons among means. *Journal of the American Statistical Association*, 56(293):52–64, 1961. doi: 10.1080/01621459.1961.10482090. 133

[101] Abdessamad Echihabi and Daniel Marcu. A noisy-channel approach to question answering. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, 7-12 July 2003, Sapporo Convention Center, Sapporo, Japan.*, pages 16–23, 2003. 104

[102] Richard Eckart de Castilho and Iryna Gurevych. A broad-coverage collection of portable nlp components for building shareable analysis pipelines. In *Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT*, pages 1–11, Dublin, Ireland, August 2014. ACL and Dublin City University. 126

[103] Dominik Maria Endres and Johannes E. Schindelin. A new metric for probability distributions. *IEEE Trans. Information Theory*, 49(7):1858–1860, 2003. doi: 10.1109/TIT.2003.813506. 11, 67

[104] Edward A. Epstein, Marshall I. Schor, Bhavani Iyer, Adam Lally, Eric W. Brown, and Jaroslaw Cwiklik. Making watson fast. *IBM Journal of Research and Development*, 56(3): 15, 2012. doi: 10.1147/JRD.2012.2188761. 149

[105] Andrea Esuli. PP-index: Using permutation prefixes for efficient and scalable approximate similarity search. *Proceedings of LSDS-IR*, i:1–48, 2009. 42, 55

[106] Ronald Fagin, Ravi Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In Alon Y. Halevy, Zachary G. Ives, and AnHai Doan, editors, *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, pages 301–312. ACM, 2003. doi: 10.1145/872757.872795. 41, 61

[107] Christos Faloutsos and King-Ip Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995.*, pages 163–174. ACM Press, 1995. doi: 10.1145/223784.223812. 25, 26

[108] András Faragó, Tamás Linder, and Gábor Lugosi. Fast nearest-neighbor search in dis-

similarity spaces. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(9):957–962, 1993. doi: 10.1109/34.232083. 15, 24, 40

[109] Manaal Faruqui, Jesse Dodge, Sujay Kumar Jauhar, Chris Dyer, Eduard H. Hovy, and Noah A. Smith. Retrofitting word vectors to semantic lexicons. In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 1606–1615, 2015. 131

[110] Christiane Fellbaum, editor. *WordNet An Electronic Lexical Database*. The MIT Press, Cambridge, MA ; London, 1998. ISBN 978-0-262-06197-1. 126

[111] David A. Ferrucci. Introduction to "this is watson". *IBM Journal of Research and Development*, 56(3):1, 2012. doi: 10.1147/JRD.2012.2184356. 149

[112] Karina Figueroa and Kimmo Fredriksson. Speeding up permutation based indexing with indexing. In Tomás Skopal and Pavel Zezula, editors, *Second International Workshop on Similarity Search and Applications, SISAP 2009, 29-30 August 2009, Prague, Czech Republic*, pages 107–114. IEEE Computer Society, 2009. doi: 10.1109/SISAP.2009.12. 42

[113] Agner Fog. Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for intel, amd and via cpus. *Copenhagen University College of Engineering*, 2011. 97, 114

[114] Marcus Fontoura, Vanja Josifovski, Jinhui Liu, Srihari Venkatesan, Xiangfei Zhu, and Jason Y. Zien. Evaluation strategies for top-k queries over memory-resident inverted indexes. *PVLDB*, 4(12):1213–1224, 2011. URL `http://www.vldb.org/pvldb/vol4/p1213-fontoura.pdf`. 5, 159, 161

[115] Daniel Fried, Peter Jansen, Gustave Hahn-Powell, Mihai Surdeanu, and Peter Clark. Higher-order lexical semantic models for non-factoid answer reranking. *TACL*, 3:197–210, 2015. 102, 104, 130

[116] Karl Pearson F.R.S. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 6*, 2(11):559–572, 1901. doi: 10.1080/14786440109462720. 26, 102

[117] Cong Fu, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with navigating spreading-out graphs. *CoRR*, abs/1707.00143, 2017. 158

[118] George W. Furnas, Thomas K. Landauer, Louis M. Gomez, and Susan T. Dumais. The vocabulary problem in human-system communication. *Commun. ACM*, 30(11):964–971, 1987. doi: 10.1145/32206.32212. 2

[119] Debasis Ganguly, Dwaipayan Roy, Mandar Mitra, and Gareth J. F. Jones. Word embedding based generalized language model for information retrieval. In Ricardo A. Baeza-Yates, Mounia Lalmas, Alistair Moffat, and Berthier A. Ribeiro-Neto, editors, *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, August 9-13, 2015*, pages 795–798. ACM, 2015. doi: 10.1145/2766462.2767780. 120, 155

[120] Stefan Gerdjikov, Stoyan Mihov, Petar Mitankin, and Klaus U. Schulz. Good parts first - a

new algorithm for approximate search in lexica and string databases. *CoRR*, abs/1301.0722, 2013. 21

[121] Kingshy Goh, Beitao Li, and Edward Y. Chang. Dyndex: a dynamic and non-metric space indexer. In Lawrence A. Rowe, Bernard Mérialdo, Max Mühlhäuser, Keith W. Ross, and Nevenka Dimitrova, editors, *Proceedings of the 10th ACM International Conference on Multimedia 2002, Juan les Pins, France, December 1-6, 2002.*, pages 466–475. ACM, 2002. doi: 10.1145/641007.641107. 17, 31, 122

[122] James Gorman and James R. Curran. Scaling distributional similarity to large corpora. In *ACL 2006, 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Sydney, Australia, 17-21 July 2006*, 2006. 102

[123] Kristen Grauman and Rob Fergus. *Learning Binary Hash Codes for Large-Scale Image Search*, pages 49–87. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-28661-2. doi: 10.1007/978-3-642-28661-2_3. URL `http://dx.doi.org/10.1007/978-3-642-28661-2_3`. 156

[124] Jiatao Gu, Yong Wang, Kyunghyun Cho, and Victor O. K. Li. Search engine guided non-parametric neural machine translation. *CoRR*, abs/1705.07267, 2017. 158

[125] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. Semantic matching by non-linear word transportation for information retrieval. In Snehasis Mukhopadhyay, ChengXiang Zhai, Elisa Bertino, Fabio Crestani, Javed Mostafa, Jie Tang, Luo Si, Xiaofang Zhou, Yi Chang, Yunyao Li, and Parikshit Sondhi, editors, *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, pages 701–710. ACM, 2016. doi: 10.1145/2983323.2983768. 120, 155

[126] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In Beatrice Yormark, editor, *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 47–57. ACM Press, 1984. doi: 10.1145/602259.602266. 31

[127] Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In Toby Walsh, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 1312–1317. IJCAI/AAAI, 2011. doi: 10.5591/978-1-57735-516-8/IJCAI11-222. 36, 37

[128] Nathan Halko, Per-Gunnar Martinsson, Yoel Shkolnisky, and Mark Tygert. An algorithm for the principal component analysis of large data sets. *SIAM J. Scientific Computing*, 33 (5):2580–2594, 2011. doi: 10.1137/100804139. 26

[129] Feng Hao, John Daugman, and Piotr Zielinski. A fast search algorithm for a large fuzzy database. *IEEE Trans. Information Forensics and Security*, 3(2):203–212, 2008. doi: 10.1109/TIFS.2008.920726. 18, 21, 160

[130] Ben Harwood and Tom Drummond. FANNG: fast approximate nearest neighbour graphs. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 5713–5722, 2016. doi: 10.1109/CVPR.2016.616.

37, 158

[131] Magnus Lie Hetland. Ptolemaic indexing. *JoCG*, 6(1):165–184, 2015. 15

[132] Djoerd Hiemstra. Using language models for information retrieval, 2001. 117

[133] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. ISSN 0036-8075. doi: 10.1126/science.1127647. 26, 102

[134] Gísli R. Hjaltason and Hanan Samet. Properties of embedding methods for similarity searching in metric spaces. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(5):530–549, 2003. doi: 10.1109/TPAMI.2003.1195989. 1, 21

[135] Matthew Honnibal and Mark Johnson. An improved non-monotonic transition system for dependency parsing. In Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1373–1378. The Association for Computational Linguistics, 2015. 128, 143

[136] Michael E. Houle and Michael Nett. Rank cover trees for nearest neighbor search. In *Similarity Search and Applications - 6th International Conference, SISAP 2013, A Coruña, Spain, October 2-4, 2013, Proceedings*, pages 16–29, 2013. doi: 10.1007/978-3-642-41062-8_3. 37

[137] Michael E. Houle and Jun Sakuma. Fast approximate similarity search in extremely high-dimensional data sets. In *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, pages 619–630, 2005. doi: 10.1109/ICDE.2005.66. 37

[138] Michael E. Houle and Jun Sakuma. Fast approximate similarity search in extremely high-dimensional data sets. In Karl Aberer, Michael J. Franklin, and Shojiro Nishio, editors, *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, pages 619–630. IEEE Computer Society, 2005. doi: 10.1109/ICDE.2005.66. 37

[139] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry P. Heck. Learning deep structured semantic models for web search using clickthrough data. In Qi He, Arun Iyengar, Wolfgang Nejdl, Jian Pei, and Rajeev Rastogi, editors, *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pages 2333–2338. ACM, 2013. doi: 10.1145/2505515.2505665. 155

[140] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952. 145

[141] Samuel Huston and W. Bruce Croft. A comparison of retrieval models using term dependencies. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*, pages 111–120, 2014. doi: 10.1145/2661829.2661894. 118

[142] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the

curse of dimensionality. In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 604–613. ACM, 1998. doi: 10.1145/276698.276876. 34, 35

[143] Fumitada Itakura and Shuzo Saito. Analysis synthesis telephony based on the maximum likelihood method. In *Proceedings of the 6th International Congress on Acoustics*, pages C17–C20. Los Alamitos, CA: IEEE, 1968. 11, 69

[144] Mohit Iyyer, Jordan Boyd-Graber, Leonardo Claudino, Richard Socher, and Hal Daumé III. A neural network for factoid question answering over paragraphs. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 633–644, Doha, Qatar, October 2014. Association for Computational Linguistics. 155

[145] Mike Izbicki and Christian R. Shelton. Faster cover trees. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1162–1170. JMLR.org, 2015. 99

[146] David W. Jacobs, Daphna Weinshall, and Yoram Gdalyahu. Classification with nonmetric distances: Image retrieval and class representation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(6):583–600, 2000. doi: 10.1109/34.862197. 1, 21, 77

[147] Nasreen Abdul Jaleel, James Allan, W. Bruce Croft, Fernando Diaz, Leah S. Larkey, Xiaoyan Li, Mark D. Smucker, and Courtney Wade. Umass at TREC 2004: Novelty and HARD. In *Proceedings of the Thirteenth Text REtrieval Conference, TREC 2004, Gaithersburg, Maryland, USA, November 16-19, 2004*, 2004. 113, 117

[148] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002. doi: 10.1145/582415.582418. 103, 111

[149] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128, 2011. doi: 10.1109/TPAMI.2010.57. 17, 29, 31, 57, 156

[150] Herve Jegou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. Searching in one billion vectors: Re-rank with source coding. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2011, May 22-27, 2011, Prague Congress Center, Prague, Czech Republic*, pages 861–864. IEEE, 2011. doi: 10.1109/ICASSP.2011.5946540. 56, 57, 156

[151] Jiwoon Jeon, W. Bruce Croft, and Joon Ho Lee. Finding similar questions in large question and answer archives. In *Proceedings of the 2005 ACM CIKM International Conference on Information and Knowledge Management, Bremen, Germany, October 31 - November 5, 2005*, pages 84–90, 2005. doi: 10.1145/1099554.1099572. 116

[152] Lu Jiang, Junwei Liang, Liangliang Cao, Yannis Kalantidis, Sachin Farfade, and Alexander G. Hauptmann. Memexqa: Visual memex question answering. *CoRR*, abs/1708.01336, 2017. 158

[153] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *CoRR*, abs/1702.08734, 2017. 156

[154] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python. `http://www.scipy.org/` [Last Checked June 2017], 2001–. 75

[155] Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. Generating query substitutions. In *Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland, UK, May 23-26, 2006*, pages 387–396, 2006. doi: 10.1145/1135777.1135835. 140

[156] Robert Kallman, Hideaki Kimura, Jonathan Natkins, Andrew Pavlo, Alex Rasin, Stanley B. Zdonik, Evan P. C. Jones, Samuel Madden, Michael Stonebraker, Yang Zhang, John Hugg, and Daniel J. Abadi. H-store: a high-performance, distributed main memory transaction processing system. *PVLDB*, 1(2):1496–1499, 2008. 7, 114

[157] David R. Karger and Matthias Ruhl. Finding nearest neighbors in growth-restricted metrics. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 741–750. ACM, 2002. doi: 10.1145/509907.510013. 15

[158] L. Kaufman and P. Rousseeuw. Clustering by means of medoids. In Yadolah Dodge, editor, *Statistical Data Analysis Based on the L1-Norm and Related Methods*, pages 405–416. North-Holland, 1987. 17, 31

[159] M. G. KENDALL. A new measure of rank correlation. *Biometrika*, 30(1-2):81, 1938. doi: 10.1093/biomet/30.1-2.81. 13

[160] Paul Kingsbury and Martha Palmer. From treebank to propbank. In *Proceedings of the Third International Conference on Language Resources and Evaluation, LREC 2002, May 29-31, 2002, Las Palmas, Canary Islands, Spain*, 2002. 126

[161] Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 3294–3302, 2015. 102, 150, 155

[162] Michal Kleinbort, Oren Salzman, and Dan Halperin. Efficient high-quality motion planning by fast all-pairs r-nearest-neighbors. In *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, pages 2985–2990. IEEE, 2015. doi: 10.1109/ICRA.2015.7139608. 99

[163] Michal Kleinbort, Oren Salzman, and Dan Halperin. Collision detection or nearest-neighbor search? on the computational bottleneck in sampling-based motion planning. *CoRR*, abs/1607.04800, 2016. 99

[164] David Konopnicki and Oded Shmueli. Database-inspired search. In Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors, *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, page 2. ACM, 2005. 2

[165] Yehuda Koren, Robert M. Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009. doi: 10.1109/MC.2009.263.

102

[166] Flip Korn, Bernd-Uwe Pagel, and Christos Faloutsos. On the 'dimensionality curse' and the 'self-similarity blessing'. *IEEE Trans. Knowl. Data Eng.*, 13(1):96–111, 2001. doi: 10.1109/69.908983. 15, 18

[167] Gregory Kucherov, Kamil Salikhov, and Dekel Tsur. Approximate string matching using a bidirectional index. In Alexander S. Kulikov, Sergei O. Kuznetsov, and Pavel A. Pevzner, editors, *Combinatorial Pattern Matching - 25th Annual Symposium, CPM 2014, Moscow, Russia, June 16-18, 2014. Proceedings*, volume 8486 of *Lecture Notes in Computer Science*, pages 222–231. Springer, 2014. doi: 10.1007/978-3-319-07566-2_23. 21

[168] Brian Kulis. Metric learning: A survey. *Foundations and Trends in Machine Learning*, 5 (4):287–364, 2013. doi: 10.1561/2200000019. 24, 25, 156

[169] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In Yoshua Bengio, Dale Schuurmans, John D. Lafferty, Christopher K. I. Williams, and Aron Culotta, editors, *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.*, pages 1042–1050. Curran Associates, Inc., 2009. 23, 25, 44

[170] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *IEEE 12th International Conference on Computer Vision, ICCV 2009, Kyoto, Japan, September 27 - October 4, 2009*, pages 2130–2137. IEEE Computer Society, 2009. doi: 10.1109/ICCV.2009.5459466. 35, 156

[171] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22 (1):79–86, 03 1951. doi: 10.1214/aoms/1177729694. URL `http://dx.doi.org/10.1214/aoms/1177729694`. 11, 14, 69

[172] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 614–623. ACM, 1998. doi: 10.1145/276698.276877. 34

[173] Matt J. Kusner, Stephen Tyree, Kilian Q. Weinberger, and Kunal Agrawal. Stochastic neighbor compression. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 622–630, 2014. 102

[174] Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. From word embeddings to document distances. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 957–966, 2015. 102, 155

[175] Victor Lavrenko and W. Bruce Croft. Relevance-based language models. In *SIGIR 2001: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, September 9-13, 2001, New Orleans, Louisiana, USA*, pages 120–127, 2001. doi: 10.1145/383952.383972. 117

[176] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents.

In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1188–1196, 2014. 130, 131, 150, 155

[177] Bastian Leibe, Krystian Mikolajczyk, and Bernt Schiele. Efficient clustering and matching for object class recognition. In Mike J. Chantler, Robert B. Fisher, and Emanuele Trucco, editors, *Proceedings of the British Machine Vision Conference 2006, Edinburgh, UK, September 4-7, 2006*, pages 789–798. British Machine Vision Association, 2006. doi: 10.5244/C.20.81. 31

[178] Daniel Lemire and Leonid Boytsov. Decoding billions of integers per second through vectorization. *Softw., Pract. Exper.*, 45(1):1–29, 2015. doi: 10.1002/spe.2203. xii, 145, 146, 147, 148

[179] Daniel Lemire, Leonid Boytsov, and Nathan Kurz. SIMD compression and the intersection of sorted integers. *Softw., Pract. Exper.*, 46(6):723–749, 2016. doi: 10.1002/spe.2326. 118, 145, 148

[180] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5: 361–397, 2004. 50, 68

[181] Chen Li, Edward Y. Chang, Hector Garcia-Molina, and Gio Wiederhold. Clustering for approximate similarity search in high-dimensional spaces. *IEEE Trans. Knowl. Data Eng.*, 14(4):792–808, 2002. doi: 10.1109/TKDE.2002.1019214. 122

[182] Chen Li, Jiaheng Lu, and Yiming Lu. Efficient merging and filtering algorithms for approximate string searches. In Gustavo Alonso, José A. Blakeley, and Arbee L. P. Chen, editors, *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*, pages 257–266. IEEE Computer Society, 2008. doi: 10.1109/ICDE.2008.4497434. 60

[183] Hao Li, Wei Liu, and Heng Ji. Two-stage hashing for fast document retrieval. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 2: Short Papers*, pages 495–500, 2014. 102

[184] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Wenjie Zhang, and Xuemin Lin. Approximate nearest neighbor search on high dimensional data - experiments, analyses, and improvement (v1.0). *CoRR*, abs/1610.02455, 2016. 37, 158

[185] Jimmy Lin and Andrew Trotman. The role of index compression in score-at-a-time query evaluation. *Information Retrieval Journal*, pages 1–22, 2017. ISSN 1573-7659. doi: 10.1007/s10791-016-9291-5. 145

[186] Jimmy J. Lin and Andrew Trotman. Anytime ranking for impact-ordered indexes. In James Allan, W. Bruce Croft, Arjen P. de Vries, and Chengxiang Zhai, editors, *Proceedings of the 2015 International Conference on The Theory of Information Retrieval, ICTIR 2015, Northampton, Massachusetts, USA, September 27-30, 2015*, pages 301–304. ACM, 2015. doi: 10.1145/2808194.2809477. 5, 159, 161

[187] Jimmy J. Lin, Matt Crane, Andrew Trotman, Jamie Callan, Ishan Chattopadhyaya, John

Foley, Grant Ingersoll, Craig MacDonald, and Sebastiano Vigna. *Toward Reproducible Baselines: The Open-Source IR Reproducibility Challenge*, pages 408–420. 2016. doi: 10.1007/978-3-319-30671-1_30. 124, 140

[188] Eric Yi Liu, Zhishan Guo, Xiang Zhang, Vladimir Jojic, and Wei Wang. Metric learning from relative comparisons by minimizing squared residual. In Mohammed Javeed Zaki, Arno Siebes, Jeffrey Xu Yu, Bart Goethals, Geoffrey I. Webb, and Xindong Wu, editors, *12th IEEE International Conference on Data Mining, ICDM 2012, Brussels, Belgium, December 10-13, 2012*, pages 978–983. IEEE Computer Society, 2012. doi: 10.1109/ICDM.2012.38. 73

[189] Xiaohua Liu, Shaodian Zhang, Furu Wei, and Ming Zhou. Recognizing named entities in tweets. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, pages 359–367, 2011. 102

[190] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Information Theory*, 28 (2):129–136, 1982. doi: 10.1109/TIT.1982.1056489. 15, 16, 31, 121

[191] David G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, pages 1150–1157, 1999. doi: 10.1109/ICCV.1999.790410. 23, 57

[192] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In Christoph Koch, Johannes Gehrke, Minos N. Garofalakis, Divesh Srivastava, Karl Aberer, Anand Deshpande, Daniela Florescu, Chee Yong Chan, Venkatesh Ganti, Carl-Christian Kanne, Wolfgang Klas, and Erich J. Neuhold, editors, *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 950–961. ACM, 2007. 34

[193] Yuanhua Lv and ChengXiang Zhai. A comparative study of methods for estimating query language models with pseudo feedback. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009*, pages 1895–1898, 2009. doi: 10.1145/1645953.1646259. 117

[194] Craig Macdonald, Richard McCreadie, Rodrygo LT Santos, and Iadh Ounis. From puppy to maturity: Experiences in developing terrier. *Proc. of OSIR at SIGIR*, pages 60–63, 2012. 123, 124

[195] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Scalable distributed algorithm for approximate nearest neighbor search problem in high dimensional general metric spaces. In Gonzalo Navarro and Vladimir Pestov, editors, *Similarity Search and Applications - 5th International Conference, SISAP 2012, Toronto, ON, Canada, August 9-10, 2012. Proceedings*, volume 7404 of *Lecture Notes in Computer Science*, pages 132–147. Springer, 2012. doi: 10.1007/978-3-642-32153-5_10. 39

[196] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Inf. Syst.*, 45: 61–68, 2014. doi: 10.1016/j.is.2013.10.006. 5, 36, 37, 39, 58, 59, 113, 158

[197] Yury A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor

search using hierarchical navigable small world graphs. *CoRR*, abs/1603.09320, 2016. v, 21, 37, 39, 122

[198] Antonio Mallia, Giuseppe Ottaviano, Elia Porciani, Nicola Tonellotto, and Rossano Venturini. Faster blockmax WAND with variable-sized blocks. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, pages 625–634, 2017. doi: 10.1145/3077136.3080780. URL http://doi.acm.org/10.1145/3077136.3080780. 5, 159, 161

[199] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008. ISBN 978-0-521-86571-5. 14, 19, 101

[200] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, System Demonstrations*, pages 55–60, 2014. 108, 125

[201] Mitchell P. Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The penn treebank: Annotating predicate argument structure. In *Human Language Technology, Proceedings of a Workshop held at Plainsboro, New Jerey, USA, March 8-11, 1994*, 1994. 125

[202] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In Raghu Ramakrishnan, Salvatore J. Stolfo, Roberto J. Bayardo, and Ismail Parsa, editors, *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA, USA, August 20-23, 2000*, pages 169–178. ACM, 2000. doi: 10.1145/347090.347123. 31

[203] J. Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 209(441-458):415–446, 1909. ISSN 0264-3952. doi: 10.1098/rsta.1909.0016. URL http://rsta.royalsocietypublishing.org/content/209/441-458/415. 17

[204] Donald Metzler and W. Bruce Croft. A markov random field model for term dependencies. In Ricardo A. Baeza-Yates, Nivio Ziviani, Gary Marchionini, Alistair Moffat, and John Tait, editors, *SIGIR 2005: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Salvador, Brazil, August 15-19, 2005*, pages 472–479. ACM, 2005. doi: 10.1145/1076034.1076115. 112, 117, 134

[205] Donald Metzler and W. Bruce Croft. Linear feature-based models for information retrieval. *Inf. Retr.*, 10(3):257–274, 2007. doi: 10.1007/s10791-006-9019-z. 104, 115

[206] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013. 130, 131, 132, 143, 155

[207] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous

space word representations. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA*, pages 746–751, 2013. 102, 130

[208] Bhaskar Mitra, Eric T. Nalisnick, Nick Craswell, and Rich Caruana. A dual embedding space model for document ranking. *CoRR*, abs/1602.01137, 2016. 155

[209] Alistair Moffat and Justin Zobel. Rank-biased precision for measurement of retrieval effectiveness. *ACM Trans. Inf. Syst.*, 27(1):2:1–2:27, 2008. doi: 10.1145/1416950.1416952. 13

[210] Sean Moran, Richard McCreadie, Craig Macdonald, and Iadh Ounis. Enhancing first story detection using word embeddings. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, July 17-21, 2016*, pages 821–824, 2016. doi: 10.1145/2911451.2914719. 102

[211] Sean James Moran. *Learning to Hash for Large Scale Image Retrieval*. PhD thesis, The University of Edinburgh, 2016. URL `https://www.era.lib.ed.ac.uk/handle/1842/20390`. 156

[212] Sean James Moran. *Learning to hash for large scale image retrieval*. PhD thesis, The University of Edinburgh, 2016. 156

[213] David M Mount. ANN Programming Manual, 2010. URL `https://www.cs.umd.edu/~mount/ANN/Files/1.1.2/ANNmanual_1.1.pdf`. Version 1.1.2, [Last checked September 2016]. 33, 35

[214] Yadong Mu and Shuicheng Yan. Non-metric locality-sensitive hashing. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010. 35

[215] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In Alpesh Ranchordas and Helder Araújo, editors, *VISAPP 2009 - Proceedings of the Fourth International Conference on Computer Vision Theory and Applications, Lisboa, Portugal, February 5-8, 2009 - Volume 1*, pages 331–340. INSTICC Press, 2009. 29, 31, 35, 36, 113

[216] J. William Murdock, James Fan, Adam Lally, Hideki Shima, and Branimir Boguraev. Textual evidence gathering and analysis. *IBM Journal of Research and Development*, 56 (3):8, 2012. doi: 10.1147/JRD.2012.2187249. 158

[217] J. William Murdock, Aditya Kalyanpur, Chris Welty, James Fan, David A. Ferrucci, David Gondek, Lei Zhang, and Hiroshi Kanayama. Typing candidate answers using type coercion. *IBM Journal of Research and Development*, 56(3):7, 2012. doi: 10.1147/JRD.2012.2187036. 156

[218] Bilegsaikhan Naidan. *Engineering Efficient Similarity Search Methods*. PhD thesis, NTNU: Norwegian University of Science and Technology, 2017. URL `https://brage.bibsys.no/xmlui/handle/11250/2441681`. v

[219] Bilegsaikhan Naidan and Leonid Boytsov. Non-metric space library manual. *CoRR*,

abs/1508.05470, 2015. 58, 59, 62, 77, 113

[220] Bilegsaikhan Naidan, Leonid Boytsov, and Eric Nyberg. Permutation search methods are efficient, yet faster search is possible. *PVLDB*, 8(12):1618–1629, 2015. 21, 35, 99

[221] Gonzalo Navarro. Searching in metric spaces by spatial approximation. *VLDB J.*, 11(1): 28–46, 2002. doi: 10.1007/s007780200060. 36

[222] Gonzalo Navarro. Spaces, trees, and colors: The algorithmic landscape of document retrieval on sequences. *ACM Comput. Surv.*, 46(4):52:1–52:47, 2013. doi: 10.1145/2535933. 21

[223] Gonzalo Navarro, Ricardo A. Baeza-Yates, Erkki Sutinen, and Jorma Tarhio. Indexing methods for approximate string matching. *IEEE Data Eng. Bull.*, 24(4):19–27, 2001. 21

[224] Roberto Navigli. Word sense disambiguation: A survey. *ACM Comput. Surv.*, 41(2): 10:1–10:69, 2009. doi: 10.1145/1459352.1459355. 126

[225] Behnam Neyshabur and Nathan Srebro. On symmetric and asymmetric lshs for inner product search. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1926–1934. JMLR.org, 2015. 19, 20, 104

[226] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 144–155. Morgan Kaufmann, 1994. 31, 122

[227] Frank Nielsen, Paolo Piro, and Michel Barlaud. Bregman vantage point trees for efficient nearest neighbor queries. In *Proceedings of the 2009 IEEE International Conference on Multimedia and Expo, ICME 2009, June 28 - July 2, 2009, New York City, NY, USA*, pages 878–881. IEEE, 2009. doi: 10.1109/ICME.2009.5202635. 21, 44

[228] Mohammad Norouzi and David J. Fleet. Minimal loss hashing for compact binary codes. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 353–360, 2011. 25

[229] Mohammad Norouzi, David J. Fleet, and Ruslan Salakhutdinov. Hamming distance metric learning. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1070–1078, 2012. 23, 25

[230] Mohammad Norouzi, Ali Punjani, and David J. Fleet. Fast search in Hamming space with multi-index hashing. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 3108–3115, 2012. doi: 10.1109/CVPR.2012.6248043. 23

[231] Mohammad Norouzi, Ali Punjani, and David J. Fleet. Fast exact search in hamming space with multi-index hashing. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(6):1107–1119, 2014. doi: 10.1109/TPAMI.2013.231. 160

[232] David Novak, Michal Batko, and Pavel Zezula. Metric index: An efficient and scalable solution for precise and approximate similarity search. *Inf. Syst.*, 36(4):721–733, 2011. doi: 10.1016/j.is.2010.10.002. 99

[233] Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51, 2003. doi: 10.1162/089120103321337421. 116

[234] Stephen M Omohundro. Five balltree construction algorithms, 1989. URL `http://www.icsi.berkeley.edu/icsi/publication_details?ID=000562`. ICSI Technical Report TR-89-063, [Last checked September 2016]. 27, 32, 42, 45, 47, 59, 70

[235] Cheng Soon Ong, Xavier Mary, Stéphane Canu, and Alexander J. Smola. Learning with non-positive kernels. In *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*, 2004. doi: 10.1145/1015330.1015443. 17

[236] Sukomal Pal, Mandar Mitra, and Jaap Kamps. Evaluation effort, reliability and reusability in XML retrieval. *JASIST*, 62(2):375–394, 2011. doi: 10.1002/asi.21403. 106

[237] Rodrigo Paredes, Edgar Chávez, Karina Figueroa, and Gonzalo Navarro. Practical construction of $k$-nearest neighbor graphs in metric spaces. In Carme Àlvarez and Maria J. Serna, editors, *Experimental Algorithms, 5th International Workshop, WEA 2006, Cala Galdana, Menorca, Spain, May 24-27, 2006, Proceedings*, volume 4007 of *Lecture Notes in Computer Science*, pages 85–97. Springer, 2006. doi: 10.1007/11764298_8. 37

[238] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543, 2014. 131, 150, 155

[239] Vladimir Pestov. Indexability, concentration, and VC theory. *J. Discrete Algorithms*, 13:2–18, 2012. doi: 10.1016/j.jda.2011.10.002. 2, 18

[240] Sasa Petrovic, Miles Osborne, and Victor Lavrenko. Streaming first story detection with application to twitter. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 2-4, 2010, Los Angeles, California, USA*, pages 181–189, 2010. 102

[241] Sasa Petrovic, Miles Osborne, and Victor Lavrenko. Using paraphrases for improving first story detection in news and twitter. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 3-8, 2012, Montréal, Canada*, pages 338–346, 2012. 102

[242] Erion Plaku and Lydia E. Kavraki. Distributed computation of the knn graph for large high-dimensional point sets. *J. Parallel Distrib. Comput.*, 67(3):346–359, 2007. doi: 10.1016/j.jpdc.2006.10.004. 37

[243] Barnabás Póczos, Liang Xiong, Dougal J. Sutherland, and Jeff G. Schneider. Nonparametric

kernel estimators for image classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 2989–2996, 2012. doi: 10.1109/CVPR.2012.6248028. 11

[244] Alexander Ponomarenko, Yury Malkov, Andrey Logvinov, and Vladimir Krylov. Approximate nearest neighbor search small world approach. `http://www.iiis.org/CDs2011/CD2011IDI/ICTA_2011/Abstract.asp?myurl=CT175ON.pdf` [Last accessed June 2017], 2011. 39

[245] Alexander Ponomarenko, Nikita Avrelin, Bilegsaikhan Naidan, and Leonid Boytsov. Comparative analysis of data structures for approximate nearest neighbor search. In *DATA ANALYTICS 2014, The Third International Conference on Data Analytics*, pages 125–130, 2014. 21, 54, 55, 99

[246] Jay M. Ponte and W. Bruce Croft. A language modeling approach to information retrieval. In *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 24-28 1998, Melbourne, Australia*, pages 275–281, 1998. doi: 10.1145/290941.291008. 117

[247] John M. Prager. Open-domain question-answering. *Foundations and Trends in Information Retrieval*, 1(2):91–231, 2006. doi: 10.1561/1500000001. 107, 127, 128

[248] Guo-Jun Qi, Jinhui Tang, Zheng-Jun Zha, Tat-Seng Chua, and Hong-Jiang Zhang. An efficient sparse metric learning in high-dimensional space via $l_1$-penalized log-determinant regularization. In Andrea Pohoreckyj Danyluk, Léon Bottou, and Michael L. Littman, editors, *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, volume 382 of *ACM International Conference Proceeding Series*, pages 841–848. ACM, 2009. doi: 10.1145/1553374.1553482. 73

[249] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2383–2392, 2016. 106, 107

[250] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`. 55, 56, 57, 58, 68, 131

[251] Alfréd Rényi. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, pages 547–561, Berkeley, Calif., 1961. University of California Press. URL `http://projecteuclid.org/euclid.bsmsp/1200512181`. 11, 69

[252] Stefan Riezler and Yi Liu. Query rewriting using monolingual statistical machine translation. *Computational Linguistics*, 36(3):569–582, 2010. doi: 10.1162/coli_a_00010. 106

[253] Stefan Riezler, Alexander Vasserman, Ioannis Tsochantaridis, Vibhu O. Mittal, and Yi Liu. Statistical machine translation for query expansion in answer retrieval. In *ACL 2007,*

*Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic*, 2007. 104

[254] Stephen Robertson. Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of Documentation*, 60(5):503–520, 2004. doi: 10.1108/00220410410560582. 2, 69, 102, 112, 115

[255] Volker Roth, Julian Laub, Joachim M. Buhmann, and Klaus-Robert Müller. Going metric: Denoising pairwise data. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems 15 [Neural Information Processing Systems, NIPS 2002, December 9-14, 2002, Vancouver, British Columbia, Canada]*, pages 817–824. MIT Press, 2002. 21

[256] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*, pages 318–362. MIT Press, Cambridge, MA, USA, 1986. ISBN 0-262-68053-X. 26

[257] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge, 2014. 56, 57

[258] Tetsuya Sakai and Teruko Mitamura. Boiling down information retrieval test collections. In *Recherche d'Information Assistée par Ordinateur, RIAO 2010: Adaptivity, Personalization and Fusion of Heterogeneous Information, 9th International Conference, Bibliotheque Nationale de France, Paris, France, April 28-30, 2010, Proceedings*, pages 49–56, 2010. 106

[259] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, Feb 1978. ISSN 0096-3518. doi: 10.1109/TASSP.1978.1163055. 14

[260] Ruslan Salakhutdinov and Geoffrey E. Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, AISTATS 2007, San Juan, Puerto Rico, March 21-24, 2007*, pages 412–419, 2007. 24, 155

[261] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986. ISBN 0070544840. 101

[262] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. ISBN 0123694469. 1, 21, 36, 40

[263] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999. doi: 10.1023/A:1007614523901. 25

[264] Nico Schlaefer, Jeongwoo Ko, Justin Betteridge, Manas A. Pathak, Eric Nyberg, and Guido Sautter. Semantic extensions of the ephyra QA system for TREC 2007. In *Proceedings of The Sixteenth Text REtrieval Conference, TREC 2007, Gaithersburg, Maryland, USA,*

*November 5-9, 2007*, 2007. 127

[265] Benjamin Schlegel, Thomas Willhalm, and Wolfgang Lehner. Fast sorted-set intersection using SIMD instructions. In Rajesh Bordawekar and Christian A. Lang, editors, *International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures - ADMS 2011, Seattle, WA, USA, September 2, 2011.*, pages 1–8, 2011. 57

[266] Frank-Michael Schleif and Peter Tiňo. Indefinite proximity learning: A review. *Neural Computation*, 27(10):2039–2096, 2015. doi: 10.1162/NECO_a_00770. 14, 15, 17

[267] Thomas B. Sebastian and Benjamin B. Kimia. Metric-based shape retrieval in large databases. In *16th International Conference on Pattern Recognition, ICPR 2002, Quebec, Canada, August 11-15, 2002.*, pages 291–296. IEEE Computer Society, 2002. doi: 10.1109/ICPR.2002.1047852. 37

[268] Uri Shaft and Raghu Ramakrishnan. When is nearest neighbors indexable? In Thomas Eiter and Leonid Libkin, editors, *Database Theory - ICDT 2005, 10th International Conference, Edinburgh, UK, January 5-7, 2005, Proceedings*, volume 3363 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 2005. doi: 10.1007/978-3-540-30570-5_11. 2, 18

[269] Chanop Silpa-Anan and Richard I. Hartley. Optimised kd-trees for fast image descriptor matching. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA*. IEEE Computer Society, 2008. doi: 10.1109/CVPR.2008.4587638. 33

[270] Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *9th IEEE International Conference on Computer Vision (ICCV 2003), 14-17 October 2003, Nice, France*, pages 1470–1477, 2003. doi: 10.1109/ICCV.2003.1238663. 17, 29, 156

[271] Tomás Skopal. Unified framework for fast exact and approximate search in dissimilarity spaces. *ACM Trans. Database Syst.*, 32(4), 2007. doi: 10.1145/1292609.1292619. 1, 5, 21, 25, 33, 64, 86, 87, 97

[272] Tomás Skopal and Benjamin Bustos. On nonmetric similarity search problems in complex domains. *ACM Comput. Surv.*, 43(4):34, 2011. doi: 10.1145/1978802.1978813. 1

[273] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195 – 197, 1981. ISSN 0022-2836. doi: http://dx.doi.org/10.1016/0022-2836(81)90087-5. 14, 15

[274] Aya Soffer, David Carmel, Doron Cohen, Ronald Fagin, Eitan Farchi, Michael Herscovici, and Yoëlle S. Maarek. Static index pruning for information retrieval systems. In W. Bruce Croft, David J. Harper, Donald H. Kraft, and Justin Zobel, editors, *SIGIR 2001: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, September 9-13, 2001, New Orleans, Louisiana, USA*, pages 43–50. ACM, 2001. doi: 10.1145/383952.383958. 102

[275] Radu Soricut and Eric Brill. Automatic question answering using the web: Beyond the

factoid. *Inf. Retr.*, 9(2):191–206, 2006. doi: 10.1007/s10791-006-7149-y. 104

[276] Kyle Soska, Christopher Gates, Kevin Roundy, and Nicolas Christin. Automatic application identification from billions of files. In *Proceedings of the 23d ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, Nova Scotia, Canada, August 13-17, 2017*, 2017. 156

[277] Alexander A. Stepanov, Anil R. Gangolli, Daniel E. Rose, Ryan J. Ernst, and Paramjit S. Oberoi. Simd-based decoding of posting lists. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011*, pages 317–326, 2011. doi: 10.1145/2063576.2063627. 145

[278] Trevor Strohman and W. Bruce Croft. Efficient document retrieval in main memory. In Wessel Kraaij, Arjen P. de Vries, Charles L. A. Clarke, Norbert Fuhr, and Noriko Kando, editors, *SIGIR 2007: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Amsterdam, The Netherlands, July 23-27, 2007*, pages 175–182. ACM, 2007. doi: 10.1145/1277741.1277774. 20

[279] Trevor Strohman, Donald Metzler, Howard Turtle, and W Bruce Croft. Indri: A language-model based search engine for complex queries. `http://ciir.cs.umass.edu/pubfiles/ir-407.pdf` [Last Checked Apr 2017], 2005. 108, 149

[280] Mihai Surdeanu, Massimiliano Ciaramita, and Hugo Zaragoza. Learning to rank answers to non-factoid questions from web collections. *Computational Linguistics*, 37(2):351–383, 2011. doi: 10.1162/COLI_a_00051. xi, 68, 102, 104, 106, 111, 116, 123, 124, 125

[281] Dougal J. Sutherland, Junier B. Oliva, Barnabás Póczos, and Jeff G. Schneider. Linear-time learning on distributions with approximate kernel embeddings. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 2073–2079. AAAI Press, 2016. 14

[282] Mona Taghavi, Ahmed Patel, Nikita Schmidt, Christopher Wills, and Yiqi Tew. An analysis of web proxy logs with query distribution pattern approach for search engines. *Computer Standards & Interfaces*, 34(1):162–170, 2012. doi: 10.1016/j.csi.2011.07.001. 19

[283] Eric Sadit Tellez, Edgar Chávez, and Antonio Camarena-Ibarrola. A brief index for proximity searching. In Eduardo Bayro-Corrochano and Jan-Olof Eklundh, editors, *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, 14th Iberoamerican Conference on Pattern Recognition, CIARP 2009, Guadalajara, Jalisco, Mexico, November 15-18, 2009. Proceedings*, volume 5856 of *Lecture Notes in Computer Science*, pages 529–536. Springer, 2009. doi: 10.1007/978-3-642-10268-4_62. 40, 42, 60, 61

[284] Eric Sadit Tellez, Edgar Chávez, and Gonzalo Navarro. Succinct nearest neighbor search. *Inf. Syst.*, 38(7):1019–1030, 2013. doi: 10.1016/j.is.2012.06.005. 43, 60, 113, 120

[285] Larry H. Thiel and H. S. Heaps. Program design for retrospective searches on large data bases. *Information Storage and Retrieval*, 8(1):1–20, 1972. doi: 10.1016/0020-0271(72)90024-1. 145

[286] Wei Tong, Fengjie Li, Rong Jin, and Anil K. Jain. Large-scale near-duplicate image retrieval

by kernel density estimation. *IJMIR*, 1(1):45–58, 2012. doi: 10.1007/s13735-012-0012-6. 31

[287] Godfried T. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, 12(4):261–268, 1980. doi: 10.1016/0031-3203(80)90066-7. 36

[288] Riadh Mohamed Trad, Alexis Joly, Nozha Boujemaa, et al. Distributed approximate knn graph construction for high dimensional data. In *BDA'2012: 28e journées Bases de Données Avancées*, 2012. 37

[289] Gilad Tsur, Yuval Pinter, Idan Szpektor, and David Carmel. Identifying web queries with question intent. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 783–793, Republic and Canton of Geneva, Switzerland, 2016. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-4143-1. doi: 10.1145/2872427.2883058. 19

[290] Ferhan Türe, Tamer Elsayed, and Jimmy J. Lin. No free lunch: brute force vs. locality-sensitive hashing for cross-lingual pairwise similarity. In *Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2011, Beijing, China, July 25-29, 2011*, pages 943–952, 2011. doi: 10.1145/2009916.2010042. 102

[291] Peter D. Turney. Human-level performance on word analogy questions by latent relational analysis. *CoRR*, abs/cs/0412024, 2004. 102, 130, 155

[292] Howard R. Turtle and James Flood. Query evaluation: Strategies and optimizations. *Inf. Process. Manage.*, 31(6):831–850, 1995. doi: 10.1016/0306-4573(95)00020-H. 102, 114

[293] Jeffrey K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Inf. Process. Lett.*, 40(4):175–179, 1991. doi: 10.1016/0020-0190(91)90074-R. 27, 32, 42, 45, 47, 59, 70

[294] Stéfan van der Walt, S. Chris Colbert, and Gaël Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science and Engineering*, 13(2):22–30, 2011. doi: 10.1109/MCSE.2011.37. 75

[295] Stijn van Dongen and Anton J. Enright. Metric distances derived from cosine similarity and pearson and spearman correlations. *CoRR*, abs/1208.3145, 2012. 67

[296] Michel Verleysen, Damien François, Geoffroy Simon, and Vincent Wertz. On the effects of dimensionality on data analysis with neural networks. In José Mira and José R. Álvarez, editors, *Artificial Neural Nets Problem Solving Methods, 7th International Work-Conference on Artificial and Natural Neural Networks, IWANN2003, Maó, Menorca, Spain, June 3-6, 2003 Proceedings, Part II*, volume 2687 of *Lecture Notes in Computer Science*, pages 105–112. Springer, 2003. doi: 10.1007/3-540-44869-1_14. 18

[297] Sebastiano Vigna. Quasi-succinct indices. In Stefano Leonardi, Alessandro Panconesi, Paolo Ferragina, and Aristides Gionis, editors, *Sixth ACM International Conference on Web Search and Data Mining, WSDM 2013, Rome, Italy, February 4-8, 2013*, pages 83–92. ACM, 2013. doi: 10.1145/2433396.2433409. 140

[298] Olli Virmajoki and Pasi Fränti. Divide-and-conquer algorithm for creating neighborhood

graph for clustering. In *17th International Conference on Pattern Recognition, ICPR 2004, Cambridge, UK, August 23-26, 2004.*, pages 264–267. IEEE Computer Society, 2004. doi: 10.1109/ICPR.2004.1334103. 37

[299] Georgy Feodosevich Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Premier mémoire. Sur quelques propriétés des formes quadratiques positives parfaites. *Journal für die reine und angewandte Mathematik*, 134:198–287, 1908. 31, 36

[300] Xiaojun Wan and Yuxin Peng. The earth mover's distance as a semantic measure for document similarity. In *Proceedings of the 2005 ACM CIKM International Conference on Information and Knowledge Management, Bremen, Germany, October 31 - November 5, 2005*, pages 301–302, 2005. doi: 10.1145/1099554.1099637. 102

[301] Hongya Wang, Jiao Cao, LihChyun Shu, and Davood Rafiei. Locality sensitive hashing revisited: filling the gap between theory and algorithm analysis. In Qi He, Arun Iyengar, Wolfgang Nejdl, Jian Pei, and Rajeev Rastogi, editors, *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pages 1969–1978. ACM, 2013. doi: 10.1145/2505515.2505765. 34

[302] Jing Wang, Jingdong Wang, Gang Zeng, Zhuowen Tu, Rui Gan, and Shipeng Li. Scalable k-nn graph construction for visual descriptors. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 1106–1113. IEEE Computer Society, 2012. doi: 10.1109/CVPR.2012.6247790. 37

[303] Jing Wang, Jingdong Wang, Gang Zeng, Rui Gan, Shipeng Li, and Baining Guo. Fast neighborhood graph search using cartesian concatenation. In *IEEE International Conference on Computer Vision, ICCV 2013, Sydney, Australia, December 1-8, 2013*, pages 2128–2135, 2013. doi: 10.1109/ICCV.2013.265. 37

[304] Jingdong Wang and Shipeng Li. Query-driven iterated neighborhood graph search for large scale indexing. In Noboru Babaguchi, Kiyoharu Aizawa, John R. Smith, Shin'ichi Satoh, Thomas Plagemann, Xian-Sheng Hua, and Rong Yan, editors, *Proceedings of the 20th ACM Multimedia Conference, MM '12, Nara, Japan, October 29 - November 02, 2012*, pages 179–188. ACM, 2012. doi: 10.1145/2393347.2393378. 37

[305] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for similarity search: A survey. *CoRR*, abs/1408.2927, 2014. 1, 21, 34

[306] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. Learning to hash for indexing big data - A survey. *Proceedings of the IEEE*, 104(1):34–57, 2016. doi: 10.1109/JPROC.2015.2487976. URL `https://doi.org/10.1109/JPROC.2015.2487976`. 156

[307] Qing Wang, Sanjeev R. Kulkarni, and Sergio Verdú. Divergence estimation for multidimensional densities via k-nearest-neighbor distances. *IEEE Trans. Information Theory*, 55 (5):2392–2405, 2009. doi: 10.1109/TIT.2009.2016060. 14

[308] Wei Wang, Chuan Xiao, Xuemin Lin, and Chengqi Zhang. Efficient approximate entity extraction with edit distance constraints. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June*

*29 - July 2, 2009*, pages 759–770, 2009. doi: 10.1145/1559845.1559925. 102

[309] Robert H. Warren and Ting Liu. A review of relevance feedback experiments at the 2003 reliable information access (RIA) workshop. In *SIGIR 2004: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Sheffield, UK, July 25-29, 2004*, pages 570–571, 2004. doi: 10.1145/1008992.1009125. URL `http://doi.acm.org/10.1145/1008992.1009125`. 158

[310] William Webber, Alistair Moffat, and Justin Zobel. A similarity measure for indefinite rankings. *ACM Trans. Inf. Syst.*, 28(4):20:1–20:38, 2010. doi: 10.1145/1852102.1852106. 12, 13, 111, 133

[311] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In Ashish Gupta, Oded Shmueli, and Jennifer Widom, editors, *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 194–205. Morgan Kaufmann, 1998. 2, 18, 30, 33, 44

[312] Kilian Q. Weinberger and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10:207–244, 2009. doi: 10.1145/1577069.1577078. 25

[313] Kilian Q. Weinberger, John Blitzer, and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in Neural Information Processing Systems 18 [Neural Information Processing Systems, NIPS 2005, December 5-8, 2005, Vancouver, British Columbia, Canada]*, pages 1473–1480, 2005. 73, 74

[314] John S. Whissell and Charles L. A. Clarke. Effective measures for inter-document similarity. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pages 1361–1370, 2013. doi: 10.1145/2505515.2505526. 102

[315] John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. Towards universal paraphrastic sentence embeddings. *CoRR*, abs/1511.08198, 2015. 131

[316] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Commun. ACM*, 30(6):520–540, 1987. doi: 10.1145/214762.214771. 145

[317] Caiming Xiong, David M. Johnson, Ran Xu, and Jason J. Corso. Random forests for metric learning with implicit pairwise position dependence. In *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012*, pages 958–966, 2012. doi: 10.1145/2339530.2339680. 25, 73, 74, 75

[318] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. End-to-end neural ad-hoc ranking with kernel pooling. *CoRR*, abs/1706.06613, 2017. 155

[319] Xiaobing Xue, Jiwoon Jeon, and W. Bruce Croft. Retrieval models for question and answer archives. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2008, Singapore, July 20-24, 2008*, pages 475–482, 2008. doi: 10.1145/1390334.1390416. 104

[320] Liu Yang, Qingyao Ai, Damiano Spina, Ruey-Cheng Chen, Liang Pang, W. Bruce Croft, Jiafeng Guo, and Falk Scholer. Beyond factoid QA: effective methods for non-factoid answer sentence retrieval. In *Advances in Information Retrieval - 38th European Conference on IR Research, ECIR 2016, Padua, Italy, March 20-23, 2016. Proceedings*, pages 115–128, 2016. doi: 10.1007/978-3-319-30671-1_9. 130

[321] Xuchen Yao, Benjamin Van Durme, and Peter Clark. Automatic coupling of answer extraction and information retrieval. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 2: Short Papers*, pages 159–165. The Association for Computer Linguistics, 2013. 157

[322] Reyyan Yeniterzi and Jamie Callan. Analyzing bias in cqa-based expert finding test sets. In Shlomo Geva, Andrew Trotman, Peter Bruza, Charles L. A. Clarke, and Kalervo Järvelin, editors, *The 37th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '14, Gold Coast , QLD, Australia - July 06 - 11, 2014*, pages 967–970. ACM, 2014. doi: 10.1145/2600428.2609486. 111

[323] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas.*, pages 311–321. ACM/SIAM, 1993. 27, 32, 42, 45, 47, 59, 70

[324] Peter N. Yianilos. Locally lifting the curse of dimensionality for nearest neighbor search. In Michael H. Goldwasser, David S. Johnson, and Catherine C. McGeoch, editors, *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges, Proceedings of a DIMACS Workshop, USA, 1999*, volume 59 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, page 177. DIMACS/AMS, 1999. 33

[325] Hamed Zamani, Javid Dadashkarimi, Azadeh Shakery, and W. Bruce Croft. Pseudo-relevance feedback based on matrix factorization. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, pages 1483–1492, 2016. doi: 10.1145/2983323.2983844. 117

[326] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search - The Metric Space Approach*, volume 32 of *Advances in Database Systems*. Kluwer, 2006. ISBN 978-0-387-29146-8. doi: 10.1007/0-387-29151-2. 21, 40

[327] Zhenjie Zhang, Beng Chin Ooi, Srinivasan Parthasarathy, and Anthony K. H. Tung. Similarity search on bregman divergence: Towards non-metric indexing. *PVLDB*, 2(1):13–24, 2009. 21, 30, 32, 33, 44

[328] Le Zhao and Jamie Callan. Term necessity prediction. In Jimmy Huang, Nick Koudas, Gareth J. F. Jones, Xindong Wu, Kevyn Collins-Thompson, and Aijun An, editors, *Proceedings of the 19th ACM Conference on Information and Knowledge Management, CIKM 2010, Toronto, Ontario, Canada, October 26-30, 2010*, pages 259–268. ACM, 2010. doi: 10.1145/1871437.1871474. 2

[329] Liang Zheng, Yi Yang, and Qi Tian. SIFT meets CNN: A decade survey of instance retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(5):1224–1244, 2018. doi: 10.1109/TPAMI.2017.2709749. 31

[330] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Information Theory*, 23(3):337–343, 1977. doi: 10.1109/TIT.1977.1055714. 145

[331] Guido Zuccon, Bevan Koopman, Peter Bruza, and Leif Azzopardi. Integrating and evaluating neural word embeddings in information retrieval. In Laurence Anthony F. Park and Sarvnaz Karimi, editors, *Proceedings of the 20th Australasian Document Computing Symposium, ADCS 2015, Parramatta, NSW, Australia, December 8-9, 2015*, pages 12:1–12:8. ACM, 2015. doi: 10.1145/2838931.2838936. 120, 155