Ranking Preserving Hashing for Fast Similarity Search

Qifan Wang, Zhiwei Zhang and **Luo Si**
Computer Science Department, Purdue University
West Lafayette, IN 47907, US
wang868@purdue.edu, zhan1187@purdue.edu, lsi@purdue.edu

Abstract

Hashing method becomes popular for large scale similarity search due to its storage and computational efficiency. Many machine learning techniques, ranging from unsupervised to supervised, have been proposed to design compact hashing codes. Most of the existing hashing methods generate binary codes to efficiently find similar data examples to a query. However, the ranking accuracy among the retrieved data examples is not modeled. But in many real world applications, ranking measure is important for evaluating the quality of hashing codes. In this paper, we propose a novel Ranking Preserving Hashing (RPH) approach that directly optimizes a popular ranking measure, Normalized Discounted Cumulative Gain (NDCG), to obtain effective hashing codes with high ranking accuracy. The main difficulty in the direct optimization of NDCG measure is that it depends on the ranking order of data examples, which forms a non-convex non-smooth optimization problem. We address this challenge by optimizing the expectation of NDCG measure calculated based on a linear hashing function. A gradient descent method is designed to achieve the goal. An extensive set of experiments on two large scale datasets demonstrate the superior ranking performance of the proposed approach over several state-of-the-art hashing methods.

1 Introduction

Similarity search is an important problem in many machine learning applications. The purpose of similarity search is to identify similar data examples to a given query example. Due to the explosive growth of data on the Internet, a huge amount of data has been generated, which indicates that it is important to design efficient solutions of similarity search for large scale data. Traditional similarity search methods are difficult to be used directly for large scale data since computing the similarity using the original features (usually in high dimensional space) exhaustively between the query example and every candidate example is impractical for large applications. Recently, hashing methods [Liu *et al.*, 2013; Kong and Li, 2012a; Wang *et al.*, 2014a; Zhang and Li, 2014; Bergamo *et al.*, 2011; Kong and Li, 2012b; Xia *et al.*, 2014; Rastegari *et al.*, 2013; Lin *et al.*, 2014; Wang *et al.*, 2014c; Zhai *et al.*, 2013; Wang *et al.*, 2013c] have been proposed for fast similarity search in many large scale problems including document retrieval [Wang *et al.*, 2013b], object recognition [Torralba *et al.*, 2008], image matching [Strecha *et al.*, 2012], etc. These hashing methods design compact binary code in a low-dimensional space for each data example so that similar data examples are mapped to similar binary codes. In the retrieval process, these hashing methods first transform each query example into its corresponding binary code. Then similarity search can be simply conducted by calculating the Hamming distances between the codes of available data examples and the query, and selecting data examples within small Hamming distances. In this way, data examples are encoded and highly compressed within a low-dimensional binary space, which can usually be loaded in main memory and stored efficiently. The retrieval process can be conducted efficiently as the Hamming distance between two codes is simply the number of bits that differ and can be calculated using bitwise operation XOR. Existing hashing methods can be divided into two groups: unsupervised and semi-supervised/supervised.

Unsupervised hashing methods generate hashing codes without the requirement of supervised information (e.g., tags). Locality-Sensitive Hashing (LSH) [Datar *et al.*, 2004] is one of the most popular methods, which simply uses random linear projections to map data examples from a high dimensional Euclidean space to a low-dimensional binary space. The work in [Kulis and Grauman, 2009] extended LSH by exploiting kernel similarity for better retrieval efficacy. The Principle Component Analysis (PCA) Hashing [Lin *et al.*, 2010] method utilize the coefficients from the top k principal components to represent each example, and the coefficients are further binarized using the median value. Recently, Spectral Hashing (SH) [Weiss *et al.*, 2008] is proposed to design compact binary codes with balanced and uncorrelated constraints. Isotropic Hashing (IsoHash) [Kong and Li, 2012b] tries to learn an orthogonal matrix to make the data variance as equal as possible along each projection dimension. The work in [Wang *et al.*, 2015] proposes to learn the binary codes on structured data.

Semi-supervised or supervised hashing methods utilize

some supervised information such as semantic labels for generating effective hashing codes. Iterative Quantization (ITQ) method has been proposed in [Gong *et al.*, 2012] that treats the content features and tags as two different views, and the hashing codes are then learned by extracting a common space from these two views. This method has been extended to multi-view hashing [Gong *et al.*, 2013]. A semi-supervised hashing (SSH) method is proposed in [Wang *et al.*, 2010] which utilizes pairwise knowledge between data examples besides their content features for learning more effective hashing codes. A kernelized supervised hashing (KSH) framework proposed in [Liu *et al.*, 2012] imposes pairwise relationship between data examples to obtain hashing codes. More recently, a ranking-based supervised hashing (RSH) [Wang *et al.*, 2013a] method is proposed to leverage listwise ranking information to preserve the ranking order.

Although existing hashing methods have achieved promising results, very limited work explores the ranking accuracy, which is important for evaluating the quality of hashing codes in real world applications. Consider the following scenario: given a query example x_q and three relevant/similar data examples x_1, x_2, x_3 but with different relevance values as $r_1 > r_2 > r_3$ to the query. Most existing hashing methods only model the relevance of a data example to a query in a binary way, i.e., each example is either relevant to the query or irrelevant. In other words, these methods treat x_1, x_2 and x_3 as relevant examples to x_q with no difference. But in practice it will be more desirable if x_1 could be presented before x_2 and x_3 since it is more relevant to x_q than the other two. Some ranking based hashing methods [Wang *et al.*, 2013a; Yagnik *et al.*, 2011; Zhang *et al.*, 2013] have been recently proposed to improve the hashing code performance by modeling the ranking order with respect to relevance values. However, these methods do not differentiate the situations where $(r_1, r_2, r_3) = (3, 2, 1)$ and $(r_1, r_2, r_3) = (10, 2, 1)$ due to their identical ranking orders, i.e., $r_1 > r_2 > r_3$. But ideally, the Hamming distance between the learned hashing codes of x_1 and x_q should be smaller in the later situation than in the former one since the relevance value of x_1 to x_q is much larger in the later situation (10 versus 3). Therefore, these methods may fail to preserve the specific relevance values in the learned hashing codes, while the relevance values are important in evaluating the search accuracy.

This paper proposes a novel Ranking Preserving Hashing (RPH) approach that directly optimizes the popular ranking accuracy measure, Normalized Discounted Cumulative Gain (NDCG), to learn effective ranking preserving hashing codes that not only preserves the ranking order but also models the relevance values of data examples to the queries in the training data. The main difficulty in direct optimization of NDCG is that it depends on the rankings of data examples rather than their hashing codes, which forms a non-convex non-smooth objective. We then address this challenge by optimizing the expectation of NDCG measure calculated based on a linear hashing function to convert the problem into a smooth and differentiable optimization problem. A gradient descent method is applied to solve this relaxed problem. We conduct an extensive set of experiments on two large scale datasets of both images and texts to demonstrate the superior search accuracy of the proposed approach over several state-of-the-art hashing methods.

2 Ranking Preserving Hashing

2.1 Approach Overview

The proposed Ranking Preserving Hashing (RPH) approach via optimizing NDCG measure mainly contains three ingredients as shown in Figure 1: (1) Ground-truth relevance list to a query, which is constructed from the training data (the left part in Fig.1). (2) Ranking positions of data examples to a query, which are computed based on the hashing codes (the right part in Fig.1). (3) NDCG value, which measures the consistency between the ground-truth relevance list and the calculated ranking positions (the middle part in Fig.1). In other words, the more the hashing codes agree with the relevance list, the higher the NDCG value will be. Then the ranking preserving hashing codes are learned by optimizing the NDCG measure on the training data.

2.2 Problem Statement

We first introduce the problem of RPH. Assume there are n data examples in the dataset, denoted as: $\boldsymbol{X} = \{x_1, x_2, \ldots, x_n\} \in \mathbb{R}^{d \times n}$, where d is the dimensionality of the features. In addition, there is a query set $\boldsymbol{Q} = \{q_1, q_2, \ldots, q_m\}$ and for each query example q_j, we have a relevance list of n_j data examples from $\boldsymbol{X}$, which can be written as:

$$r(q_j, \boldsymbol{X}) = (r_1^j, r_2^j, \ldots, r_{n_j}^j) \tag{1}$$

where each element r_i^j represents the relevance of data example x_i^j to the query q_j. If $r_u^j > r_v^j$, it indicates that data example x_u^j is more relevant or more similar to q_j than x_v^j and x_u^j should rank higher than x_v^j. The goal is to obtain a linear hashing function $f : \mathbb{R}^d \to \{-1, 1\}^B$, which maps each data example x_i to its binary hashing code c_i (B is the number of hashing bits) to maximize the search/ranking accuracy. The linear hashing function is defined as:

$$c_i = f(x_i) = sgn(\boldsymbol{W} x_i) \tag{2}$$

where $\boldsymbol{W} \in \mathbb{R}^{B \times d}$ is the coefficient matrix representing the hashing function and sgn is the sign function. $c_i \in \{-1, 1\}^B$ is the binary hashing code of x_i.

Note that the ground-truth relevance list can be easily obtained if a relevance measure between data examples is predefined, e.g., l_2 distance in Euclidean space. On the other hand, if given the semantic label/tag information, it is also fairly straightforward to convert semantic labels to relevance values through counting the number of shared labels between the query and the data example.

2.3 Problem Formulation

Hashing methods are popularly used for large scale similarity search. As aforementioned, most of existing hashing methods only focus on retrieving all relevant or similar data examples to a given query without exploring the ranking accuracy.

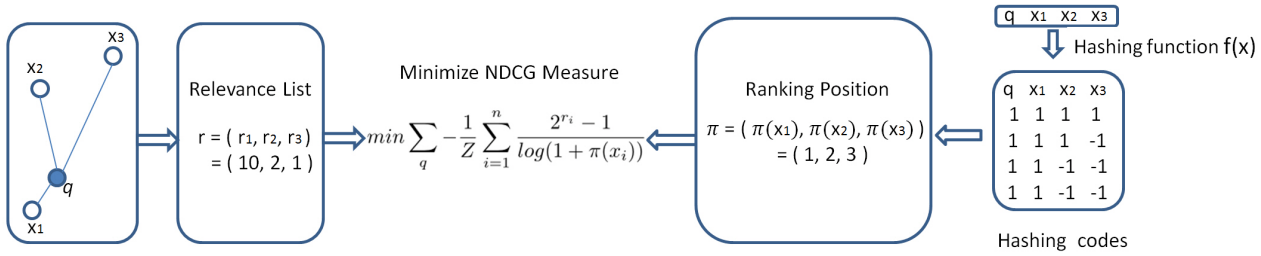

Figure 1: An overview of the proposed RPH approach.

However, in many real world applications, it is desirable and important to present a more relevant example to a query in front of a less relevant one. Different from existing hashing method, in this work, we propose to learn ranking preserving hashing codes that not only retrieve all possible relevant examples but at the same time preserve their rankings based on their relevance values to the query.

Given the binary hashing codes, the ranking positions of data examples to a query q are determined by the Hamming distances between their hashing codes and the query code. Specifically, if a data example is similar or relevant to a query, then their Hamming distance should be small. In other words, the higher the rank of a data example to a query, the smaller the Hamming distance between the hashing codes is. The Hamming distance between two binary hashing codes is given by the number of bits that are different between them and can be calculated as:

$$Ham(c_q, c_i) = \frac{1}{4}\|c_q - c_i\|^2 = \frac{1}{2}(B - c_q^T c_i) \tag{3}$$

Then the ranking position $\pi(x_i)$ can be calculated as:

$$\begin{aligned}\pi(x_i) &= 1 + \sum_{k=1}^{n} I\left(Ham(c_q, c_i) > Ham(c_q, c_k)\right) \\ &= 1 + \sum_{k=1}^{n} I\left(c_q^T(c_k - c_i) > 0\right)\end{aligned} \tag{4}$$

where $I(s)$ is the indicator function that outputs 1 when statement s is true and 0 otherwise. Intuitively, the ranking position of a data example to a query is equivalent to 1 plus the number of data examples whose hashing codes are closer to the query code.

In order to achieve high ranking quality hashing codes, we want the ranking positions calculated in the Hamming space in Eqn.4 to be consistent with the ground-truth relevance list in Eqn.1. Then a natural question to ask is how to measure the ranking consistency? In this paper, we use a well-known measure, Normalized Discounted Cumulative Gain (NDCG) [Qin *et al.*, 2010; Wang *et al.*, 2013d] which is widely applied in many information retrieval and machine learning applications, to evaluate the ranking consistency as:

$$NDCG = \frac{1}{Z}\sum_{i=1}^{n}\frac{2^{r_{\pi^{-1}(i)}}-1}{log(1+i)} = \frac{1}{Z}\sum_{i=1}^{n}\frac{2^{r_i}-1}{log(1+\pi(x_i))} \tag{5}$$

where Z is the normalization factor so that the maximum value of NDCG is 1, which can be calculated by ranking the examples based on their relevance to the query. $\pi(x_i)$ is the ranking position of x_i to the query based on the Hamming distance of their hashing codes and $\pi^{-1}(i)$ denotes the data example at i-th ranking position. r_i is the corresponding relevance value. $\frac{1}{log(1+i)}$ can be viewed as the weight of the i-th rank data example, which indicates that NDCG emphasizes the importance of the higher ranked data examples than those examples with lower ranks. Therefore, NDCG is usually truncated at a particular rank level (e.g., top K retrieved examples) instead of all n examples. From the above definition of NDCG, it can be seen that the larger the NDCG value is, the more the hashing codes agree with the relevance list, and the maximal NDCG value is obtained when the ranking positions of data examples are completely consistent with their relevance values to the query. By optimizing the NDCG measure, the learned hashing function not only preserves the ranking order of the data examples but also ensures that the hashing codes are consistent with the relevance values in the training data. Then the entire objective is to minimize the negative summation of NDCG values on all training queries:

$$J(\boldsymbol{W}) = -\sum_{j=1}^{m}\frac{1}{Z_j}\sum_{i=1}^{n_j}\frac{2^{r_i^j}-1}{log(1+\pi_j(x_i^j))} \tag{6}$$

Directly minimizing the objective function in Eqn.6 is intractable since it depends on the ranking positions of data examples (Eqn.4), resulting in a non-convex non-smooth optimization problem. We then address this challenge by using the expectation of ranking position $\hat{\pi}_j(x_i^j)$ instead of

$\pi_j(x_i^j)$ as:

$$\begin{aligned}\hat{\pi}_j(x_i^j) &= 1 + E\left[\sum_{k=1}^{n} I(c_{q_j}^T(c_k - c_i) > 0)\right] \\ &= 1 + \sum_{k=1}^{n} Pr\left(c_{q_j}^T(c_k - c_i) > 0\right)\end{aligned} \tag{7}$$

where $Pr(c_{q_j}^T(c_k - c_i) > 0)$ means the probability that the ranking position of data example x_k is higher than the position of x_i to query q_j and we use a logistic function to model this probability as:

$$\begin{aligned}Pr\left(c_{q_j}^T(c_k - c_i) > 0\right) &= \frac{1}{1 + exp(-c_{q_j}^T(c_k - c_i))} \\ = \frac{1}{1 + exp(-sgn(\boldsymbol{W}q_j)^T(sgn(\boldsymbol{W}x_k) - sgn(\boldsymbol{W}x_i)))}\end{aligned} \tag{8}$$

The motivation of the derivation in Eqn.7 and Eqn.8 is that we approximate the intractable optimization for NDCG with a tractable probabilistic framework. Firstly, the ranking position of each data example can be calculated exactly based on Eqn.4. However, due to the intractability, we model the problem in a probabilistic framework by computing the expectation of the ranking position. The using of expectation to represent the true ranking position is widely adopted in learning to rank approaches due to its good probability approximation and computational tractability. Secondly, the using of logistic function in Eqn.8 to model the probability is based on the intuition that a data example should be ranked higher if its hashing code is closer to the query. There are also other alternatives to model the probability. Due to the popularity of logistic function used in learning to rank, we adopt it in our formulation.

The above probability function is still non-differentiable with respect to $\boldsymbol{W}$ due to the embedded sign function. Therefore, as suggested in [Wang *et al.*, 2013a; 2014b], we drop off the sign function and use the signed magnitude in the probability function as:

$$Pr\left(c_{q_j}^T(c_k - c_i) > 0\right) = \frac{1}{1 + exp(-q_j^T\boldsymbol{W}^T\boldsymbol{W}(x_k - x_i))} \tag{9}$$

By substituting the expected ranking position into the NDCG measure, the final objective in Eqn.6 can be rewritten as:

$$\begin{aligned}\min\ J(\boldsymbol{W}) &= -\sum_{j=1}^{m}\frac{1}{Z_j}\sum_{i=1}^{n_j}\frac{2^{r_i^j} - 1}{log(1 + \hat{\pi}_j(x_i^j))} \\ s.t. \quad & \boldsymbol{W}\boldsymbol{W}^T = \boldsymbol{I}\end{aligned} \tag{10}$$

where $\boldsymbol{W}\boldsymbol{W}^T = \boldsymbol{I}$ is the orthogonality constraint which ensures the learned hashing codes to be uncorrelated with each other and hold least redundant information.

2.4 Optimization

We first convert the hard constraint into a soft penalty term by adding a regularizer to the objective. The reason is that most of the variance is contained in a few top projections for many real world datasets. The orthogonality constraint forces hashing methods to choose those directions with very low variance progressively, which may substantially reduce the quality of hashing codes. This issue is also pointed out in [Liu *et al.*, 2012; Wang *et al.*, 2012]. Therefore, instead of adding hard orthogonality constraint, we impose a soft orthogonality/penalty term as:

$$J(\boldsymbol{W}) = -\sum_{j=1}^{m}\frac{1}{Z_j}\sum_{i=1}^{n_j}\frac{2^{r_i^j} - 1}{log(1 + \hat{\pi}_j(x_i^j))} + \alpha\|\boldsymbol{W}\boldsymbol{W}^T - \boldsymbol{I}\|_F^2 \tag{11}$$

where α is a trade-off parameter to balance the weights between the two terms. Although the objective in Eqn.11 is still non-convex, it is smooth and differentiable which enables gradient descent methods to be applied for efficient optimization. The gradients of the two terms with respect to $\boldsymbol{W}$ are given below:

$$\begin{aligned}\frac{d\hat{\pi}_j(x_i^j)}{d\boldsymbol{W}} = &\sum_{k=1}^{n_j} exp(-q_j^T\boldsymbol{W}^T\boldsymbol{W}(x_k - x_i)) \\ &\frac{\boldsymbol{W}\left((x_k - x_i)q_j^T + q_j(x_k - x_i)^T\right)}{(1 + exp(-q_j^T\boldsymbol{W}^T\boldsymbol{W}(x_k - x_i)))^2}\end{aligned} \tag{12}$$

$$\frac{d\|\boldsymbol{W}^T\boldsymbol{W} - \boldsymbol{I}\|_F^2}{d\boldsymbol{W}} = 4\boldsymbol{W}^T(\boldsymbol{W}\boldsymbol{W}^T - \boldsymbol{I}) \tag{13}$$

Then the gradient of $\frac{dJ(\boldsymbol{W})}{d\boldsymbol{W}}$ can be computed by combining the above two gradients with some additional mathematical calculation. With this obtained gradient, L-BFGS quasi-Newton method [Liu and Nocedal, 1989] is applied to solve the optimization problem. The full RPH approach is summarized in Algorithm 1.

Algorithm 1 Ranking Preserving Hashing (RPH)

Input: Training examples $\boldsymbol{X}$, query examples $\boldsymbol{Q}$ and parameters α.

Output: Hashing function $\boldsymbol{W}$ and hashing codes $\boldsymbol{C}$.

1: Compute the relevance vector r_i^j in Eqn.1.
2: Initialize $\boldsymbol{W}$.
3: **repeat** Gradient Descent
4: Compute the gradient in Eqn.12.
5: Compute the gradient in Eqn.13.
6: Update $\boldsymbol{W}$ by optimizing the objective function.
7: **until** the solution converges
8: Compute the hashing codes $\boldsymbol{C}$ using Eqn.2.

2.5 Discussion

The idea of modeling the NDCG measure to maximize the search/ranking accuracy is also utilized in learning to rank [Valizadegan *et al.*, 2009; Weimer *et al.*, 2007]. However, these learning to rank methods are not based on binary hashing codes, but on learning effective document permutation. Unlike in our formulation, the NDCG measure modeled in learning to rank methods does not involve linear-projection based hashing function, on which the ranking

	NUSWIDE			$Flickr1m$		
Methods	NDCG@5	NDCG@10	NDCG@20	NDCG@5	NDCG@10	NDCG@20
RPH	**0.257**	**0.249**	**0.234**	**0.313**	**0.298**	**0.283**
RSH	0.242	0.238	0.226	0.288	0.271	0.259
KSH	0.223	0.217	0.198	0.265	0.252	0.237
SSH	0.216	0.209	0.195	0.251	0.242	0.230
SH	0.193	0.185	0.172	0.250	0.234	0.221

Table 1: Results of NDCG@K using $Hamming\ Ranking$ on both datasets, with 64 hashing bits.

position is determined. Moreover, we need to find the expected ranking position of each data example according to the Hamming distance between the hashing codes, which is very different to learning to rank methods.

The learning algorithm of RPH for deriving the optimal hashing function is fairly fast. During each iteration of the gradient descent method, we need to compute the gradients in Eqns.12 and 13, which involves some matrix multiplications. The complexity for calculating the gradient in Eqn.12 is bounded by O(mn_jdB) since both $\boldsymbol{W}(x_k - x_i)$ and $\boldsymbol{W}(x_k - x_i)q_j^T$ requires O(dB). The complexity for calculating the gradient in Eqn.13 is simply O(d^2B) which only involves $\boldsymbol{W}\boldsymbol{W}^T$. Therefore, the total complexity of each iteration of the gradient descent method is O($m\hat{n}dB + d^2B$) and the learning algorithm is fairly scalable since its time complexity is linear in the number of training queries m and the average number of data examples $\hat{n}$ associated with each query.

3 Experiment

3.1 Datasets and Setting

We evaluate proposed research on two image benchmarks: *NUSWIDE* and $Flickr1m$, which have been widely used in the evaluation of hashing methods. *NUSWIDE*[1] [Chua *et al.*, 2009] is created by NUS lab for evaluating image retrieval techniques. It contains $270k$ images associated with about $5k$ different tags. We use a subset of $110k$ image examples with the most common $1k$ tags in our experiment. $Flickr1m$[2] [Huiskes *et al.*, 2010] is collected from Flicker images for image annotation and retrieval tasks. This dataset contains 1 million image examples associated with more than $7k$ unique semantic tags. A subset of $250k$ image examples with the most common $1k$ tags is used in our experiment. 512-dimensional GIST descriptors [Oliva and Torralba, 2001] are used as image features. Since both datasets are associated with multiple semantic labels/tags, the ground-truth relevance values can be naturally derived based on the number of shared semantic labels between data examples.

We implement our algorithm using Matlab on a PC with Intel Duo Core i5-2400 CPU 3.1GHz and 8GB RAM. The parameter α is tuned by cross validation through the grid $\{0.01, 0.1, 1, 10, 100\}$ and we will discuss more details on how it affects the performance of our approach later. For each experiment, we randomly choose $1k$ examples as testing queries. Within the remaining data examples, we randomly sample 500 training queries and for each query, we randomly sample 1000 data examples to construct the ground-truth relevance list. We will discuss the performance with different number of training queries later in our experiments. Finally, we repeat each experiment 10 times and report the result based on the average over the 10 runs.

The proposed RPH approach is compared with four different hashing methods, including Spectral Hashing (SH) [Weiss *et al.*, 2008], Semi-Supervised Hashing (SSH) [Wang *et al.*, 2010], Kernel Supervised Hashing (KSH) [Liu *et al.*, 2012] and Ranking-based Supervised Hashing (RSH) [Wang *et al.*, 2013a]. SH is an unsupervised method and does not use any label information. We use the standard settings in [Weiss *et al.*, 2008] in our experiments. For SSH and KSH, we randomly sample $2k$ data examples and use their ground-truth labels to generate pairwise similarity matrix as part of the training data. Gaussian RBF kernel is used in KSH. To get a fair comparison, for RSH, we randomly sample 500 query examples and 1000 data examples to compute the ground-truth ranking lists.

3.2 Evaluation Metrics

To conduct fair evaluation, we follow two criteria which are commonly used in the literature: $Hamming\ Ranking$ and $Hash\ Lookup$. $Hamming\ Ranking$ ranks all the candidate examples according to their Hamming distance from the query and the top K examples are returned as the desired neighbors. We use NDCG@K to evaluate the ranking quality of the top K retrieved examples. $Hash\ Lookup$ returns all the examples within a certain Hamming radius r of the query. We use average cumulative gain (ACG) to measure the quality of these returned examples, which is calculated as: $ACG_r = \frac{1}{|N_r|}\sum_{x_i \in N_r} r_i$, where N_r is the set of the retrieved data examples within a Hamming radius r and r_i is the relevance value of a retrieved data example x_i. A hamming radius of $r = 2$ is used to retrieve the neighbors in the experiments.

3.3 Results and Discussion

We first report the results of NDCG@5, NDCG@10 and NDCG@20 of different hashing methods using $Hamming\ Ranking$ on two datasets with 64 hashing bits in Table 1. From these comparison results, it can be seen that RPH gives the overall best performance among all five hashing methods on both datasets. For example, the performance of our method boosts about 4.6% on *NUSWIDE* dataset, with 9.9% improvement on $Flickr1m$ dataset compared to RSH

[1] http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm

[2] http://press.liacs.nl/mirflickr/

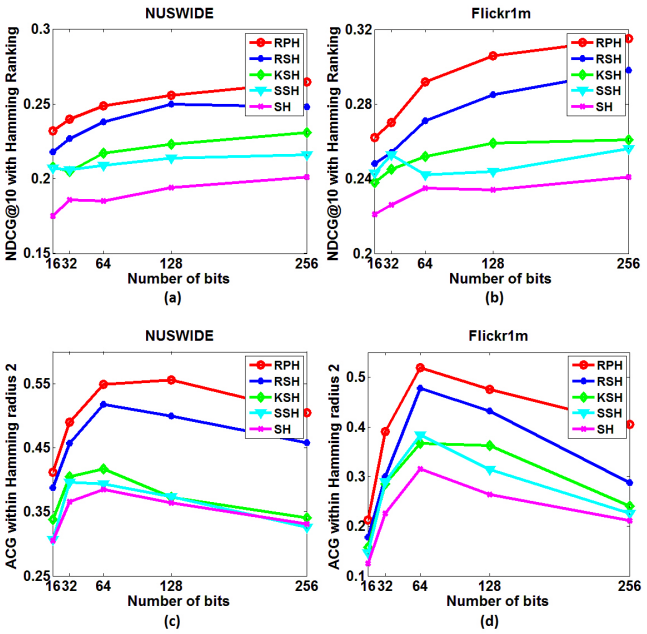

Figure 2: Performance evaluation on both datasets with different number of hashing bits. (a)-(b): NDCG@10 using *Hamming Ranking*. (c)-(d): ACG with Hamming radius 2 using *Hash Lookup*.

under NDCG@10 measure. We can see from Table 1 that SH does not perform well in all cases. This is because SH is an unsupervised hashing method which does not utilize any supervised information into learning hashing codes. For methods SSH and KSH, they both achieve better results than SH since these methods incorporate some pairwise knowledge between data examples in addition to the content features for learning effective hashing codes. However, the ranking order is not preserved in the learned hashing codes of these two methods and thus, the ranking-based supervised hashing method RSH which models the listwise ranking information can generate more accurate hashing codes with larger NDCG values than SSH and KSH. On the other hand, RPH method substantially outperforms RSH since it directly optimizes the NDCG measure to learn high quality hashing codes that not only preserve the ranking order but also preserve the relevance values of data examples to the query in the training data. Therefore, the search/ranking accuracy can be maximized which is coincides with our expectation.

The second set of experiments evaluate the performance of different hashing methods by varying the number of hashing bits in the range of $\{16, 32, 64, 128, 256\}$. The results of NDCG@10 using *Hamming Ranking* on both datasets are reported in Fig.2(a)-(b), with the ACG results of Hamming radius 2 using *Hash Lookup* shown in Fig.2(c)-(d). Not surprisingly, from Fig.2(a)-(b) we can see that the performance of different methods improves when the number of hashing bits increases from 16 to 256 and our RPH method outperforms the other compared hashing methods which is consistent with the results in Table 1. However, we can also observe from Fig.2(c)-(d) that the ACG result

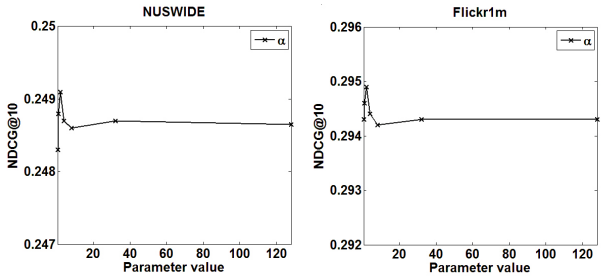

Figure 3: Parameter sensitivity results of NDCG@10 on both datasets with 64 hashing bits.

of most compared methods decreases when the number of hashing bits increases after 64. The reason is that when using longer hashing bits, the Hamming space becomes increasingly sparse and very few data examples fall within the Hamming ball of radius 2, resulting in many queries with empty returns (we count the ACG as zero in this case). Similar behavior is also observed in [Wang *et al.*, 2013a; 2014b]. In this situation, however, the ACG results of RPH are still consistently better than other baselines.

To prove the robustness of the proposed method, we conduct parameter sensitivity experiments on both datasets. In each experiment, we tune the trade-off parameter α from the grid $\{0.5, 1, 2, 4, 8, 32, 128\}$. We report the results of NDCG@10 with 64 hashing bits in Fig.3. It is clear from these experimental results that the performance of RPH is relatively stable with respect to α in a wide range of values. The results also prove that using soft penalty with an appropriate weight parameter is better than enforcing the hard orthogonality constraint (corresponds to infinite α).

4 Conclusion

This paper proposes a novel Ranking Preserving Hashing (RPH) approach that directly optimizes the ranking accuracy measure, Normalized Discounted Cumulative Gain (NDCG). We handle the difficulty of non-convex non-smooth optimization by using the expectation of NDCG measure calculated based on the linear hashing function and then solve the relaxed smooth optimization problem with a gradient descent method. Experiments on two large scale datasets demonstrate the superior performance of the proposed approach over several state-of-the-art hashing methods. In future, we plan to investigate generalization error bound for the proposed learning method. We also plan to apply some sequential learning approach to accelerate the training speed.

5 Acknowledgments

This work is partially supported by NSF research grants IIS-0746830, DRL-0822296, CNS-1012208, IIS-1017837, CNS-1314688 and a research grant from Office of Naval Research (ONR-11627465). This work is also partially supported by the Center for Science of Information (CSoI), an NSF Science and Technology Center, under grant agreement CCF-0939370.

References

[Bergamo *et al.*, 2011] Alessandro Bergamo, Lorenzo Torresani, and Andrew W. Fitzgibbon. Picodes: Learning a compact code for novel-category recognition. In *NIPS*, pages 2088–2096, 2011.

[Chua *et al.*, 2009] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yantao Zheng. Nus-wide: a real-world web image database from national university of singapore. In *CIVR*, 2009.

[Datar *et al.*, 2004] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computational Geometry*, pages 253–262, 2004.

[Gong *et al.*, 2012] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE TPAMI*, 2012.

[Gong *et al.*, 2013] Yunchao Gong, Qifa Ke, Michael Isard, and Svetlana Lazebnik. A multi-view embedding space for modeling internet images, tags, and their semantics. *IJCV*, 2013.

[Huiskes *et al.*, 2010] Mark J. Huiskes, Bart Thomee, and Michael S. Lew. New trends and ideas in visual concept detection: the mir flickr retrieval evaluation initiative. In *Multimedia Information Retrieval*, pages 527–536, 2010.

[Kong and Li, 2012a] Weihao Kong and Wu-Jun Li. Double-bit quantization for hashing. In *AAAI*, 2012.

[Kong and Li, 2012b] Weihao Kong and Wu-Jun Li. Isotropic hashing. In *NIPS*, pages 1655–1663, 2012.

[Kulis and Grauman, 2009] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, pages 2130–2137, 2009.

[Lin *et al.*, 2010] Ruei-Sung Lin, David A. Ross, and Jay Yagnik. Spec hashing: Similarity preserving algorithm for entropy-based coding. In *CVPR*, pages 848–854, 2010.

[Lin *et al.*, 2014] Guosheng Lin, Chunhua Shen, and Jianxin Wu. Optimizing ranking measures for compact binary code learning. In *ECCV*, pages 613–627, 2014.

[Liu and Nocedal, 1989] Dong C. Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989.

[Liu *et al.*, 2012] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *CVPR*, pages 2074–2081, 2012.

[Liu *et al.*, 2013] Xianglong Liu, Junfeng He, and Bo Lang. Reciprocal hash tables for nearest neighbor search. In *AAAI*, 2013.

[Oliva and Torralba, 2001] Aude Oliva and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV*, 42(3):145–175, 2001.

[Qin *et al.*, 2010] Tao Qin, Tie-Yan Liu, and Hang Li. A general approximation framework for direct optimization of information retrieval measures. *Inf. Retr.*, 13(4):375–397, 2010.

[Rastegari *et al.*, 2013] Mohammad Rastegari, Jonghyun Choi, Shobeir Fakhraei, Daume Hal, and Larry S. Davis. Predictable dual-view hashing. In *ICML (3)*, pages 1328–1336, 2013.

[Strecha *et al.*, 2012] Christoph Strecha, Alexander A. Bronstein, Michael M. Bronstein, and Pascal Fua. Ldahash: Improved matching with smaller descriptors. *IEEE TPAMI*, 34(1):66–78, 2012.

[Torralba *et al.*, 2008] Antonio Torralba, Robert Fergus, and William T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE TPAMI*, 30(11):1958–1970, 2008.

[Valizadegan *et al.*, 2009] Hamed Valizadegan, Rong Jin, Ruofei Zhang, and Jianchang Mao. Learning to rank by optimizing ndcg measure. In *NIPS*, pages 1883–1891, 2009.

[Wang *et al.*, 2010] Jun Wang, Ondrej Kumar, and Shih-Fu Chang. Semi-supervised hashing for scalable image retrieval. In *CVPR*, pages 3424–3431, 2010.

[Wang *et al.*, 2012] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Semi-supervised hashing for large-scale search. *IEEE TPAMI*, 34(12):2393–2406, 2012.

[Wang *et al.*, 2013a] Jun Wang, Wei Liu, Andy Sun, and Yu-Gang Jiang. Learning hash codes with listwise supervision. In *ICCV*, 2013.

[Wang *et al.*, 2013b] Qifan Wang, Dan Zhang, and Luo Si. Semantic hashing using tags and topic modeling. In *SIGIR*, pages 213–222, 2013.

[Wang *et al.*, 2013c] Qifan Wang, Dan Zhang, and Luo Si. Weighted hashing for fast large scale similarity search. In *CIKM*, pages 1185–1188, 2013.

[Wang *et al.*, 2013d] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, and Tie-Yan Liu. A theoretical analysis of ndcg type ranking measures. In *COLT*, pages 25–54, 2013.

[Wang *et al.*, 2014a] Qifan Wang, Bin Shen, Shumiao Wang, Liang Li, and Luo Si. Binary codes emmbedding for fast image tagging with incomplete labels. In *ECCV*, 2014.

[Wang *et al.*, 2014b] Qifan Wang, Luo Si, and Dan Zhang. Learning to hash with partial tags: Exploring correlation between tags and hashing bits for large scale image retrieval. In *ECCV*, pages 378–392, 2014.

[Wang *et al.*, 2014c] Qifan Wang, Luo Si, Zhiwei Zhang, and Ning Zhang. Active hashing with joint data example and tag selection. In *SIGIR*, 2014.

[Wang *et al.*, 2015] Qifan Wang, Luo Si, and Bin Shen. Learning to hash on structured data. In *AAAI*, 2015.

[Weimer *et al.*, 2007] Markus Weimer, Alexandros Karatzoglou, Quoc V. Le, and Alex J. Smola. Cofi rank - maximum margin matrix factorization for collaborative ranking. In *NIPS*, 2007.

[Weiss *et al.*, 2008] Yair Weiss, Antonio Torralba, and Robert Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008.

[Xia *et al.*, 2014] Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. Supervised hashing for image retrieval via image representation learning. In *AAAI*, pages 2156–2162, 2014.

[Yagnik *et al.*, 2011] Jay Yagnik, Dennis Strelow, David A. Ross, and Ruei-Sung Lin. The power of comparative reasoning. In *ICCV*, pages 2431–2438, 2011.

[Zhai *et al.*, 2013] Deming Zhai, Hong Chang, Yi Zhen, Xianming Liu, Xilin Chen, and Wen Gao. Parametric local multimodal hashing for cross-view similarity search. In *IJCAI*, 2013.

[Zhang and Li, 2014] Dongqing Zhang and Wu-Jun Li. Large-scale supervised multimodal hashing with semantic correlation maximization. In *AAAI*, pages 2177–2183, 2014.

[Zhang *et al.*, 2013] Lei Zhang, Yongdong Zhang, Jinhui Tang, Ke Lu, and Qi Tian. Binary code ranking with weighted hamming distance. In *CVPR*, pages 1586–1593, 2013.