IIIT Hyderabad at Million Query Track TREC 2009

Prashant Ullegaddi, Sudip Datta, Vinay Pande, Kushal Dave, Vasudeva Varma

{prashant.ullegaddi, sudip.datta, vinay.pande, kushal.dave}@research.iiit.ac.in
vv@iiit.ac.in
Search and Information Extraction Lab,
International Institute of Information Technology,
Hyderabad-500032.

Abstract

This was our maiden attempt at Million Query track, TREC 2009. We submitted three runs for ad-hoc retrieval task in Million Query track. We explored ad-hoc retrieval of web pages using Hadoop—a distributed infrastructure. To enhance recall, we expanded the queries using WordNet and also by combining the query with all possible subsets of tokens present in the query. To prevent query drift we experimented on giving selective boosts to different steps of expansion including giving higher boosts to sub-queries containing named entities as opposed to those that did not. In fact, this run achieved highest precision among our other runs. Using simple statistics we identified authoritative domains such as wikipedia.org, answers.com, etc and attempted to boost hits from them, while preventing them from overly biasing the results. An attempt to query classification was also made.

1 Introduction

The main goals of Million Query track in TREC 2009 were 1) To explore ad-hoc retrieval and classify the queries as precision-oriented/recall-oriented queries, 2) To investigate the reusability of the relevance judgments. To evaluate ad-hoc task a set of 40K queries were provided. First 1000 queries were of the highest priority. And every system was required provide results for the first 1000 queries compulsorily. We explored ad-hoc retrieval by designing a distributed system built using Lucene[1] and Hadoop[2] which will be discussed in Section 2.

We started off by exploiting the finer details embedded in a document structure. We had extracted various fields of a page (e.g., title, bold, href, etc.). We initially tried using this information in ranking the result (weighing terms in title higher, and so on). But the results were not too encouraging. Finally, we retained three fields: document-url, title and body after parsing the content.

Few experiments on initial search results showed that a lot of spam/duplicate pages were present in the data set especially in the top hits. When we checked with the contents of those pages, it was found that there was a negligible difference between the duplicate documents. Due to this, most of the results in the top hits looked similar and had similar content. To overcome this, We employed a technique called deduplication explained in Section 3.1 to remove duplicates from the result set, resulting in more diversified results. The main motivation behind this was any user would not want to see very similar (in fact the same) results in top hits. In all we tried to model our retrieval system as a web search engine.

Another area we concentrated on was *query expansion*. Since majority of the queries were short, a query expansion module had to be designed. We also experimented with proper nouns in query expansion. Proper nouns in a query are important than any other query terms for they seem to carry more information. In expanded query, sub-queries not containing any of proper noun terms should have lower weightage (as they are more likely to result in *query drift*) as opposed to ones containing one or more of these terms. E.g. consider query "University of Hyderabad", here proper noun "Hyderabad" carries more information. "Hyderabad" together with "University" makes the above query complete. Hence, here query term "Hyderabad" is an important query term and should be weighed high. We employed a simple heuritsic to identify all proper nouns in the query log as explained in Section 3.2.

Finally, we also experimented with giving additional boost (document level boost) to authoritative pages among the search results. The webgraph pro-

[1] http://lucene.apache.org/
[2] http://hadoop.apache.org/

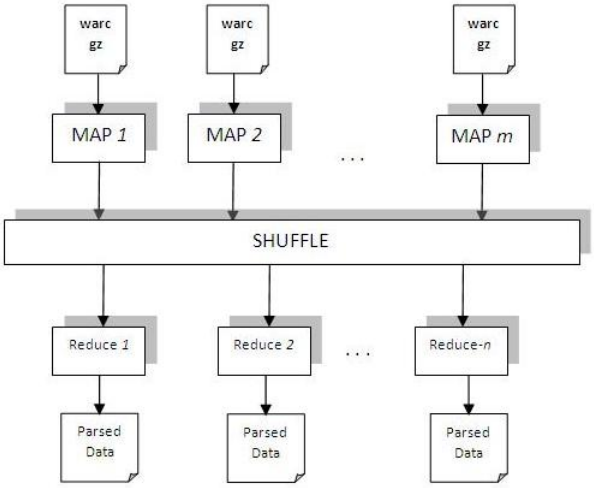

Figure 1: Stage 1— Parsing of HTML pages on cluster–One time job

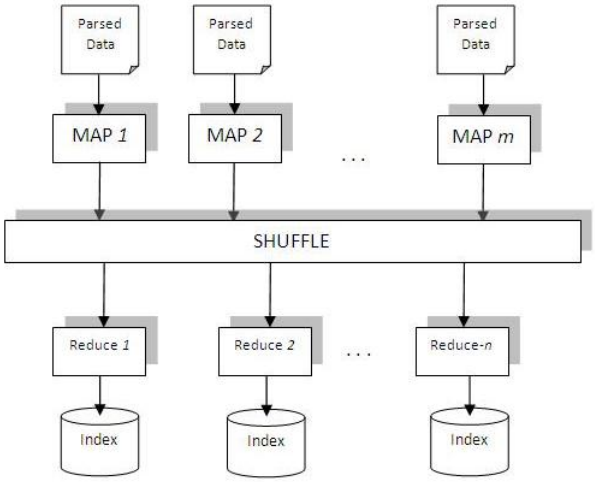

Figure 2: Stage 2—Indexing on Hadoop cluster

vided with the dataset was not a complete graph as many outlinks were pointing to pages outside the corpus. Hence pageranking was not found to be too usefull. We needed another way to devise a way to find authoritative pages in the corpus over different domains. We used another method as discussed in Section 3.3 to find authoritative pages and boost them in the search results.

We also attempted the original *query classification* task of classifying queries into 4 classes - Hard Precision, Soft Precision, Soft Recall, Hard Recall. To determine the viability of finding a global consensus and possible rules that could later be used to build a classifier. Four of us independently tagged the first 500 queries, but observed large inter-annotator disagreement both in terms of rules and labels. Thus we concluded that the task of classifying queries into precision and recall is very subjective and even harder to generalize than classifying queries into navigational and informational.

The rest of the paper explains in detail the approaches we followed.

2 System Design

The data set consisted of 50 million HTML web pages, with some long web pages truncated to 256KB. The data set size was just over 1TB. Indexing 1TB data set (50million documents) and handling truncated pages required a careful system design. Truncation caused a major problem in parsing the data as most of the HTML parsers were vulnerable to missing tags/malformed tags. We needed a robust HTML parser and after trying various publicly available parser, selected Jericho[3] HTML parser.

We used Lucene + Hadoop combination in a similar fashion to Nutch[4] architecture, for indexing using Map/Reduce [2] paradigm. Here we describe Hadoop cluster which was used to index the data set. We used a cluster of 9 nodes. Each node was a quad core machine with 4GB RAM. One node was specially designated as the master, which only took care of assigning tasks to the remaining nodes i.e., slaves. Each slave node was set to run 8 maps and 4 reduces. With this setting we could index the whole data in less than 4 hours which otherwise would have taken a day to index!

The architecture used for parsing and indexing the data is shown in figure 1 and figure 2. The distributed infrastructure we employed for indexing the data consisted of two stages:

- Stage 1: Parsing—Every map read a WARC file, and parsed it with WARC reader. Once HTML pages were extracted, they were parsed with Jericho HTML parser, and saved on HDFS as a key/value pair SequenceFile where key was clueweb-id of the page, and value was the parsed content of the page . The architecture for parsing on Hadoop is shown in figure 1.

- Stage 2: Indexing—The parsed data in first stage was fed to indexer. We had 8 reduces and each used Lucene to creat 4 indexes of the data it processed. In all we had 32 indexes of the data

[3]http://jerichohtml.sourceforge.net
[4]http://www.nutch.org/

created by eight reduces. The architecture for indexing on Hadoop is shown in figure 2.

- Stage 3: Merging—We then merged these 32 indexes into 8 indexes so that they could be searched in distributed manner by placing each of the 8 indexes on individual machines. We used RMI search to get result from each machine and then merged results on a server. In this way we could substantially reduce the search time.

3 Experiments

In this section, we present the various approaches we followed and the experiments we conducted.

3.1 De-duplication

The TREC web corpus contained many duplicate pages having almost the same title and content but slightly different URLs. We refer to process of handling these duplicates as De-duplication. When the search results were analyzed without de-duplication it was found that the results contained many duplicates clustered together and there was no diverseness in the top results. For Example: searching for the string "naïve bayes" gave many wikipedia pages that had the same content and similar URLs. As all these duplicate results have scores very close to each other, they come in a group in search results. This situation is not desirable in the domain of Web search as it results in large redundancy in search hits. Another ill-effect of this redundancy is that other important results fail to figure among the top hits.

3.1.1 Duplicate Detection

As the duplicates have scores very close to each other, the search for duplicates is carried out only in the vicinity of the result for which duplicates are checked. Hence, search for duplicates is done in a window of 50, where the first result(in the group of duplicates) is kept as it has the highest score amongst all the duplicates and rest are treated as duplicates.

We used following heuristics to detect duplicates:

1. The string edit distance of the URLs of the two results was less than a threshold (was set to 10).

2. If the difference in scores of the two duplicates was less than a threshold (set to 0.001).

3. The titles of the duplicates match exactly.

The duplicates were searched in a sliding window where one result (the first) was checked for duplicate with the rest of the results in the window. There are two approaches that were followed to tackle the problem:

1. Removing duplicates: As the duplicates do not contribute anything to the result set, the duplicates are simply deleted from the result set. As the maximum number of results to be submitted per query was 1000, deleting duplicates gave chance to any relevant result outside the first 1000 results. This is the approach followed in all the runs.

2. Demoting duplicates: Another approach that can be followed is demoting the results. In this approach duplicates are penalized and are moved down the order. The process of demotion has to be in such a way that the duplicates are disperesed down the results while other hits from the top 1000 are promoted up. Descending all the duplicates by a Fibonacci sequence (2, 3, 5, 8 . . .) is a natural choice for such an application. For example, the first duplicate is demoted by 2 positions from its current position, the next by 3, then next by 5 and so on. For some duplicates the position to which the duplicate descended can reach 1000, hence they should be promoted up from the position 1000 again by a Fibonacci sequence.

3.2 Query Expansion using Lucene

We search the query terms in different fields of documents like "url", "title", "h1", "bold", "text", etc. We observed that if query terms occured in url or title of the document, then chances of document being relevant are high. Hence, we gave more boost to documents having query terms in their url and title. For rest of the fields we assigned weights in decreasing order of their importance.

For query expansion, we generated a large number of simple queries containing query terms from the original query and process them separately. After executing all the subqueries we obtained final result by combining all results after giving differential boosts to results from individual queries. Procedure for creating subqueries is stated below:

1. We assume that query term order and proximity are very important for finding relevant documents. Hence, we did exact search of query terms in the documents. Documents having exact query terms in url and titles have more weigh-

tage than documents with exact terms in other fields of the document.

2. We also search query term in approximate vicinity by using window of three words. Here, we assume that query terms can occur toghether in any order within the window of three words containing other words with query terms.

3. To increase recall, we also constructed queries from all the possible subsets of tokens (except the stop words) contained in the query. As an ill-effect, this could result in large query drift. We observed that for queries containing noun terms, subqueries which do not contain any of the noun terms result in very noisy results. Further, if the queries contain 'proper nouns', giving higher weightage to results from subqueries containing these terms, results in further containment of quert drift. We used WordNet [3] to identify noun terms. Any term not contained in WordNet is likely to either be a 'proper noun' or a mis-spelling. We employed this hypothesis as a first step to identifying proper nouns. Besides, we manually short-listed proper nouns from the list of words marked as nouns by WordNet. This query expansion technique could be more effectively employed if there exist other efficient ways to identify 'proper nouns'.

To decide weightage among subqueries containing nouns we use a simple formula to provide more boost to subqueries containing more proper nouns than query containing mixture of nouns and proper noun terms. The weightage assigned to a sub-query Q_{sub} is given by:

$$Weightage(Q_{sub}) = \frac{|Q_{sub}|}{|Q|} \cdot \frac{|P_{sub}|}{|P|} \cdot B_f$$

where,

$|Q_{sub}|$ is number of terms in subquery

$|Q|$ is total number of terms in original query

$|P_{sub}|$ is number of pronouns in subquery

$|P|$ is total number of proper nouns in original query

B_f is boost set for field 'f '

From this formula, it is clear that sub-query containing more number of terms and more number of proper nouns gets more weightage than smaller subqueries containing less proper nouns; thus naturally, original query getting maximum weightage.

For some queries we found less recall even after doing all of the above steps. For such queries, we expand the query by adding synonyms of query terms. To get synonyms we use WordNet database.

Table 1: Top Authoritative Domains

Authoritative Domains	*No. of Pages*
en.wikipedia.org	5996421
dictionary.reference.com	34686
dir.yahoo.com	30428
www.aboutus.org	26818
acronyms.thefreedictionary.com	25849

Results returned by subquries may not be mutually independent. If a document is retrieved by multiple queries, then it is assigned highest of these scores.

Example: For a query "President Barrack Obama", final expanded query becomes:

```
(title:"president barack obama"^3000.0 |
 title:"president barack obama"~3^1000.0 |
 title:"barack obama"^11.304348 |
 title:"barack obama"~3^5.652174 |
 title:"president obama"^13.043479 |
 title:"president obama"~3^6.5217395 |
 title:"president barack"^0.05 |
 title:"president barack"~3^0.05 |
 title:obama^4.347826 |
 title:obama^2.173913 |
 title:barack^0.5 |
 title:president^0.5)~0.1
```

Because the fully expanded query is very large we display only a part of the expanded query. Out of three query terms only 'Obama' is tagged as proper noun in our proper noun list. Hence, subqueries containing term Obama have higher weightage.

3.3 Promoting Authoritative Pages

As Web pages from authoritative domains like wikipedia.org provide reliable and less noisy information, the chances of them being relevant hits are high. Hence, we promoted the authoritative pages in the search results. To identify authoritative domains in our corpus, we generated a list of top domains from the corpus based on number of pages from that domain. We promoted authoritative pages in search results which were ranked among top 100 hits. Table 1 shows a few top authoritative domains.

4 Results

The performance measures used for evaluation were Prec@N and nDCG@N, where Prec@N is precision

Table 2: Prec@N for our runs

RunID	*Prec*@10	*Prec*@30	*Prec*@50	*Prec*@100	*MAP*
iiithAuEQ	0.208	0.188	0.178	0.189	0.154
iiithAuthPN	0.214	0.195	0.183	**0.194**	0.156
iiithExpQry	**0.285**	**0.230**	**0.207**	0.180	**0.179**

Table 3: nDCG@N for our runs

RunID	*nDCG*@10	*nDCG*@30	*nDCG*@50	*nDCG*@100
iiithAuEQ	0.260	**0.268**	**0.279**	**0.290**
iiithAuthPN	**0.207**	0.240	0.252	0.267
iiithExpQry	0.202	0.233	0.247	0.263

at rank N, and nDCD@N is Normalized Discounted Cumulative Gain(NDCG) at rank N. Table 2 shows the Prec@N values for N = 10, 30, 50 and 100 and Table 3 shows the nDCG@N values for N = 10, 30, 50 and 100.

We submitted 3 runs for the Million Query track where each of the runs were built on top of Lucene. The salient features for each of the runs are mentioned:
In iiithExpQry to enhance recall we did query expansion using WordNet, giving higher boost to subqueries containing named entity terms from the original query, to prevent query drift. It also incorporated de-duplication wherein we removed repeated hits. We did this, because we observed that a number or similar/identical hits were coming in the top results and deduplication helped in removing them. In retrospect, we realised that this severly hurt our performance as it resulted in removal of several relevant (though possibly repeated) results.

The iiithAuthPN run included features from iiithExpQry except query expansion using WordNet. It incorporated changes in terms of improved query expansion (my minimizing query drift using selective boosting). It also incorporated boosting of hits from authoritative domains.

The iiithAuEQ incorporated a combination of all approaches that we experimented with. Since, it resulted in the worst performance, we believe that its difficult to integrate multiple approaches to come up with a cumulative improved performance.

5 Summary

We started off with the understanding of how Nutch [1] works. Nutch is an open source search tool built on top of Lucene and Hadoop. We also indexed the clueweb09 data in the same way. Our architecture which combines both Lucene and Hadoop is explained in detail in Section 2 and also depicted in Figure 1 and Figure 2.

To summarize, we experimented with 1) Proper noun based query expansion, 2) Authoritative boost for hits, and 3) Deduplication to remove similar pages from the top hits. Expanding the proper noun terms helped in increasing the recall for the search results, while authoritative pages helped in promoting the top relevant results from authoritative domains. Using de-duplication resulted in the removal of relevant but redundant hits from the search results. We attribute the lower precision of all the three runs to the removal of such duplicates.

We also made an attempt at query classification task of classifying quries into precision/recall queries. The results obtained were not encouraging. We annotated the queries manually. Each query was annotated by four annotators, but the annotations differed to a greater extent. This called for some sort of personalization in query classification.

In future, we wish to work on query classification and design a more robust proper noun based query expansion. Also we want to investigate better ways of detecting authoritative pages in a data set.

References

[1] Mike Cafarella and Doug Cutting. Building nutch: Open source search. *Queue*, 2(2):54–61, 2004.

[2] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[3] George A. Miller. Wordnet: a lexical database for english. *Commun. ACM*, 38(11):39–41, 1995.