Development of a Search Engine and Applications in Information Retrieval

Anand Vishwaraj
G1501033J
Wee Kim Wee School of Communication and Information
vishwara001@e.ntu.edu.sg

Bhatt Dhiraj
G1501036L
Wee Kim Wee School of Communication and Information
dhiraj002@e.ntu.edu.sg

Ngo Thanh Tung
G1501449H
School of Computer Engineering
ngot0008@e.ntu.edu.sg

Pattamasattayasonthi Napas
G1501068K
Wee Kim Wee School of Communication and Information
napas001@e.ntu.edu.sg

Rakpong Kaewpuang
G1501453B
School of Computer Engineering
rakpong001@e.ntu.edu.sg

ABSTRACT

This work is a part of academic assignment in Information Retrieval & Analysis course'2016 at NTU. Following first part of our project, we have setup elastic search cluster and performed several indexing related experiments like lowercase indexing, removal of stopwords and stemming while indexing to notice the changes in indexing speed and index disk-size for DBLP dataset. On query side, we developed a GUI web-application where any user can do queries over the indexes of elastic search. For project 2, we indexed the DBLP dataset in order to retrieve specified information such as most relevant publication venue and year given a conference/journal as query, and showcase our information retrieval system efficacy in that. We have also reported the accuracy of our system.

Keywords

Indexing, Search Engine, Elastic Search, Information Retrieval, Classification

1. INTRODUCTION

Information retrieval is very prominent today. With the taxonomiess of websites growing, Text search is now a basic feature of any website. Search is just a concept under teh big blanket of infromation retrieval. In this assignment, we play around with information retrieval over elastic search perform some experiments over indexing, query and analyzing some of the information retrieval related tasks.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

2. SEARCH ENGINE DEVELOPMENT

This section gives the details on the development of the search engine using open-source APIs.

2.1 Indexing

Indexing process takes in documents and feeds it to elastic search. We read the json document and only indexed the documents with type article or inproceedings, ignoring the rest. Since, we are using elastic search, we need to structure the document in a json format. This json data contains all the required fields and extra information required to differentiate in between various types of data for example: A typical json data for article will contain fields such as title:extracting data from elastic search, author:vishwaraj, etc and an extra json field like type:article, so that the document can be identified as an article at the time of retrieval.

In order to save network callls and optimize the time of indexing, we have used bulk indexing, which allows us to send a set of documents to elastic searhc and get it indexed all at once, hence saving a significant number of network calls. Elastic search stores data in its indexes, each of which can have different settings. We perform settings over this index to try out various experiments such as indexing time, disk size of index, etc. For example, we added the setting of filtering documents to be indexed as lowercase, or maybe perform stemming, or maybe remove stopwords. Elastic search uses a standard analyzer which reads each documents and performs operations according to the index settings. We have added a few index settings inside elastic search which will override the standard analyzer and give us desired results of lowercase, stopwords removal, and stemming.

2.2 Indexing System Architecture

There are several steps in indexing process:

1. We parse the DBLP dataset and do the first stage filtering, where we discard entries which are not of type article or inproceedings.

2. We keep reading the dataset, until we get a certain number of files to form a batch.

3. We form a batch of json objects with the batch of doc-

uments, because we elastic search expects documents in json format and json can be easily transferred over the network in the form of a serialized string.

4. We send this batch over to elastic search for indexing. We are waiting over elastic search to acknowledge the indexing process being successfully completed.

5. Once indexing is finished, we flush the elastic search client in order yo make sure that all the remainign documents which could not form a batch or have failed indexing process are also indexed.

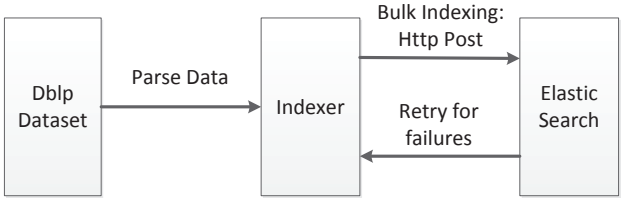

Figure 1: Indexing System Architecture.

2.3 Results for Indexing Experimentals

2.3.1 Stemming on "Title" field

Table 1: Stemming on "Title" field

	Before SnowBall Stemming Filter	After SnowBall Stemming Filter
Time (ms)	270,677	414,775
Size of Index	856.4 MB	384.1 MB

Table 1 summarizes the result of stemming on "*Title*" field. Size of index in Table 1 represents the data folder disk size. **No significant reduction in the size of index is obtained after "*Snowball*" stemming filter**. Stemming library automatically removed stopwords for us. Size reduction is comparable to only stopwords removal. In terms of impact on query, before "*Snowball*" stemming filter, the search results did not entertain mis-aligned word forms. After "*SnowBall*" stemming filter, phrase search queries became unsupported.

Here, we have applied the filter of snowball, which forces standard tokenizer of elastic search to strip any word to its root form. We have used the snowball filters to remove any language transformations of the word.

2.3.2 Lowercase filter for "Title" field

Table 3 summarizes the result of the lowercase filter on the "*Title*" field. Size of index in Table 3 represents the data folder disk size. No significant reduction in the size of index is obtained after applying the lowercase filter.

In terms of impact on query, before lowercase filter, the search results were not able to retrieve results with mismatched case. After applying the lowercase filter, the query needed to be converted to lowercase. Search queries became unsupported.

Here, we have applied the filter of lowercase, which forces standard tokenizer of elastic search to convert any capital letter to lowercase.

Table 2: Settings for stemming

```
"index": {
  "analysis": {
   "filter": {
    "snowball": {
     "type": "snowball",
    "language": "English"
    }
   },
   "analyzer": {
    "dblp_analyzer": {
    "type": "custom",
    "tokenizer": "standard",
   "filter": [ "lowercase", "snowball"]
    }}}
```

Table 3: Lowercase filter for "Title" field

	Before Lowercase Filter	After Lowercase Filter
Time (ms)	283,526	397,255
Size of Index	856.4 MB	855.4 MB

2.3.3 Stopwords filter on "Title" field

Table 5 summarizes the result of stemming on "*Title*" field. Size of index in Table 1 represents the data folder disk size. No significant reduction in the size of index is obtained after the stopwords filter. Stemming library automatically removed stopwords for us. Size reduction is comparable to only stopwords removal. In terms of impact on query, before applying the stopword filter, any stopwords in the queries led to undesired results. After stopword filter is applied, phrase search queries became unsupported.

Default stopwords list from Lucene is as follows: "*a*", "*an*", "*and*", "*are*", "*as*", "*at*", "*be*", "*but*", "*by*", "*for*", "*if*", "*in*", "*into*", "*is*", "*it*", "*no*","*not*", "*of*", "*on*", "*or*", "*such*", "*that*", "*the*", "*their*", "*then*", "*there*", "*these*", "*they*", "*this*", "*to*", "*was*", "*will*", and "*with*".

Here, we have applied the filter of stopwords, which forces standard tokenizer of elastic search to remove any word if it matches from the default stopwords list.

Table 4: Settings for lowercase

```
"index": {
  "analysis": {
   "tokenizer": {
    "dblp_tokenizer": {"type": "standard"}
   },
    "analyzer": {
   "dblp_analyzer": {
  "type": "custom",
  "tokenizer": "keyword",
   "filter": ["trim", "lowercase"]}
  }}}} #lowercase filter was added
to force lowercase filter in each token.
```

Table 5: Stopwords filter on "Title" field

	Before Stopward Filter	After Stopword Filter
Time (ms)	270,677	447,523
Size of Index	856.4 MB	386.6 MB

Table 6: Settings for stopwords removal

```
"index": {
  "analysis": {
   "filter": {
    "my_stop": }
  "type": "stop",
    "stopowords": '_english_"
```

2.4 Querying

A web-based user interface was designed to connect to the localhost server where elastic search server is hosted. The query page of the web interface allows users to enter free text queries on any/specific fields, and enter phrase queries as well. The GUI is shown in Fig. 2.

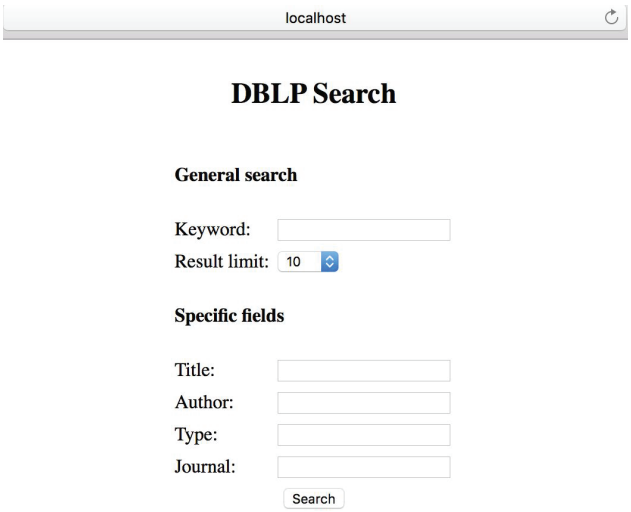

Figure 2: Graphical User Interface.

2.5 Querying System Architecture

Elastic search only understands queries in the form of json. Hence, we need to form a json query from the user input into the web UI form for viewing indexing results. Also, queries run on user side on their web portals. So, in order to make queries accessible from all clients and to make the elastic search cluster secured from public visibility, we make a query server which caters to the client search requests and responds back with the elastic search response.

There are several fields which need attention over the portal page: Json format of the queries must be sensitive to these entries on the query page entries:

1. If json format of query is made, it needs to take care of the number of results expected by the UI page.
2. If indexing has happened with lowercase filter enabled, queries must be done with lowercase filter on.
3. If indexing has stopwords removed, then we need to drop the drop words from the query.
4. If indexing has stemming activated, then we also need to stem the search query words before forming the json query.
5. There are different query formats for phrase query (query with quotes), term query (query on certain field), and general text query (match_all query).

We need to fire the json query and the response contains a lot of garbage results. We only ask the details which is required to show over elastic search in order to make elastic search faster. For example, we only need rank of query results, title, author, type of document. Also, we need to parse the queries and then arrange the documents in a table view. We do not do ranking of results because of lack of scores availability from the portals.

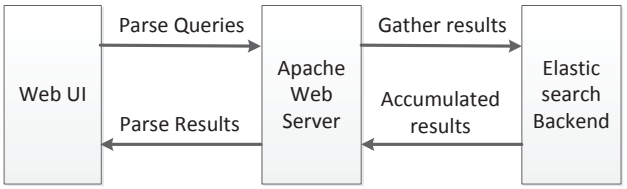

Figure 3: Query System Architecture.

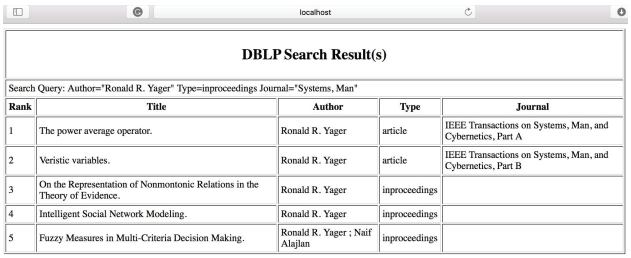

Figure 4: Phrase queries.

3. DEVELOPMENT OF IR APPLICATIONS

3.1 Methodologies

To find the most popular research topics in a given year, there is a requirement to build a model that is able to extract topics from each paper. This can be done by training a classifier that takes the title of paper as input, and classifies them into different research topics. We start with a few topics (general topics such as *Machine Learning*, *High Performance Computing*, etc.), and then increase the number of topics by dividing them into sub-topics. Naïve Bayes algorithm [1] is used to train the classifier. We consider each year as a virtual document, with attributes representing the number of papers about each specific topic published in that year. After the indexing, the topics that have highest frequency in a year will be returned as the most popular research topics in that year.

For the second requirement, the same approach can be applied. Each conference/journal is indexed as a document.

Those documents can be converted in to vectors of which elements represent frequency of each research topics. Then we can compare two conferences/journals by calculating the similarity/distance between their two correspondence vectors. We use cosine similarity as the method of computing the similarity between two vectors.

3.2 Implementation

3.2.1 Classification Model

- *Data Collection:* We obtain the training data for the classifier in two ways. First, for each research topic, we manually select some important keywords related to that area. For example, the topic "*Machine Learning*" is highly associated with the keywords "*Classification*", "*Clustering*", "*Deep Learning*", etc. Then we enrich the dataset by finding papers about that topic and add their content to the training data. To avoid special symbols and notations, we only contain the abstract of those papers and ignore the body. Since number of topics is small and the topics are very well separated, we realized that only a small amount of training data is sufficient to build a classifier which generalizes well when tested on testing data.

- *Classifier Training:* Naïve Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem. In this project, we utilize a java library provided by Weka to handle the training process. Given a title T to be predicted, the algorithm first represent it as a vector of word $X = \{x_1, x_2, \ldots, x_n\}$. For each possible class C_k, the probability that T belongs to k is compute as $P(C_k|X) = \frac{P(C_k)P(X|C_k)}{P(X)}$. The class with highest probability will be returned as the output of the classifier. In our problem, since each paper can be categorized into more than one research topics, we pick the top k classes with highest probability instead. The method of how to pick the best k will be described in the experiment.

  To increase the accuracy of the classifier, before performing prediction, we first pre-processed the input by removing stop words from the titles. Since the number of words in each title is very limited, we do not need an external library for stop words. Instead, we manually create a short array that contains some very common stop words such as "*a*", "*the*", "*to*", etc. This will help to boost the performance as well as reduce the computational time. To avoid false positive result, we will classify a title as "*unknown*" class if the probabilities of the title falling into all categories are very similar, which means $p_{\max} - \text{average}(p) < \epsilon$, where ϵ is a very small number.

- *Information Retrieval:* For the first requirement, we simply consider each year as a document and store them in the index. Then for each query, the system will retrieve the document with exact year value and return topics with highest frequency. Similarly, we store each publication venue and year in the index as a single document. Given a conference name and year, we first extract all conferences and calculate the similarity between the given input and each of documents from the index. The top 10 publication venue and year with highest similarity will be returned.

3.3 Experiment and Evaluation

3.3.1 Experimental Setup

The testing data for classification module can be obtained from DBLP dataset. For each topic, we query the index with some relevant keywords and use the result as input of the classifier. To make a fair experiment, we avoid using too obvious key words when querying. Only words which do not appear in the training data or appear very infrequently are chosen.

3.3.2 Evaluation Criteria

We define a title to be k-successful classified if the right topic of that title lies in top k topics returned by the classifier. Then the accuracy can be computed as:

$$\text{Accuracy}(k) = \text{Success Rate } (k) = \frac{\text{number of } k-\text{successful classified titles}}{\text{number of titles}} \quad (1)$$

Since there is no benchmark for the information retrieval module, we just show the result and give some explanation.

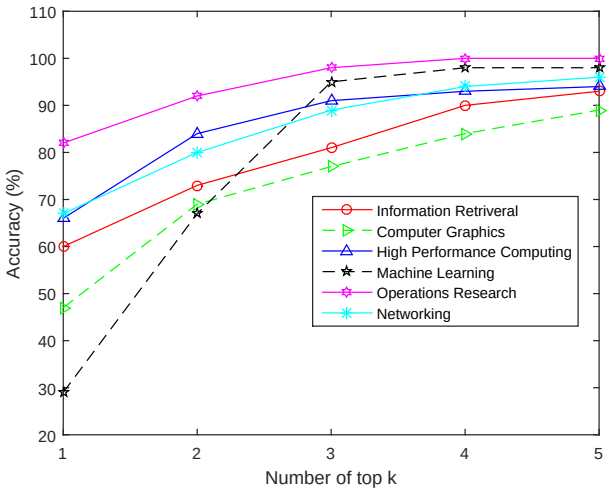

Figure 5: The success rate of the classifier.

3.3.3 Experimental Result

The Fig.5 shows the success rate of the classifier when we test on the titles from different topics. We only show the result of 6 topics since the graphs are very similar among all topics. It can be seen that when $k = 5$, the classifier can achieve more than 90% accuracy rate. However, to avoid false positive result, we only choose the top 2 topics as the output of the classification, as we can still achieve a average success rate of roughly 80%.

To see how removing the stop words actually improve the performance of classification module, we conduct experiment on a set contains 100 titles with many stop words, with and without stop words removal. The result shows that

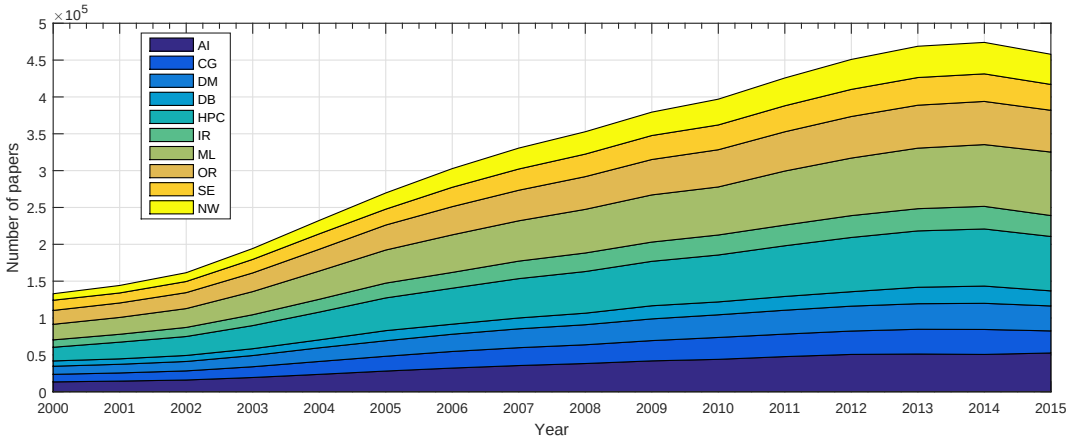

Figure 6: The distribution of research topics. Note that AI, CG, DM, DB, HPC, IR, ML, OR, SE, and NW stand for Artificial Intelligence, Computer Graphics, Data Mining, Database, High Performance Computing, Information Retrieval, Machine Learning, Operations Research, Software Engineering, and Networking, respectively.

removing stop words increases the ability of correct topics by an average percentage of 2%. Hence we can expect that this may help improve the accuracy of the classifier. Since the number of topics we are working on is small (from 10 to 20), we will return only top-3 most popular research topics in each year.

Fig.6 shows distribution of research topics from year 2000 to 2015. It can be observed that *Networking*, *Software Engineering* and *Operations Research* are always in the top 3 most popular topics. It can be explained by the unbalance among classes, some classes are more general and related too more research papers. If we increases the number of topics by dividing those general topics into more specific sub-topics, it is expected that different result would be obtained.

The Table 7. shows the result when searching for similar conferences/journal. We choose 6 conferences randomly from the XML data set and use them as inputs. From the result, we can see that the almost all the output conferences/journals are about the same research area as the query conference. Even though no information is learnt from the name of publication venue, we can still approximate the similarity between two conferences by comparing the topics of papers published in those conferences.

4. CONCLUSION AND RECOMMENDATION

In this section, we discuss the strength and weakness of our system, as well as provide some approaches that can be taken to further improve the performance of the application. Those approaches were not implemented in this project due to the scope limitation of our work and resources/time limitation.

Assignment 1 has been described in detail as in Section 2. The experiments included setting up elastic search cluster and performing indexing and query experiments. We described how changing a few parameters in indexing like indexing in lowercase only, removing stopwords and stemming words can have a significant impact on index size and indexing time. On query side, we developed a niche GUI to showcase the impact of indexing experiments on querying. We needed to change the queries according to indexing parameters, for example for lowercase indexing we need to do lowercase queries to elastic search. Similarly, we need to perform stemming on the query words and remove stopwords if the same has been done while indexing.

For assignment 2 described in Section 3, the experimental result has shown that our classifier can extract correct topics from the title with high accuracy. However, the experiment is still bias since we use relevant keywords to collect testing data for each topic. The title itself contains very little or very ambiguous information. For some papers, it is nearly impossible to find the correct topics. To address those difficult cases, we can try to retrieve more information from each document by exploiting other attributes. For example, we can send a request to the URL provided by each document and crawl more keywords from the Internet. Those keywords will be appended to the original title, which will be used as input to the classifier. This approach requires a lot of time, since the number of documents in dataset is very huge. However, it will definitely boost the performance of the classification model.

To find the most relevant publication venue and year given a conference/journal as query, we need to compute (n-1) cosine similarity, where n is the number of conferences and journals. Since the search result should be returned in a few milliseconds, the performance of our system could be damaged seriously as n increases. To improve the searching speed, we can first cluster the virtual documents into different groups. It can be observed that two conference about the same research area should be highly related to each other (e.g., ICML and NIPS). We can leverage this fact to remove all candidate answers that are not closed to the given query. Assume that there are k clusters, then each cluster has approximately ($\frac{n}{k}$) documents. When a conference/journal is given as input, we then only need to find related conferences/journals from the same cluster.

Table 7: Searching Results

Conference	Top 5 Relevance
Multimedia Information Systems 2004	1. Adaptive Multimedia Retrieval 2007 2. AND 2008 3. AAAI Fall Symposium: Multimedia Information Extraction 2008 4. MMDB 2003 5. ACM Multimedia Workshops 2000
NIPS 2012	1. Journal of Machine Learning Research 2010 2. Int. J. Machine Learning & Cybernetics 2015 3. AISTATS 2012 4. ICML 2004 5. ICML (3) 2013
Multimedia Information Retrieval 2004	1. CIVR 2006 2. Storage and Retrieval for Media Databases 2001 3. CBMI 2014 4. Storage and Retrieval Methods and Applications for Multimedia 2005 5. CLEF (2) 2009
IEEE Computer Graphics and Applications 2008	1. International Conference on Virtual Storytelling 2005 2. Eurographics (Tutorials) 2013 3. CGVR 2006 4. Smart Graphics 2003 5. TPCG 2007
J. Internet Services and Applications 2012	1. MACE 2010 2. NOMS (1) 2004 3. Integrated Network Management 2007 4. FTDCS 2004 5. IICIS 2004
The Future of Software Engineering 2010	1. Software Education and Training Sessions @ ICSE 2005 2. ICSE - Future of SE Track 2000 3. Computer Science in Sport 2006 4. SWAN@SANER 2015 5. PhiSE 2006

5. REFERENCES

[1] D. D. Lewis, "Naive (Bayes) at forty: The independence assumption in information retrieval", *Machine Learning: ECML-98: 10th European Conference on Machine Learning Chemnitz*, pp. 4-15, April 21–23, 1998.

[2] *Elastic*, https://www.elastic.co/

[3] *Kibana*, https://www.elastic.co/downloads/kibana

[4] *Apache HttpComponents Client 4.5.2*, https://hc.apache.org/downloads.cgi

[5] *Apache Tomcat 5.5*, http://tomcat.apache.org/

[6] *Eclipse Java IDE*, https://eclipse.org/downloads/

[7] *Javax Json 1.0*, https://jsonp.java.net/download.html

[8] *Sense 2.0.0*, https://www.elastic.co/guide/en/sense/master/installing.html

[9] *Weka3*, http://www.cs.waikato.ac.nz/ml/weka/