Project 3: Evaluation of IR Models
CSE535 - Information Retrieval

Yash Narendra Saraf
UB Person No. 50290453
MS Computer Science and Engineering
School of Engineering and Applied Science
ysaraf@buffalo.edu

Abstract

The project is about studying different IR models like Vector Space Model, Best Matching 25 Model and Divergence from Randomness model which all uses different techniques to find the similarity scores. The models has been discussed in brief detail and also optimization techniques are discussed for improving on all three models.

1 IR Models

For the document retrieval based on ranking from the corpus there has to be a ranking mechanism involved so that the user can access the documents according to the relevance of the documents. Now for ranking the documents one of the most common techniques used is to assign a score with respect to the query so that the documents can be ranked according to the decreasing order of scores. The above three models use the scoring of documents approach to rank documents. Now the technique used by the above three models for scoring the documents is way different then each other.

2 Project Goal

We have been given 15 train queries with their manual relevance judgment over the entire corpus of 3440 documents so that we could build the optimum retrieval system by defining the parameters, changing query parsers, using specific filters and trying out different means to optimize the system. This can be done by continuously monitoring the Mean Average Precision or MAP scores using the tool trec_eval which is opensource software for running evaluations. There are many optimization scores which can be used by the system like MAP, nDCG, R precision which are essentially different ways to judge the system performance. So we here are using the MAP values.

3 Mean Average Precision

Mean Average Precision or the MAP values are used are the evaluation criteria about how well the system is performing for the particular queries. The important part is that the system should have the manual relevance scores for those documents beforehand to make that judgement. The MAP scores are considered so important as the MAP scores have a very good discrimination and stability qualities. The MAP scores are the average of precision values obtained for the set of top k documents existing after relevant documents is retrieved. So this value is averaged over all the information needs. The MAP value has very high significance as the MAP values approximately give the area under the Precision Recall Curve. This is used as this assigns a single valued score for a particular query.

The MAP calculation is done using the formula -

$$MAP[Q] = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} Precision(R_{jk}) \tag{1}$$

$$q_j \in Q \tag{2}$$

Set of relevant documents and R_jk is set of ranked retrieval results from the top results from the top results until we get document d_k.

4 Models and their description

4.1 Model - Best Matching 25

The Okapi BM-25 model is an extension of the Binary Independence Model which is an probabilistic approach for scoring of the documents. The Okapi-BM-25 model takes care of a few things which are not given enough importance in the BIM model.

The model also factors in term frequency and the document length which are considered to be very important 2 parameters for the scoring of any document.

In the BM-25 model we find out the odds for a document being relevant given the document vectors and query vector, and simplify the equation in such a way that the only factor which is not constant for a particular query is termed as the RSV i.e. Retrieval status value. Now the RSV need to be calculated for all the documents in the corpus for the particular query.

The RSV value for the simple Binary Independence Model does not include any consideration for the term frequency and document length. The Okapi Model has an modified RSV criteria which includes the two parameters. The scaling for the two parameters are given by k1 and b which can be adjusted according the corpus type and type of application it is being used for.

$$RSV_d = \sum_{t \in Q} \log \frac{N}{df_t} \frac{(k_1 + 1) tf_d}{k_1((1 - b) + b(\frac{L_d}{L_{ave}})) + tf_d} \tag{3}$$

The BM-25 has been used as the default model of evaluation in solr after solr version 6.0 . The model definition has been given below-

```
<similarity class="solr.BM25SimilarityFactory">
      <str name="b">0.0</str>
      <str name="k1">0.3</str>
</similarity>
```

Now here we have two parameters to optimize the value of MAP value on

k1 - Controls non-linear term frequency normalization.

b - Controls to what degree document length normalizes tf values.

Now during optimization we tried out many k1 and b values and had an observation that we can't optimize k1 and then look for ideal value for b as both are independent variables and have different effect on the final MAP score. So for finding of an appropriate values I first used k1 as the reference and found out that the k1 has good values over the range of low values i.e ¡ 0.4 and for few of those values I tried to find the optimum value for the b parameter. After multiple such trial and errors I found the above values shown to be very effective.

The value of b here is 0 as we here are dealing with tweets and the document length when it comes to tweets is not very highly variable as the tweets has the maximum length of about 160 characters and the average very close to that. So there is not a very high variance among the document lengths so there is no need for normalization of document lengths.

Methods tried to optimize the MAP score

4.1.1 First Optimization - Using Language Based Search

I used the google language detection for the detection for all the queries and translated the queries to other two languages. So now my final query os the concatenation of all query in all the three languages. This is done to make sure that there is retrieval of documents from all the languages rather than only one language as the relevant information is present in all the languages. Using this gave me increase of about 0.2 in the MAP scores which is a very good response.

4.1.2 Second Optimization - Using dismax query parser

The solr standard query parser is very capable still it does'nt allow you to define fields specific weighing for the search. Usually in a dataset we have multiple fields which are not of equal importance. It might be that one field is more important than the other when it comes to queries.

So using the qf parameter and pf parameter I tried tweaking the weights for the different fields by passing the query. Now for the pf I did'nt observe any change so I dropped it and then moved onto the qf which was significant. The qf parameter introduces a list of fields, each of which is assigned a boost factor to increase or decrease that particular fields importance in the query. Now boosting particular fields like hashtags in this particular model helped. The pf I think did'nt improve the results by a lot as the pf is used for the phrase queries and the dataset is very small for the phrase queries to actually show some significant difference.

4.1.3 Third Optimization - Query Manipulation

Instead of using the given query directly I have first tailored the query for the given dataset. For example the query has been converted to all the three languages and then the query has been applied. According to my observations this was not working specifically well when the queries were in the language english so for the english language the query was left as it is.

Now since hashtags are kind of keywords which try to sum the query into single field, I have appended the field based searcch for the hashtags by passing them as the "$tweets\_hashtags$: "$hashtags$" which seemed to boost the results to a very large extent. The hastags are one of the most important parameters. The query was formulated using all the languages and the hashtags to find the ideal MAP value.

4.1.4 Fourth Optimization - Using synonymGraphFilterFactory

Synonyms are an important part of the search as the corpus might contain many terms which mean the same but are not found as the synonyms are not defined. The synonymGraphFilterFactory helps us to define all the synonyms in the seperate text file synonyms.txt which has all the relevant synonyms with respect to the corpus. The synonyms may also end up hurting the MAP scores or the quality of document relevance when the synonyms are added which are not present in the corpus or cannot be replaced in place of the original term. This might end up skewing the document retrieval far from the actual results. So the synonym file has to be defined very carefully.

Using this filter pushed the MAP scores about 0.2 to 0.3. The synonyms improved the retrieval. Also to the synonym files the translations are added to make sure the cross language retrieval is also very good.

4.1.5 Fifth Optimization - Using Analyzers and Filters

Further language specific filters has been added which enhance the judgement of that field better. For example treating the german text as text_de helps as now we apply the german normalization and the german stemming factories along with the language specific stopwords. These techniques improve the performance as the language specific processing clears all the clutter and then the information can be queried way more efficiently.

```
<analyzer type="index">
 <tokenizer class="solr.StandardTokenizerFactory"/>
```

```
 <filter class="solr.SynonymGraphFilterFactory" expand="true"
    ↪ ignoreCase="true" synonyms="synonyms.txt"/>
 <filter class="solr.LowerCaseFilterFactory"/>
 <filter class="solr.StopFilterFactory" format="snowball" words="
    ↪ lang/stopwords_de.txt" ignoreCase="true"/>
 <filter class="solr.GermanNormalizationFilterFactory"/>
 <filter class="solr.GermanLightStemFilterFactory"/>
</analyzer>
```

The above optimization has been shown for the german language.

```
<analyzer type="index">
 <tokenizer class="solr.StandardTokenizerFactory"/>
 <filter class="solr.LowerCaseFilterFactory"/>
 <filter class="solr.StopFilterFactory" format="snowball" words="
    ↪ lang/stopwords_ru.txt" ignoreCase="true"/>
 <filter class="solr.SnowballPorterFilterFactory" language="
    ↪ Russian"/>
</analyzer>
```

The above optimization has been shown for the Russian language.

4.1.6 Optimizations which failed

Also tried defining the searchhandler inside the solrconfig.xml. The searchhandler was used for the field specific search. For example it will check for the all the translations of the query in only their respective field and the hashtags only in the hashtags field using the OR operator. This was not as effective as the above techniques used. Also defined the dismax query parser in the solrconfig.xml which is an alternative for the dismax query URL. There was no difference in the results as both of them as essentially doing the same thing.

Also there weere several filters which did not perform as well I expected them to for example porter stemmer filter hampered the results rather than giving them the boost.

Following is the example of the search handler which I tried.

```
 <requestHandler name="/trial" class="solr.SearchHandler" default
    ↪ ="false">
 <lst name="invariants">
 <str name="q">
 (_query_:"{!edismax qf='text_en' v=$uq1}"
 OR _query_:"{!edismax qf='text_de' v=$uq2}"
 OR _query_:"{!edismax qf='text_ru' v=$uq3}"
 OR _query_:"{!edismax qf='tweet_hashtags' v=$uq4}"
 OR _query_:"{!edismax qf='tweet_hashtags' v=$uq4}")
 </str>
 </lst>
 </requestHandler>
```

4.2 Vector Space Model

The vector space model revolves around high dimensional geometry where all the documents and queries are considered as the vectors in the high dimensional space. This was one of the primitive models which gave very good results. This model was used as the standard model before solr 6.0. This model uses the $tf - idf$ weights as the standard weights for all the terms in the vector. This model is so flexible that it allows you to find the documents which are close to one another, document clustering and also query on the entire corpus.

The model uses a very basic idea for similarity between vectors that is the standard cosine similarity which simply finds out the angle difference between the two vectors.

The VSM is defined in the Solr in the following way.

```
<similarity class="solr.ClassicSimilarityFactory"/>
```

The scoring of the documents is done in the following manner -

$$score(q,d) = queryboost(q)\frac{V(q).V(d)}{V(q)}.doclennorm(d).docboost(d) \quad (4)$$

Normalizing V(d) to the unit vector is known to be problematic in that it removes all document length information. To avoid this problem, a different document length normalization factor is used, which normalizes to a vector equal to or larger than the unit vector: doc-len-norm(d).

At indexing, users can specify that certain documents are more important than others, by assigning a document boost: doc-boost(d).

Lucene is field based, hence each query term applies to a single field, document length normalization is by the length of the certain field, and in addition to document boost there are also document fields boosts.

At search time users can specify boosts to each query, sub-query, and each query term, query-boost(q).

A document may match a multi term query without containing all the terms of that query (this is correct for some of the queries).

Optimizations Used

The optimizations used for the VSM model are the same which were used for the BM-25 model. For the project the optimization has been done first on the BM-25 model and then the same has been used for the other models. The only changes made are in the weighting of different fields for the dismax query parser as it significantly affected the results.

The weighting of queries is going to be sensitive to the type of weights we assign to the fields. But the other parameters like the synonyms, filters used are not going to be as sensitive. So even those parameters were changed but no positive response was observed so I stuck with the original parameters by only changing the weighting of the fields in the qf parameter in the dismax definition.

4.3 Divergence from Randomness Model

The basis for this model is based on randomness and probability. The model is simply trying to map the randomness of the document from the normal distribution. For example if we have a set of words then what are the chances of those words being arranged in that particular order. So this model is also a very accurate model when compared to the other two. There are now multiple techniques to map the randomness of a variable. This has been all implemented in the solr. There are a lot of models which are a subset of this model and based on the parameters we can select any one of the model and optimize. The model definition we used is as following

The model which gave the highest map scores in as follows -

```
<similarity class="solr.DFRSimilarityFactory">
<str name="normalization">H2</str>
<str name="afterEffect">L</str>
<str name="basicModel">Be</str>
</similarity>
```

The parameters are as follows

```
basicModel: Basic model of information content:
Be: Limiting form of Bose-Einstein
G: Geometric approximation of Bose-Einstein
P: Poisson approximation of the Binomial
```

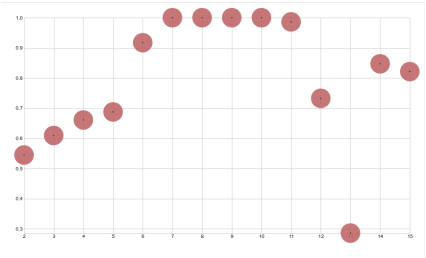
```
D: Divergence approximation of the Binomial
I(n): Inverse document frequency
I(ne): Inverse expected document frequency [mixture of Poisson and
    ↪ IDF]
I(F): Inverse term frequency [approximation of I(ne)]
afterEffect: First normalization of information gain:
L: Laplace's law of succession
B: Ratio of two Bernoulli processes
none: no first normalization
normalization: Second (length) normalization:
H1: Uniform distribution of term frequency
parameter c (float): hyper-parameter that controls the term
    ↪ frequency normalization with respect to the document length.
    ↪ The default is 1
H2: term frequency density inversely related to length
parameter c (float): hyper-parameter that controls the term
    ↪ frequency normalization with respect to the document length.
    ↪ The default is 1
H3: term frequency normalization provided by Dirichlet prior
parameter mu (float): smoothing parameter . The default is 800
Z: term frequency normalization provided by a Zipfian relation
parameter z (float): represents A/(A+1) where A measures the
    ↪ specificity of the language. The default is 0.3
none: no second normalization
```

Optimizations used

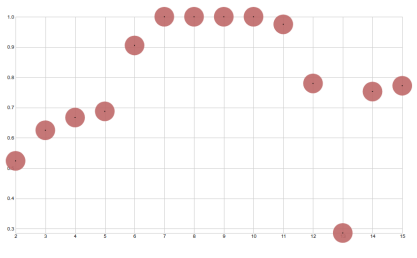
The optimizations used for this model were to use the core used for the BM25 and then change the similarity class. Now since the core was optimized for the BM25 there were significant changes to be made. So first the values for the aftereffect, normalizations and the base model were optimized. The model after being optimized then the dismax has to be optimized. Over here also we have different kinds of weights for all the fields. Rest almost all the parameters are used as they were used in the BM25 model.

5 Some results from the experimentations

The initial values for all the 3 models was around 6.6 or 6.7 and after optimization for all the three models the results for each model and the comparison of average MAP has been shown below.



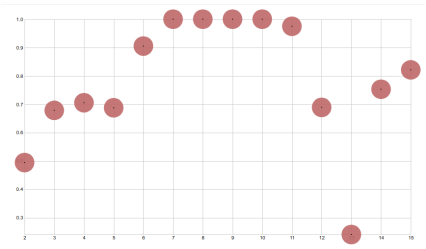
(a) BM25 Model Values



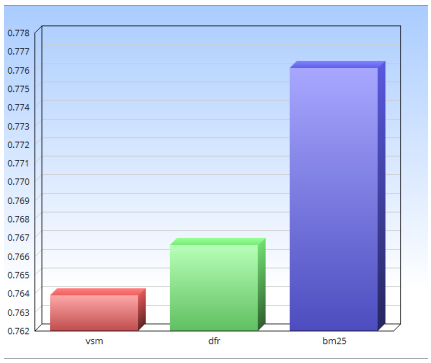
(b) DFR Model Values

Figure 1: MAP values for 15 queries

Query 1,2,3,4, and 13 has performed significantly worse than the other queries which can be clearly seen from their MAP values. The performance of these queries has been bad in all the 3 models. The performance of the 3 models is almost identical except the performance is slightly better for the BM25 model .



(a) BM25 Model Values



(b) Comparsion Between the 3 models

Figure 2: MAP values for 15 queries

Conclusion

We used different similarity class on a real dataset and optimized them based on the trec evaluation which provides measures to estimate the accuracy of the retrieval model. Also for this model we implemented several query parsing techniques and how each of them effects the relevance retrieval and compared the models.

The final results for the 15 queries are as follows -

```
map   all 0.7666 dfr
map   all 0.7639 vsm
map   all 0.7761 bm25
```

References

[1] Solr Ref Guide 6.6. *The Dismax Query Parser*. *https://lucene.apache.org/solr/guide/6_6/the-dismax-query-parser.html*

[2] Lucene Similarity Documentation, *https://lucene.apache.org/core/5_4_0/core/org/apache/lucene/search/similarities/*

[3] Solr Filter Descriptions, *https://lucene.apache.org/solr/guide/6_6/filter-descriptions.html*

[4] Introduction to Information Retrieval, *Book By Christopher D. Manning, Prabhakar Raghavan and Hinrich Schtze*. *https://nlp.stanford.edu/IR-book/*

[5] Evaluation of IR models, *Synonyms.txt and How does synonyms effect results*. *https://github.com/shreya-ravikumar/Evaluation-of-IR-Models*