Evaluation of IR Models

CSE-535 Information Retrieval

Snigdha Gupta
(UBIT: 50314905)
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
snigdhag@buffalo.edu

Abstract

The project aims at studying different IR models like Best Matching 25 model, Divergence from Randomness model and Language model which all uses different techniques to find the similarity scores. The models have been discussed in brief detail and optimization techniques have been discussed for improving mean average precision on all three models.

1 Introduction

1.1 Information Retrieval

Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).[1]

1.2 IR Models

In order to retrieve information several models have been designed. Such as– Boolean and Statistical models that include vector space and probabilistic models, Language and Knowledge -based models. The former is usual 'exact match' models while the latter are 'best match' models. When retrieving information, it is important that user can access the relevant data (documents). Hence, we use ranking mechanism to ensure that the user can access the relevant documents. For the same reason we use scoring to rank the documents.

In Solr/Lucene, scoring is very much dependent on the way documents are indexed. In Lucene, the objects we are scoring are Documents. A Document is a collection of Fields. Each Field has semantics about how it is created and stored (i.e. tokenized, untokenized, raw data, compressed, etc.) It is important to note that Lucene scoring works on Fields and then combines the results to return Documents.[3]

1.2 The Aim

A total of 15 train queries have been provided to us along with their manual relevance judgment or ground truth. The tweet corpus consists of 3440 documents in mainly three languages – English, German and Russian. The aim is to build an optimum retrieval system by defining the IR models, parameters, query parsers, using specific filters and trying out different means to optimize the system. By using the TREC-eval (Text Retrieval Conference) application we can monitor the score. This is an opensource software used to evaluate Solr search functionality. TREC-eval provides with many evaluation parameters – MAP, nDCG, k-precision, R-precision etc. For this project we will focus on optimizing the MAP score.

1.3 Mean Average Precision (MAP)

The most standard among the TREC community is Mean Average Precision (MAP), which provides a single-figure measure of quality across recall levels. Among evaluation measures, MAP has been shown to have especially good discrimination and stability. For a single information need, Average Precision is the average of the precision value obtained for the set of top k documents existing after each relevant document is retrieved, and this value is then averaged over information needs. That is, if the set of relevant documents for an information need $q_j \in Q$ is $\{d_1, \ldots d_m\}$ and R_{jk} is the set of ranked retrieval results from the top result until you get to document d_k, then

$$MAP(Q) = \frac{1}{|Q|}\sum_{j=1}^{|Q|}\frac{1}{m}\sum_{k=1}^{m_j} Precison(R_{jk})$$

The MAP value is then used to determine how well the system is performing over the queries. The system should have the manual relevance scores for those documents beforehand to make such a judgement.

2 Models

2.1 Best Match 25 (BM25)

Okapi BM25 is a ranking function used by search engines to estimate the relevance of documents to a given search query. It is based on the probabilistic retrieval framework developed in the 1970s and 1980s by Stephen E. Robertson, Karen Spärck Jones, and others.[4]

BM25 is a bag-of-words retrieval function that ranks a set of documents based on the query terms appearing in each document, regardless of their proximity within the document. It is a family of scoring functions with slightly different components and parameters. One of the most prominent instantiations of the function is as follows:

Given a query Q, containing keywords $q_1, \ldots, q_n$, the BM25 score of a document D is:

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)},$$

Figure 1BM25 score

Where $f(q_i,D)$ is q_i's term frequency in the document D, |D| is the length of the document D in words , and avgdl is the average document length in the text collection from which document are drawn. k_1 and b are free parameters, usually chosen, in absence of an advanced optimization, as $k_1 \in [1.2, 2.0]$ and b=0.75 (b can vary between 0.3 to 0.9). $\text{IDF}(q_i)$ is the inverse document frequency weight of the query term q_i. It is usually computed as:

$$\text{IDF}(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5},$$

Figure 2 Inverse Document Frequency

Where N is the total number of documents in the collection, and $n(q_i)$ is the number of documents containing q_i.

2.2 Divergence from randomness model

This is probabilistic model used to test the amount of information carried in the documents. It is based on the idea: the more the divergence of the within-document term-frequency from its frequency within the collection, the more the information carried by the word t in document d. In other words, the term-weight is inversely related to the probability of term-frequency within the document d obtained by a model M of randomness.[5]

$$\mathrm{weight}(t|d) = k\mathrm{Prob}_M(t \in d|\mathrm{Collection}) \quad \text{(Formula 1)}$$

Figure 3

M represents the type of model of randomness which employs to calculate the probability.
d is the total number of words in the documents.
t is the number of a specific word in d.
k is defined by M.

2.3 Language Model

A statistical language model is a probability distribution over sequences of words. Given such a sequence, say of length m, it assigns a probability P $(w_1,\ldots,w_m)$ to the whole sequence.[6]

The language model provides context to distinguish between words and phrases that sound similar. For example, in American English, the phrases "recognize speech" and "wreck a nice beach" sound similar, but mean different things.

We make the assumption that the probability of a word only depends on the previous n words. This is an n-gram model. The unigram model (n=1) is bag of words model.

The language model is widely used in Natural Language Processing applications. Other applications include speech recognition, machine translation etc.

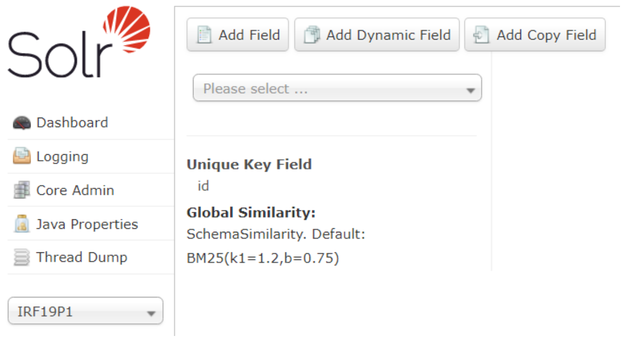
3 Implementation Techniques

In order to optimize the MAP score we use the Similarity class provided by Lucene. It is used to score a document in searching.

Each collection has one "global" Similarity, and by default Solr uses an implicit `SchemaSimilarityFactory` which allows individual field types to be configured with a "per-type" specific Similarity and implicitly uses `BM25Similarity` for any field type which does not have an explicit Similarity.

This default behavior can be overridden by declaring a top level `<similarity/>` element in the schema.xml, outside of any single field type. This similarity declaration may or may not have optional initialization parameters.

In Solr the default model is BM25 with k1 = 1.2 and b = 0.75.



Unique Key Field
id
Global Similarity:
SchemaSimilarity. Default:
BM25(k1=1.2,b=0.75)

Figure 4 Solr Default Model Configuration

3.1 Optimization I – Using Brackets

In order to retrieve more relevant documents, using brackets in the query ensures that Solr searches for each word present in the query. This gives us higher recall as well as higher precision. To obtain this result, we simply use brackets for each query. We also eliminate any unwanted characters such as newlines, colon etc. This allows us to perform query with more combinations.

```
q_en = "(" + query + ")"
q_en = quote(q_en)
q_de = "(" + query + ")"
q_de = quote(q_de)
q_ru = "(" + query + ")"
q_ru = quote(q_ru)
```

```
"q":"text_txt_en:(Russia's intervention in Syria) OR
text_en:(Russia's intervention in Syria) OR
text_txt_de:(Russia's intervention in Syria) OR
text_de:(Russia's intervention in Syria) OR
text_txt_ru:(Russia's intervention in Syria) OR
text_ru:(Russia's intervention in Syria)"
```

3.2 Optimization II – Exact Match

Another optimization technique was to search for both text_en, text_ru, text_de (of field type text_general) and text_txt_en, text_txt_de and text_txt_ru (copied fields).
This allowed us to look for any exact matches. Additionally, to implement this we can also remove LowerCaseFilterFactory from text_general.

Adding copied fields

```
<copyField source="text_en" dest="text_txt_en"/>
<copyField source="text_ru" dest="text_txt_ru"/>
<copyField source="text_de" dest="text_txt_de"/>
```

Query with text_xx and text_txt_yy

```
http://52.15.182.115:8983/solr/core1_bm/select?q=text_txt_en:%28Russia%27s%20interventi
on%20in%20Syria%29%20OR%20text_en:%28Russia%27s%20intervention%20in%20Syria
%29%20OR%20text_txt_de:%28Russia%27s%20intervention%20in%20Syria%29%20OR%2
0text_de:%28Russia%27s%20intervention%20in%20Syria%29%20OR%20text_txt_ru:%28R
ussia%27s%20intervention%20in%20Syria%29%20OR%20text_ru:%28Russia%27s%20inter
vention%20in%20Syria%29&fl=id%2Cscore&wt=json&indent=true&rows=20
```

```
"q":"text_txt_en:(Russia's intervention in Syria) OR
text_en:(Russia's intervention in Syria) OR
text_txt_de:(Russia's intervention in Syria) OR
text_de:(Russia's intervention in Syria) OR
text_txt_ru:(Russia's intervention in Syria) OR
text_ru:(Russia's intervention in Syria)"
```

Removing LowerCaseFilterFactory from text_general

```
<fieldType name="text_general" class="solr.TextField"
positionIncrementGap="100" multiValued="true">
    <analyzer type="index">
      <tokenizer class="solr.StandardTokenizerFactory"/>
      <filter class="solr.StopFilterFactory"
words="stopwords.txt" ignoreCase="true"/>
    </analyzer>
    <analyzer type="query">
      <tokenizer class="solr.StandardTokenizerFactory"/>
      <filter class="solr.StopFilterFactory"
words="stopwords.txt" ignoreCase="true"/>
      <filter class="solr.SynonymGraphFilterFactory"
expand="true" ignoreCase="true" synonyms="synonyms.txt"/>
    </analyzer>
  </fieldType>
```

3.3 Optimization III – Including hashtags

Including hashtags also helped in optimizing the MAP score. This was achieved by extracting hashtags from tweet_hashtags field and querying the same to solr.

Extracting hashtags from each query

```
hashtags = re.findall(r"#(\w+)", query)
```

Querying Solr with tweet_hashtags: <hashtag_value>

```
"q":"text_txt_en:(#Syria #SALMA #LATAKIA) OR text_en:(#Syria
#SALMA #LATAKIA) OR text_txt_de:(#Syria #SALMA #LATAKIA) OR
text_de:(#Syria #SALMA #LATAKIA) OR text_txt_ru:(#Syria #SALMA
#LATAKIA) OR text_ru:(#Syria #SALMA #LATAKIA)tweet_hashtags:
Syria tweet_hashtags: SALMA tweet_hashtags: LATAKIA"
```

4 Implementation Techniques (Model Specific)

4.1 BM25 Model

We implement the BM25 model by adding the BM25SimilarityFactory class as shown below:

```
<similarity class="solr.BM25SimilarityFactory">
    <str name="b">0.75</str>
    <str name="k1">1.2</str>
</similarity>
```

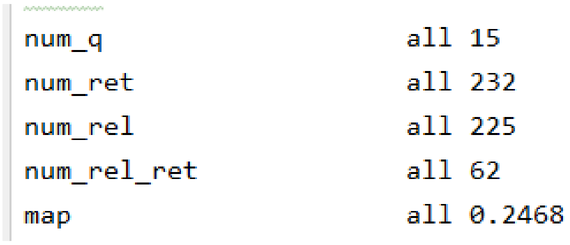
k1 (float)	: Controls non-linear term frequency normalization (saturation).
b (float)	: Controls to what degree document length normalizes tf values.

The value of b ranges from 0.3 to 0.9 and value of k1 ranges from 1.2 to 2.0

- We run trec-eval using simple query and observe that the MAP score is very low.

```
http://52.15.182.115:8983/solr/core1_bm/select?q=text_txt_en:Russia%27s%20intervention%20in%20Syria%20OR%20text_txt_de:Russia%27s%20intervention%20in%20Syria%20OR%20text_txt_ru:Russia%27s%20intervention%20in%20Syria&fl=id%2Cscore&wt=json&indent=true&rows=20

"q":"text_txt_en:Russia's intervention in Syria OR
text_txt_de:Russia's intervention in Syria OR
text_txt_ru:Russia's intervention in Syria"
```

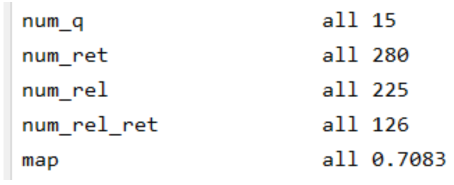


```
num_q                    all 15
num_ret                  all 232
num_rel                  all 225
num_rel_ret              all 62
map                      all 0.2468
```

Figure 5MAP score without optimization

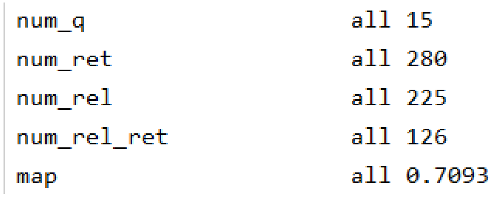
Optimization: Changing k1 and b values.

When k1=1.4 and b=0.3



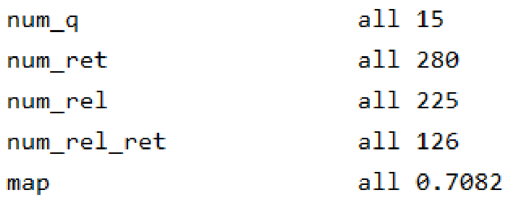
```
num_q                    all 15
num_ret                  all 280
num_rel                  all 225
num_rel_ret              all 126
map                      all 0.7083
```

When k1=1.5 and b=0.5



```
num_q                    all 15
num_ret                  all 280
num_rel                  all 225
num_rel_ret              all 126
map                      all 0.7093
```

When k1=1.0 and b=0.2



```
num_q                    all 15
num_ret                  all 280
num_rel                  all 225
num_rel_ret              all 126
map                      all 0.7082
```

When k1=1.2 and b=0.6

```
num_q                    all 15
num_ret                  all 280
num_rel                  all 225
num_rel_ret              all 126
map                      all 0.7103
```

We observe that on different values of k1 and b the MAP score changes. Since MAP=0.7103 is our best bet right now, we will now observe how score changes when k1 is fixed at 1.2

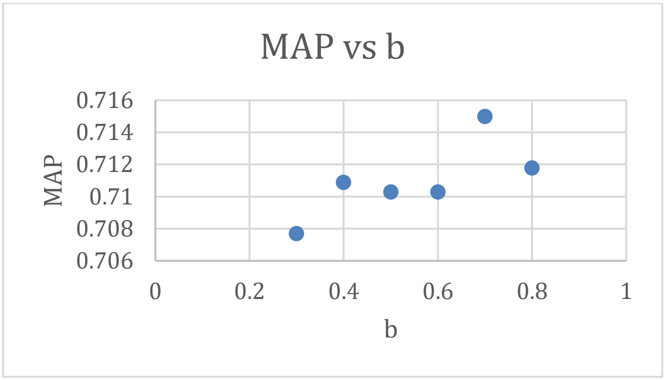


Figure 6 MAP vs b at k1 = 1.2

The MAP score increases as b increases. However, at b=0.8 the MAP score decreases.

Now, let fix b=0.7 and see how MAP varies for different values of k1

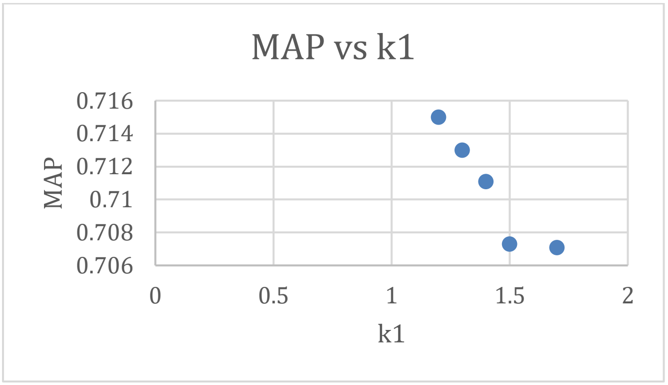


Figure 7MAP vs k1 at b=0.7

We see that with increasing value of k1 the MAP score decreases.

k1	b	MAP
1.2	0.3	0.7077
1.2	0.4	0.7109
1.2	0.5	0.7103
1.2	0.6	0.7103
1.2	0.7	0.715
1.2	0.8	0.7118
1.2	0.7	0.715

1.3	0.7	0.713
1.4	0.7	0.7111
1.5	0.7	0.7073
1.7	0.7	0.7071

Figure 8 Trial and Error for b and k1

Final Parameter Values: k1=1.2, b=0.7

```
num_q                   all 15
num_ret                 all 280
num_rel                 all 225
num_rel_ret             all 127
map                     all 0.7150
```

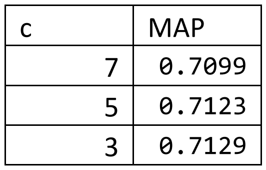
4.2 DFR Model

We implement the DFR model by adding the DFRSimilarity class as shown below:

```
<similarity class="solr.DFRSimilarityFactory">
    <str name="normalization">H2</str>
    <str name="afterEffect">B</str>
    <str name="basicModel">G</str>
      <float name="c">7</float>
</similarity>
```

1. basicModel: Basic model of information content:
    - G: Geometric approximation of Bose-Einstein
    - 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)]
2. afterEffect: First normalization of information gain:
    - L: Laplace's law of succession
    - B: Ratio of two Bernoulli processes
3. 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 μ. 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

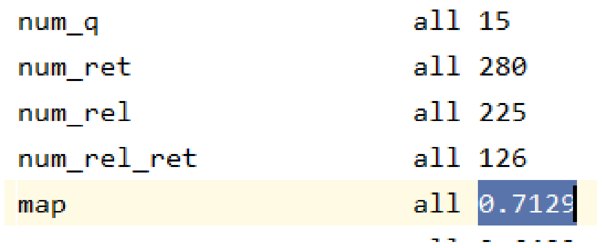
- none: no second normalization



c	MAP
7	0.7099
5	0.7123
3	0.7129

Figure 9 Trial and Error for c

Final Parameter Values: c=3



```
num_q                   all 15
num_ret                 all 280
num_rel                 all 225
num_rel_ret             all 126
map                     all 0.7129
```

4.3 LM Model

We implement the LM model by adding LMDirichletSimilarityFactory class as shown below:

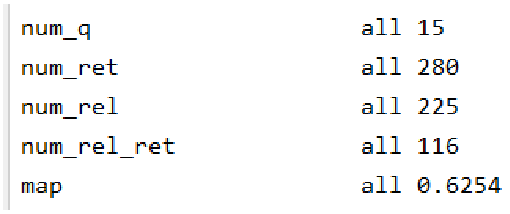
```
<similarity class="solr.LMDirichletSimilarityFactory">
   <float name="mu">2000</float>
</similarity>
```

mu (float)   : smoothing parameter μ. The default is 2000

The mu value is related to the average number of words present in the collection. Since larger documents have high word count, the average number of words is also a lot. Hence the default is set to 2000.

However, since we are dealing with tweets (240 characters), we can keep the mu value low.

At mu=2000



```
num_q                   all 15
num_ret                 all 280
num_rel                 all 225
num_rel_ret             all 116
map                     all 0.6254
```

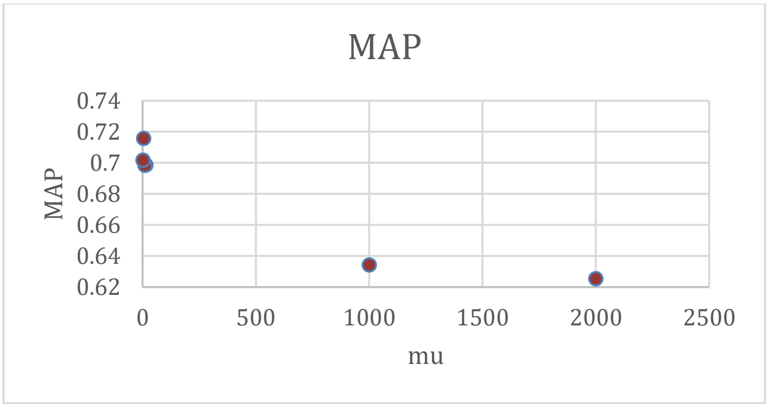


Figure 10MAP vs mu

mu	MAP
2000	0.6254
1000	0.6343
15	0.6987
10	0.6984
5	0.7028
2	0.7018

Figure 11 Trial and Error for mu values

Final Parameter Values: mu=5

```
num_q                   all 15
num_ret                 all 280
num_rel                 all 225
num_rel_ret             all 127
map                     all 0.7028
```

4 Result

The DFR model showed the best fit to us with MAP = 0.7129

References

[1] An Introduction to Information Retrieval by Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze

[2] https://aspoerri.comminfo.rutgers.edu/InfoCrystal/Ch_2.html

[3] https://lucene.apache.org/core/2_9_4/scoring.html

[4] https://en.wikipedia.org/wiki/Okapi_BM25

[5] https://en.wikipedia.org/wiki/Divergence-from-randomness_model

[6] https://en.wikipedia.org/wiki/Language_model