

Project 3:

Evaluation of IR Models

CSE 535: Information Retrieval (Fall 2016)

Instructor: Rohini K. Srihari

Teaching Assistants: Nikhil Londhe, Lu Meng, Le Fang, Jialiang Jiang

Team Number: 65

Members:

Sugosh Nagavara Ravindra (50207357)

Vaibhav Sinha (50208769)

Enhancement

Query Expansion

Query Expansion is required so that we can match related documents which do not have exact match of terms with the query. To do so, we add synonyms to the query so that it matches similar documents as well.

It is of significant importance when we have cross lingual data as standard terms in one language would not match the documents in the other language.

E.g.

Сирии ,Syrian , syrien
Flüchtlinge, Flüchtlinge, refugee, refugees, беженцев, Беженцы
russischen, russian, русский

So, whenever there is query in Russian which mentions Syria (Сирии), it will match documents in English and Russian as well.

Before Query Expansion :

map	001	0.3722
map	002	0.5274
map	003	0.5610
map	004	0.5484
map	005	0.6875
map	006	0.5090
map	007	1.0000
map	008	1.0000
map	009	1.0000
map	010	1.0000
map	011	0.9861
map	012	0.7145
map	013	0.1066
map	014	0.7745
map	015	0.8667
map	016	0.8690
map	017	0.2857
map	018	0.6261
map	019	1.0000
map	020	0.4858
map	all	0.6960

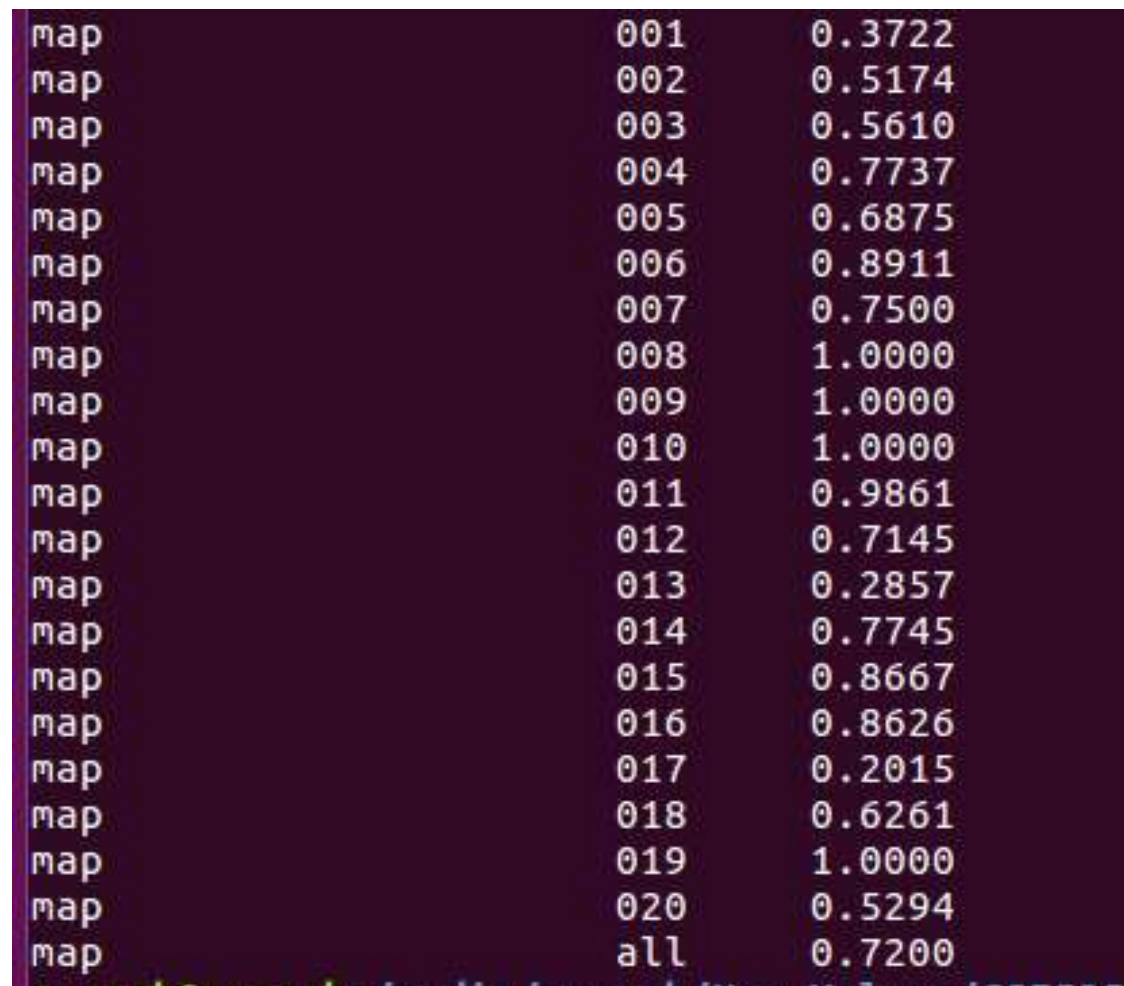
The above process can be automated by translating and adding as synonym, specific query terms using a translation API provided by Google or Microsoft.

The specific terms to be translated and added as synonym can be decided by their IDF values and the part of speech they belong to.

e.g. We can add synonyms for rare query terms who have IDF values greater than a threshold. We can also add synonyms for terms which are nouns, especially proper nouns.

In order to add synonyms for Nouns, we need to do parts of speech tagging to identify nouns.

After Query Expansion:



A screenshot of a terminal window with a dark background and light-colored text. It displays a list of terms and their corresponding IDF values. The terms are listed in the first column, followed by an index number in the second column, and the IDF value in the third column. The terms are mostly 'map', with the last one being 'all'. The IDF values range from 0.2015 to 1.0000.

map	001	0.3722
map	002	0.5174
map	003	0.5610
map	004	0.7737
map	005	0.6875
map	006	0.8911
map	007	0.7500
map	008	1.0000
map	009	1.0000
map	010	1.0000
map	011	0.9861
map	012	0.7145
map	013	0.2857
map	014	0.7745
map	015	0.8667
map	016	0.8626
map	017	0.2015
map	018	0.6261
map	019	1.0000
map	020	0.5294
map	all	0.7200

Tuning Models – BM25

There were two parameters provided by Solr for BM 25 model.

k1 - Controls non-linear term frequency normalization (saturation).

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

We tried for various values of k1 and b. There were significant changes in the overall MAP value as these parameters were varied.

The default values being 1.2 and 0.76 had results with a MAP value of **0.7110**.

With Default Values:

map	001	0.3673
map	002	0.5303
map	003	0.5610
map	004	0.7391
map	005	0.6875
map	006	0.8532
map	007	0.7500
map	008	1.0000
map	009	1.0000
map	010	1.0000
map	011	1.0000
map	012	0.6721
map	013	0.2857
map	014	0.7745
map	015	0.8667
map	016	0.8593
map	017	0.2253
map	018	0.6261
map	019	0.8922
map	020	0.5294
map	all	0.7110

On trying different values, we found that the optimum value of these parameters turned out to be.

k1 = 0.4

b = 0

For which MAP value came out to be **0.7200**

```
<similarity class="solr.BM25SimilarityFactory">  
  <float name="k1">0.4</float>  
  <float name="b">0</float>  
</similarity>
```

With Tuned Values:

map	001	0.3722
map	002	0.5174
map	003	0.5610
map	004	0.7737
map	005	0.6875
map	006	0.8911
map	007	0.7500
map	008	1.0000
map	009	1.0000
map	010	1.0000
map	011	0.9861
map	012	0.7145
map	013	0.2857
map	014	0.7745
map	015	0.8667
map	016	0.8626
map	017	0.2015
map	018	0.6261
map	019	1.0000
map	020	0.5294
map	all	0.7200

Tuning Models – DFR

DFR model Similarity offers the following parameters:

- 1 basicModel: Basic model of information content:
- 2 afterEffect: First normalization of information gain:
- 3 normalization: Second (length) normalization

Since, for this project, we were limited to using

basicModel -> G

aftereffect -> B

normalization -> H2

For the above, we got a map value of **0.6935**

map	001	0.3673
map	002	0.3504
map	003	0.5471
map	004	0.6718
map	005	0.6875
map	006	0.7596
map	007	0.8333
map	008	1.0000
map	009	1.0000
map	010	1.0000
map	011	0.9861
map	012	0.6897
map	013	0.2857
map	014	0.7745
map	015	0.7721
map	016	0.8722
map	017	0.2253
map	018	0.6261
map	019	0.8922
map	020	0.5294
map	all	0.6935

After tuning parameters we found the following optimum values:

basicModel – I(F)

AfterEffect - L

Normalization – H2

C – 10,000

```
<similarity class="solr.DFRSimilarityFactory">  
  <str name="basicModel">I(F)</str>  
  <str name="afterEffect">L</str>  
  <str name="normalization">H2</str>  
  <float name="c">10000</float >  
</similarity>
```

For the above, we were able to achieve **0.7148**

map	001	0.3722
map	002	0.5097
map	003	0.5842
map	004	0.7783
map	005	0.6875
map	006	0.9149
map	007	0.7000
map	008	1.0000
map	009	1.0000
map	010	1.0000
map	011	1.0000
map	012	0.6815
map	013	0.2857
map	014	0.7838
map	015	0.8667
map	016	0.7500
map	017	0.2253
map	018	0.6261
map	019	1.0000
map	020	0.5294
map	all	0.7148

Tuning Models – VSM

For VSM Mode, there was not much parameters to tweak. There is only one parameter “Discount Overlaps” which is a Boolean.

Changing its value to true or false did not have any impact on the MAP value.

MAP Value achieved: **0.6972**

```
<similarity class="solr.ClassicSimilarityFactory">  
  <bool name="discountOverlaps">true</bool>  
</similarity>
```

map	001	0.3533
map	002	0.4663
map	003	0.5610
map	004	0.6957
map	005	0.6875
map	006	0.8767
map	007	0.8333
map	008	1.0000
map	009	1.0000
map	010	1.0000
map	011	1.0000
map	012	0.4615
map	013	0.2857
map	014	0.7430
map	015	0.7721
map	016	0.8269
map	017	0.2253
map	018	0.6261
map	019	1.0000
map	020	0.5294
map	all	0.6972

Query Boosting

We used the lucene parser, the “dismax” query parser instead of default query parser to boost specific fields in the query(qf). The “qf” parameter introduces a list of fields, each of which is assigned a boost factor to increase or decrease that particular field's importance in the query. qf = tweet_hashtags^5 text_all^4. Text_all copy field was created which contained indexed values from text_en, text_de, text_ru. The above was introduced into the solrconfig file so that all the queries will be effected by it.

We also boosted specific terms in the queries which held more importance and meaning to the query. On doing so, the queries with those terms will be scored higher and the overall MAP value increased.

We also tried using the “pf” parameter to boost terms based on proximity. This did not affect the MAP values as the provided queries were of short length and any change would be observed if the query is of a greater length.

Before Query Boosting for BM25 Model

map	001	0.3722
map	002	0.4431
map	003	0.5610
map	004	0.7737
map	005	0.5000
map	006	0.6164
map	007	0.7500
map	008	1.0000
map	009	1.0000
map	010	1.0000
map	011	0.9861
map	012	0.7145
map	013	0.2857
map	014	0.5942
map	015	0.8667
map	016	0.8626
map	017	0.2015
map	018	0.6261
map	019	1.0000
map	020	0.5235
map	all	0.6839

After Query Boosting for B 25 Model
Map value obtained is **0.7200**

map	001	0.3722
map	002	0.5174
map	003	0.5610
map	004	0.7737
map	005	0.6875
map	006	0.8911
map	007	0.7500
map	008	1.0000
map	009	1.0000
map	010	1.0000
map	011	0.9861
map	012	0.7145
map	013	0.2857
map	014	0.7745
map	015	0.8667
map	016	0.8626
map	017	0.2015
map	018	0.6261
map	019	1.0000
map	020	0.5294
map	all	0.7200

Other Enhancements...

1. Stemmers: We tried alternating between Porter and K Stemmer and found that K Stemmer was less aggressive and lead to a slightly better MAP value
2. Exact Case Matches: Since exact case matches can be treated as more relevant, we created separate copy fields for all fields without a lower case filter.
3. PatternReplaceFilterFactory : Used it to change U.S.A to USA, etc. so that query terms for uniform query results.

Future Enhancements

1. Part of Speech(POS) Tagging and Translate

To do POS tagging, we can use Stanford Log-linear Part-Of-Speech Tagger.

To implement it in Solr, we have to make our own Solr filter which identifies Nouns in the query and then translates those terms in other language equivalents using the Google Translate API.

For each query term, we also fetch the IDF values, if that term is a rare term then it will have a high IDF value. If that IDF is higher than a threshold, we do translation of that term in other languages as well.

2. Clustering

Clustering can be done to group similar words together and thereby serve various purposes, such as query expansion, document grouping, document indexing, and visualization of search results