INFORMATION RETRIEVAL – CSE535 – FALL 2016

Project 3 – Report

Team 1:

Ajay Gandhi - 50207403

agandhi3@buffalo.edu

Siddharth Pateriya – 50206348

spateriy@buffalo.edu

Introduction:

The main objective for this project is to improve search relevancy of a Solr instance through the use of various techniques. For these purposes, we have implemented various IR models such as BM25, Divergence From Randomness (DFR) and Vector Space Model(VSM) and used the training queries provided to judge the relevance/precision of our results with the help of the TREC tool.

The measures which we have used to judge the relevancy of our system are Mean Average Precision(MAP), Normalized Discounted Cumulative Gain (nDCG) and F measure. These scores gave us an idea of how relevant were the results given by our Solr instance, and improve it using various techniques such as

1. **Tuning the parameters of the IR models**
2. **Trying out different query parsers and query expansion techniques**
3. **Use boosting for certain fields, using copyFields to create fields storing phrases instead of tokens**
4. **Using various filters, tokenizers during indexing an querying of data**
5. **Attempting translation of queries**
6. **Using synonyms to improve relevance**

We take an analytical look at all of these and decide on the characteristics of our best effort system, for each model.

IR Models Used:

We have used the following IR models

BM25

The BM25(or Okapi BM25) model has given us the best results for the training queries which we used, both when fetching 20 results or when fetching 1000 results.The default values for parameters k1 and b is 1.2 and 0.75 respectively. In our final system, we have tweaked these values to find a combination which best serves our interests

Usually the default values should suffice for this model, however, since we are dealing with a corpus of tweets, the length of the documents is fairly even throughout (maximum length of a tweet is 140 characters). Since the parameter b controls to what degree document length normalizes tf values, we got the best values for MAP and nDCG when keeping b to 0.0, since it is a factor for normalization based on document length. Since for this corpus,

document should more or less be equal throughout, without much variance, the results reflect our reasoning for choosing this as 0.0.

Also, it is important to note that BM25 is the default similarity model in Solr (since version 6.0). However to tune the parameters, we need to explicitly mention the similarity class and parameters in the Schema file. Here is how we implemented it:

Default

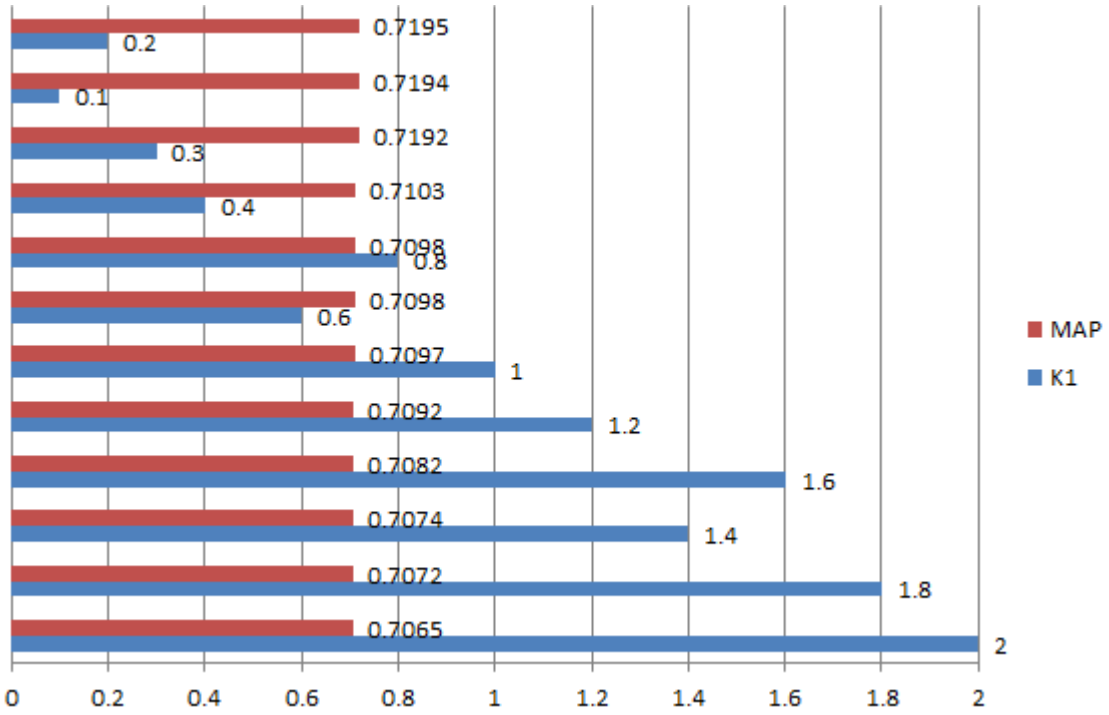
```
<similarity class="org.apache.solr.search.similarities.BM25SimilarityFactory">

</similarity>
```

Optimum:

```
<similarity class="org.apache.solr.search.similarities.BM25SimilarityFactory">
  <float name="k1">0.3</float>
    <float name="b">0.0</float>
</similarity>
```

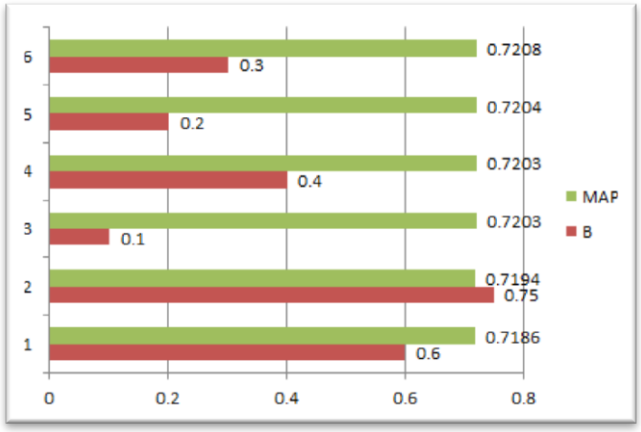
Tuning of BM25:

We tried various different combinations of k1 and b and decided on the optimum value for our system. Here are the values of MAP, keeping b as the default value(0.75).

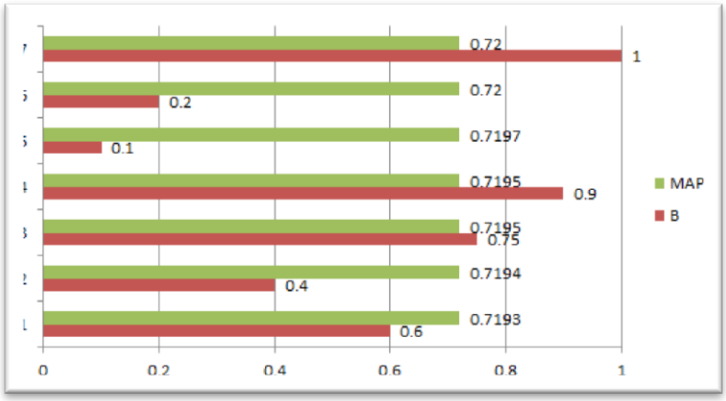


We find that 0.1,0.2,0.3 as K1 give the highest MAP value. Hence we test further taking these 3 values and modifying b this time.

Keeping K1 = 0.1



Keeping K1 = 0.2



Keeping K1 = 0.3

Hence we find that K1 = 0.3 gives us an optimum value of MAP. We further tuned the value of B to 0.0,basically eliminating any normalization based on the length of the document,since the length of all tweets is more or less spread evenly.

As we can observe from the table below, the optimum MAP and nDCG value was obtained keeping k1 = 0.3 and b = 0.0 (The table shows the TREC results for querying of 1000 rows, but similar observations were there for retrieval of 20 rows as well).

K1	B	ndcg	MAP
0.3	0	0.9154	0.7353
0.1	0.3	0.9136	0.7208
0.3	0.2	0.9135	0.7206
0.3	0.1	0.9135	0.7205
0.3	0.3	0.912	0.7205
0.1	0.2	0.9135	0.7204
0.1	0.1	0.9115	0.7203
0.1	0.4	0.913	0.7203
0.2	0.2	0.9129	0.72
0.2	1	0.9129	0.72
0.3	0.4	0.9127	0.7198
0.2	0.1	0.9116	0.7197
0.2	0.75	0.9127	0.7195
0.2	0.9	0.9126	0.7195
0.1	0.75	0.9129	0.7194
0.2	0.4	0.9128	0.7194
0.2	0.6	0.9129	0.7193
0.3	0.75	0.9122	0.7192
0.1	0.6	0.911	0.7186
0.5	0.6	0.9104	0.7113
0.4	0.75	0.91	0.7103
0.6	0.75	0.9098	0.7098
0.8	0.75	0.9098	0.7098
1	0.75	0.9086	0.7097
1.2	0.75	0.9078	0.7092
1.6	0.75	0.9061	0.7082
1.4	0.75	0.9058	0.7074
1.8	0.75	0.9051	0.7072
2	0.75	0.9044	0.7065

Divergence From Randomness (DFR):

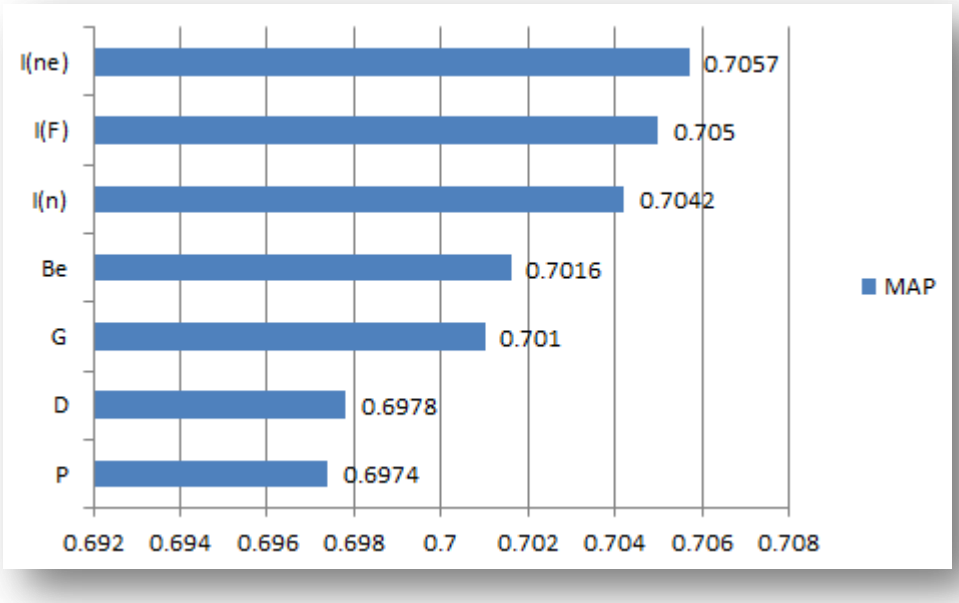
The DFR model has various parameters which can give different results:

1. Basic model of information content:
2. AfterEffect: First normalization of information gain:
3. Normalization: Second (length) normalization:

We tried various combinations of the Basic Model, Aftereffect, and 2nd Normalization to decide on the optimum configuration when using DFR. The default DFR model had the following settings:

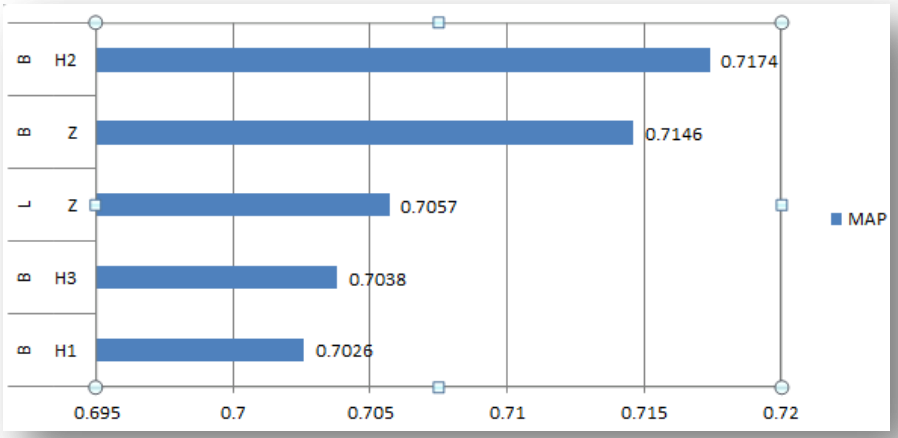
```
<similarity class="solr.DFRSimilarityFactory">
  <str name="basicModel">G</str>
  <str name="afterEffect">B</str>
  <str name="normalization">H2</str>
</similarity>
```

We played around with the Basic Models for DFR first, before tuning the aftereffect and normalization parameters. Here is an analysis of MAP values based on DFR, keeping AfterEffect as B and Normalization as H2, and changing the basicModel in each case:



We see that I (ne), I(n) and I(F) give the optimal value of MAP for the basic models. Now we further analyse these models changing the aftereffect and normalization parameters. We found that I(ne) model gives the optimal value for relevance with B and H2, across all the three models picked above.

Here are some values for different variations of the I(ne) model:



Thus we find an optimum value keeping Aftereffect as B(Bernoulli) and 2nd Normalization as H2.Hence we have chosen the above parameters for our DFR Model.

VSM Model:

In VSM or Vector Space Model, documents and queries are represented as weighted vectors in a multi-dimensional space, where each distinct index term is a dimension, and weights are Tf-idf values.

It was also previously the default similarity class in Solr, before it was changed to BM25 starting with version 6.0. The VSM similarity class is implemented through the ClassicSimilarityFactory in Solr.

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

Though initially VSM did not give a very good relevance score, with MAP hovering around 0.61 – 0.62 for the default model (without any stopword removal, stemming, synonyms etc),after using various techniques we were able to improve it to 0.6975. There are no parameters to be configured for VSM, hence there is no tuning required. However the impact of changing the tokenizers, filters and usage of synonyms may help increase the relevance and thus the overall precision of the search results.

Choosing the base model:

Since BM25 gave us a better value compared to other IR models, taking into account parameters such as MAP, nDCG and F0.5, we have chosen it to be the testbed for our the further improvements which we test using the training queries on our Solr instance.

Improvements to the system:

1. **Creation of a copyfield text_all** to store all text_en, text_ru and text_de data as phrases (not as tokens).
   Since relevance also depends on how closely the search results match the given query, we thought relevance would increase if we create a field containing only sentences of text from all the various text fields, and boosting the score for that field, if a match is found, using the qf and pf parameters in the query syntax.
   We created the copy field by adding the following lines in the schema file and defining it

```
 <field name="text_all" type="text_all" indexed="true" stored="true" multiValued="true"/>
 <copyField source="text_de" dest="text_all"/>
<copyField source="text_en" dest="text_all"/>
<copyField source="text_ru" dest="text_all"/>
```

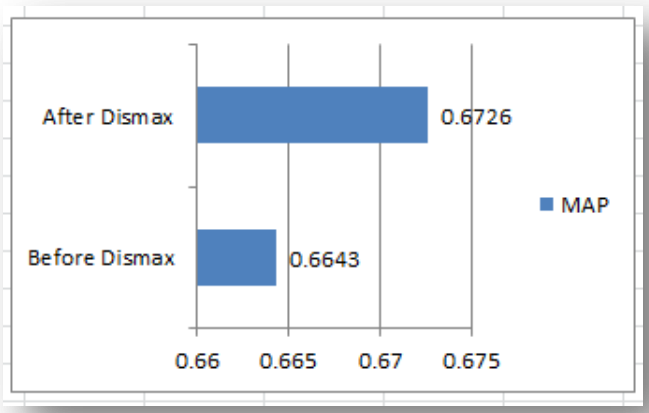
2. **Using the DisMax query parser and boosting fields**

   The standard query parser searches mainly in the default fields specified in the config, and returns the results, depending on the cumulative score from all the fields for a given document.

However, the DisMax Query Parser searches for a given query across all the fields of a document, and allows us to boost certain fields if there is a strong phrase match in that document.

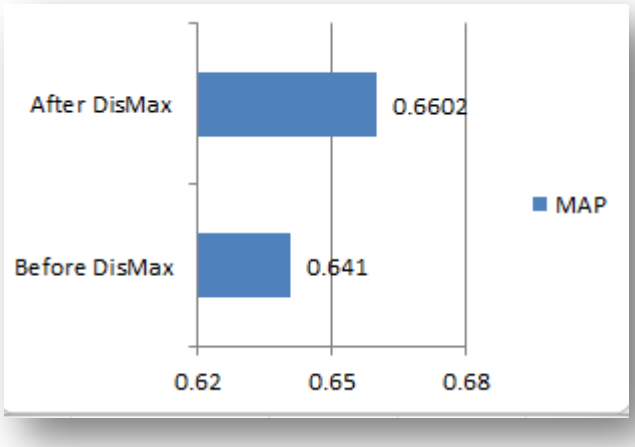
Hence we have used the DisMax Query Parser in our query, adding it using defType=dismax in the query, and boosted the fields tweet_hashtags and text_all, since hashtags definitely should have more weightage in a tweet, since they are usually relevant terms, and text_all would match phrases much better than having tokenized fields.

Hence we have also added the following clauses in our query, qf=text_all^10.0+tweet_hashtags^12.0+text_en+text_de+text_ru&pf=text_all^100



For BM25, Before & After

For VSM, before and after DisMax



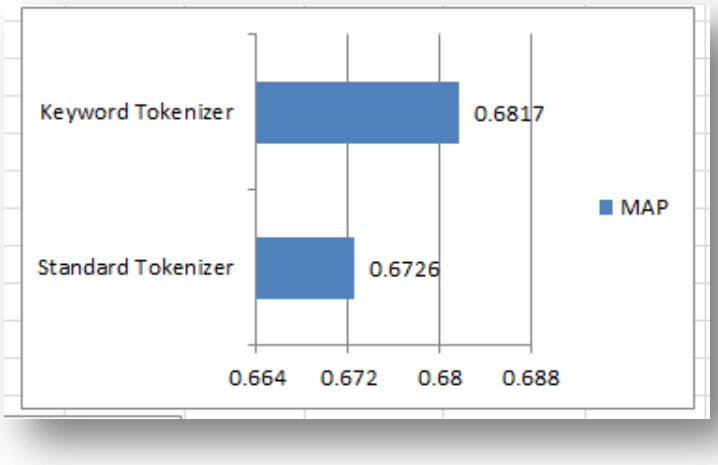
3. Using various filters and tokenizers

We have used the **UAX29URLEmailTokenizerFactory** for all default text fields: text_en, text_de and text_ru. This breaks down the tweet texts into tokens at the time of indexing, and will also be used to break query terms into tokens when this field is queried upon.

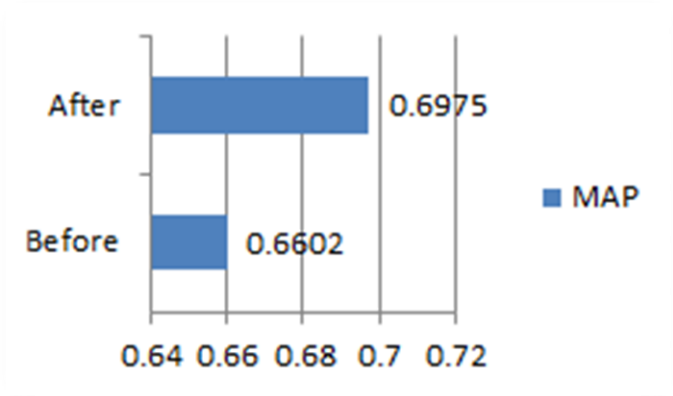
The field text_all uses the **KeywordTokenizerFactory**, which does not break the text into tokens, and stores them as phrases, which in effect help us match the queries better. This has shown a considerable boost in the MAP score for BM25, however, the same change does not go well with VSM and DFR, and decreases the MAP, NDCG and F measure scores. However, because of the considerable boost for BM25, we have kept the above tokenizer for BM25, and used the standard one for VSM and DFR.

Other additions include the use of a **LowerCaseFilter**, **Stopword** Filter,**Stemmer** and Porter Filters, and **RemoveDuplicateTokens** together for the fields text_en, text_ru and text_de, and just the **LowerCaseFilter** for text_all. These are applied both at query and indexing time.

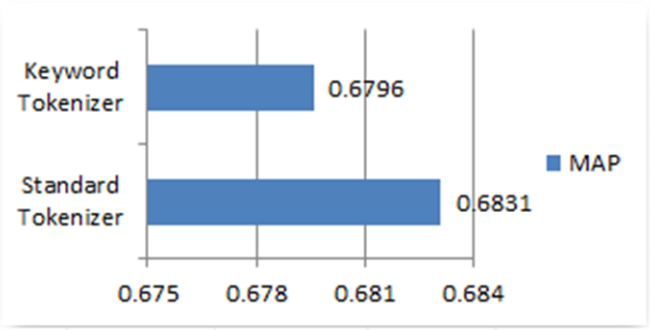
For BM25



ForVSM



For DFR



4. Analysis of the results, and comparison with the relevant tweets:

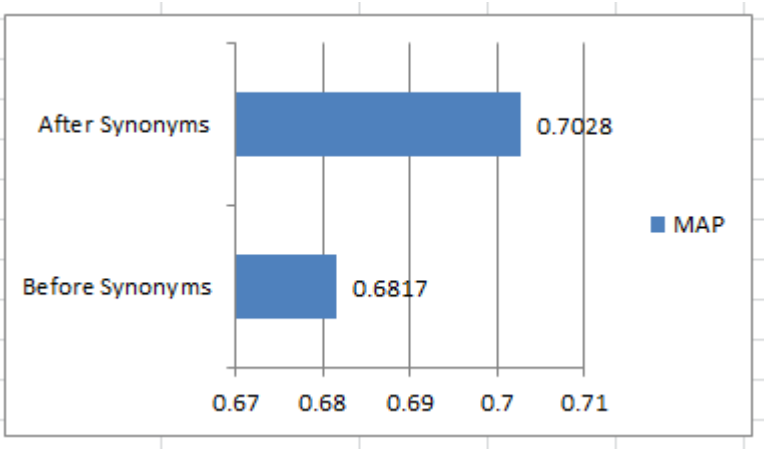
After implementing all the above improvements, we still thought we should get more relevant tweets, but weren't. So we analysed the results which we got and compared them with the relevant tweets which were marked in the QREL file given to us. What we found was that there were a lot of words which may be relevant, but were not getting captured while searching in Solr. Examples such as:

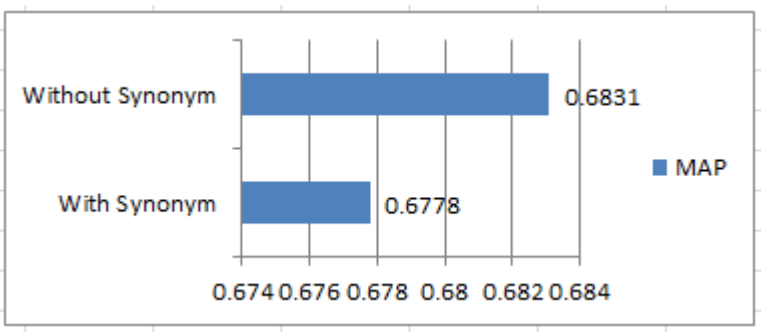
Refugee->**Migrant**
Ammo->**Missiles**,**weapons**,**guns**

Also, many relevant tweets were in languages other than the query language. Hence we thought it would be great if we could translate the original queries into the three languages and then query on them, however, we were not able to implement such a translation, but we're sure this concept would work wonders for the system.

We did however use Synonyms for expanding the query to other relevant words as stated above, and also for multi-language query expansion, in the form of say, if I query Putin, then Solr would also search for путин, which is the Russian translation for Putin. Likewise, many words which are relevant to the topics of all the given queries, such as Syria, Obama,USA,Refugees,Botschaft, Germany, Merkel, were translated into English, Russian and German, and added as synonyms in the synonyms.txt file.
This gave us a significant boost in relevance by around 4% in BM25, and VSM. However, DFR did not take so kindly to the use of synonyms and the MAP and ndcg values decreased. However, the technique we implemented is more of a implementation of synonyms for a given topic, in this case Syrian War/Refugee Crisis. This can be expanded to a more general usage too.

Usage of synonyms for BM25, significantly increased the MAP value





Using Synonyms with DFR decreases MAP

Usage of synonyms helped us increase the MAP score to 0.7028 for BM25 and to 0.6975 for VSM. Hence we can infer that this change is desirable.

An example of the multiple changes implemented for text_de

```
<fieldType name="text_de" class="solr.TextField" positionIncrementGap="100">
  <analyzer>
    <tokenizer class="solr.UAX29URLEmailTokenizerFactory"/>
    <filter class="solr.LengthFilterFactory" min="3" max="100" />
   <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="lang/stopwords_de.txt" format="snowball" />
<filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="true"/>
   <filter class="solr.GermanNormalizationFilterFactory"/>
    <filter class="solr.GermanLightStemFilterFactory"/>
    <filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
  </analyzer>
```

Summary of improvements:

1. Created CopyField text_all to store text from tweets as complete phrases
2. Tried various tokenizers and filters for text fields
3. Use of DisMax Query Parser
4. Boosting of query terms and phrase queries
5. Boosting of hashtags
6. Use of synonyms to match context better and increase relevance
7. Tuning of parameters for BM25 and DFR