IR PROJECT 3
REPORT

Team No: 84

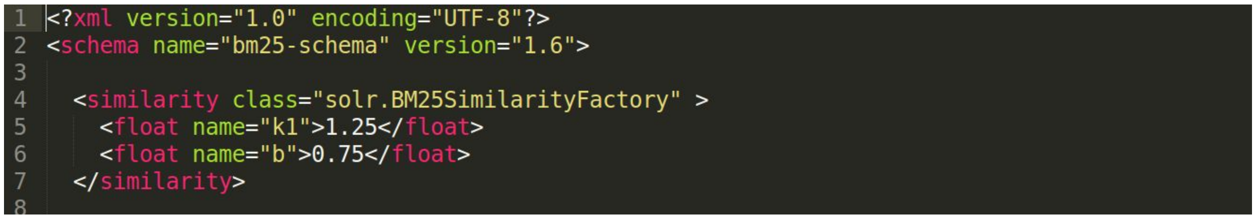
Team Members:
Nikhil Prakash (nikhilpr@buffalo.edu - 50208057)
Junaid Shaikh (junaidsh@buffalo.edu - 50208476)

Model Implementation

The IR Similarity models have been implemented by creating a separate core in solr for each model. We have used schema.xml (instead of schema less mode) to define fields and field types and the similarity definition for each model.

BM25 MODEL

We have overridden the default value of “k1” and “b” to improve search results.

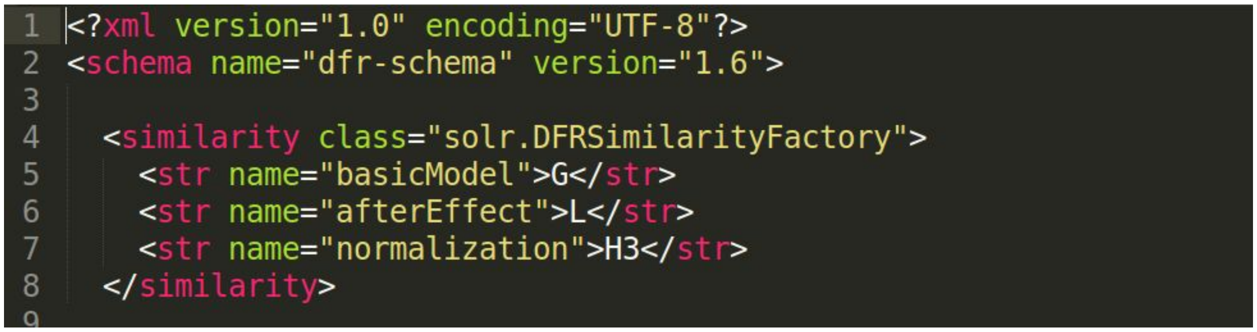


```
<?xml version="1.0" encoding="UTF-8"?>
<schema name="bm25-schema" version="1.6">

  <similarity class="solr.BM25SimilarityFactory" >
    <float name="k1">1.25</float>
    <float name="b">0.75</float>
  </similarity>
```

Fig: Screenshot of schema.xml for BM25 model

DFR MODEL

As per the project requirements, we have chosen ‘BasicModelG’ (G) as the basic Model, ‘Bernoulli First Normalization’ (B) for after Effect parameter and finally ‘H2’ second normalization for normalization parameter. We have chosen the value of c as 7. However, for our implementation we have found some improvements for the Basic Model (G)’, after effect ‘(L)’ and ‘(H3)’ as the normalization parameter.

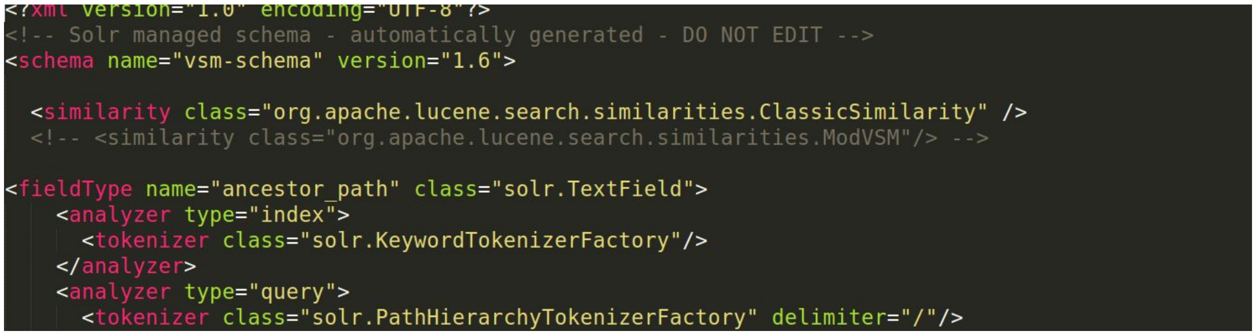


```
<?xml version="1.0" encoding="UTF-8"?>
<schema name="dfr-schema" version="1.6">

  <similarity class="solr.DFRSimilarityFactory">
    <str name="basicModel">G</str>
    <str name="afterEffect">L</str>
    <str name="normalization">H3</str>
  </similarity>
```

Fig: Screenshot of schema.xml for DFR model

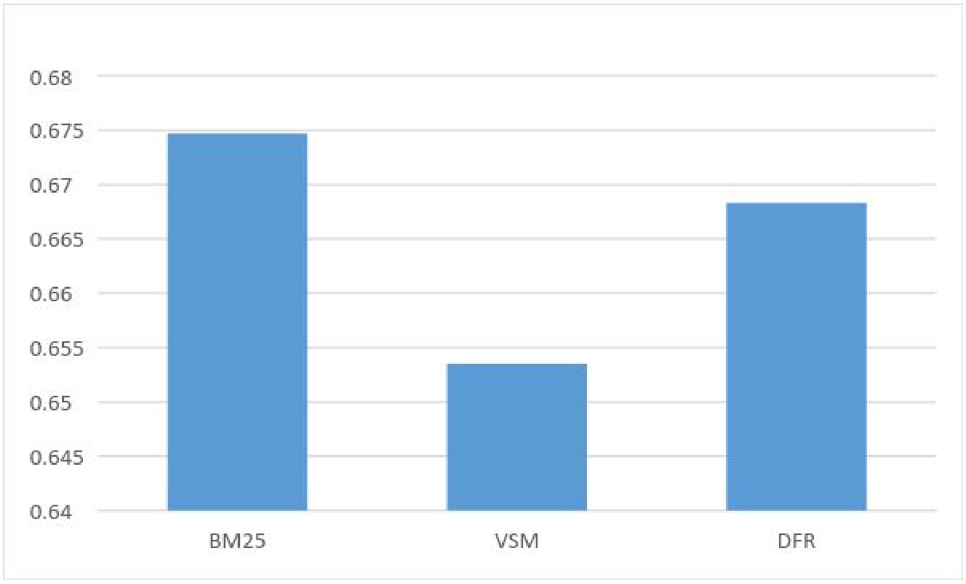
VSM MODEL

Vector Space Model which is named as ClassicSimilarity Model, is the third model to be implemented. Screenshot shown below:



```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Solr managed schema - automatically generated - DO NOT EDIT -->
<schema name="vsm-schema" version="1.6">

  <similarity class="org.apache.lucene.search.similarities.ClassicSimilarity" />
  <!-- <similarity class="org.apache.lucene.search.similarities.ModVSM"/> -->

<fieldType name="ancestor_path" class="solr.TextField">
    <analyzer type="index">
      <tokenizer class="solr.KeywordTokenizerFactory"/>
    </analyzer>
    <analyzer type="query">
      <tokenizer class="solr.PathHierarchyTokenizerFactory" delimiter="/"/>
```

Fig: Screenshot of schema.xml for VSM model



Graph: Map values after initial implementation

Techniques used to improve MAP (Mean Average Precision)

Stages:

1) Query language translation
2) URL filtering
3) Model specific improvement - BM25
4) Boosting query fields
5) Exact case matching
6) Synonyms
7) Model specific improvement - DFR
8) Advanced text filtering

1) Query Language Translation

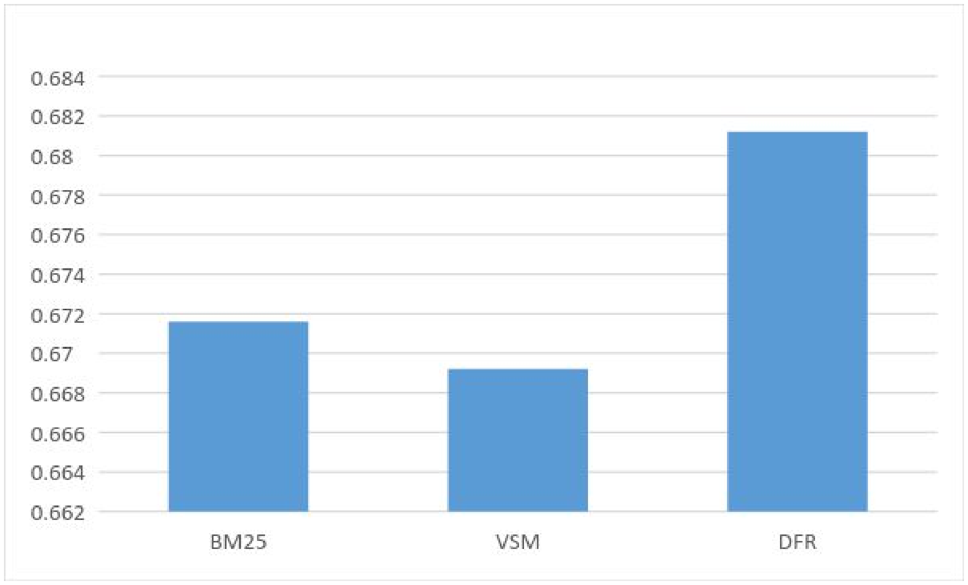
Initial Analysis

Our initial analysis involved finding global parameters that were affecting retrieval of relevant documents for the given set of queries. On closer inspection, we found that for the many of english queries, solr wasn't returning relevant documents in other languages thus affecting the overall precision. And so, we added translations of each query as a set of Boolean OR operations. I.e.

```
lang:en
    text_en:(Query in English) OR text_de:(Query in
        German) OR text_ru:(Query in Russian)
```

Post Implementation Analysis

Next we evaluated the increase in average precision over the entire set of queries (or MAP). Query language translation, as we saw later on, stood as the biggest stimulus for improving the MAP value.



Graph: Map values after Query Translation

2) <u>URL Filtering</u>

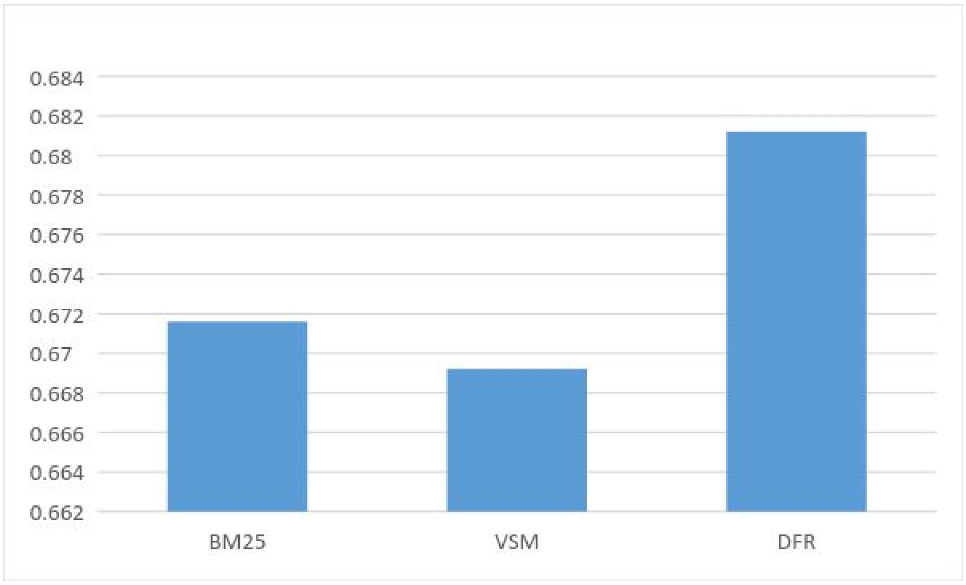
<u>Initial Analysis</u>

The default search field for the set of given queries was set as the text field in each of the three languages. I.e. text_en, text_de and text_ru. These fields, besides containing text that queries can match against for relevancy, also contained data that was irrelevant and did not contribute towards relevancy. E.g. the urls within the text field were irrelevant was contributing negatively towards average precision of the system and thus needed to be removed.

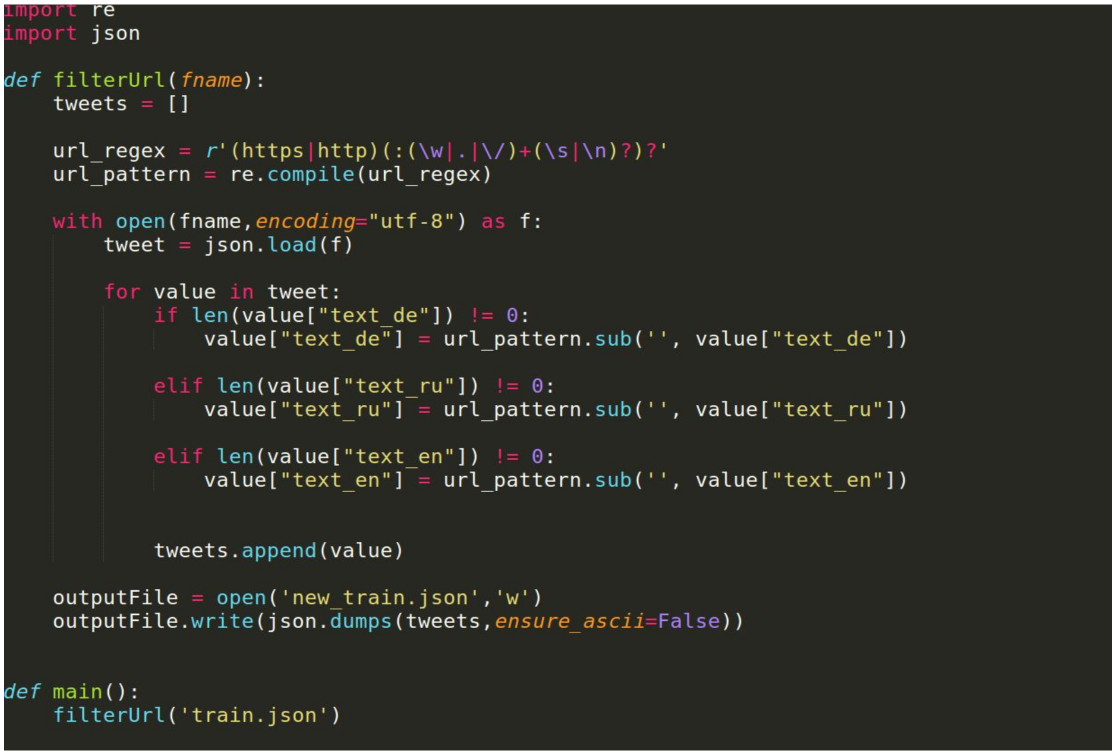
We wrote a python script to remove urls from each tweet (using regex) and then use this as the new set of documents to the search system.

<u>Post Implementation Analysis</u>

On removing the urls from the documents, the overall efficiency of the search system improved in terms of MAP value. A screenshot of the script is shown below. But the overall effect of removing urls weren't seen for 20 documents as opposed to when retrieving 1000 documents for a query. This gives us an idea that this technique works when a large number of relevant documents needs to be retrieved. But if the search involves retrieving few documents, then url filtering does not improve the system.



Graph: Map values after URL Filtering



```
import re
import json

def filterUrl(fname):
    tweets = []

    url_regex = r'(https|http)(:(\w|.|\/)+(\s|\n)?)?'
    url_pattern = re.compile(url_regex)

    with open(fname,encoding="utf-8") as f:
        tweet = json.load(f)

        for value in tweet:
            if len(value["text_de"]) != 0:
                value["text_de"] = url_pattern.sub('', value["text_de"])

            elif len(value["text_ru"]) != 0:
                value["text_ru"] = url_pattern.sub('', value["text_ru"])

            elif len(value["text_en"]) != 0:
                value["text_en"] = url_pattern.sub('', value["text_en"])


            tweets.append(value)

    outputFile = open('new_train.json','w')
    outputFile.write(json.dumps(tweets,ensure_ascii=False))


def main():
    filterUrl('train.json')
```

Fig: Screenshot of tweet_filter.py

3) <u>Model Specific Improvement – BM25</u>

<u>Initial Analysis</u>

The formula for BM25 Retrieval Model is given by:

$$IDF * ((k + 1) * tf) / (k * (1.0 - b + b * (|d|/avgDl)) + tf)$$

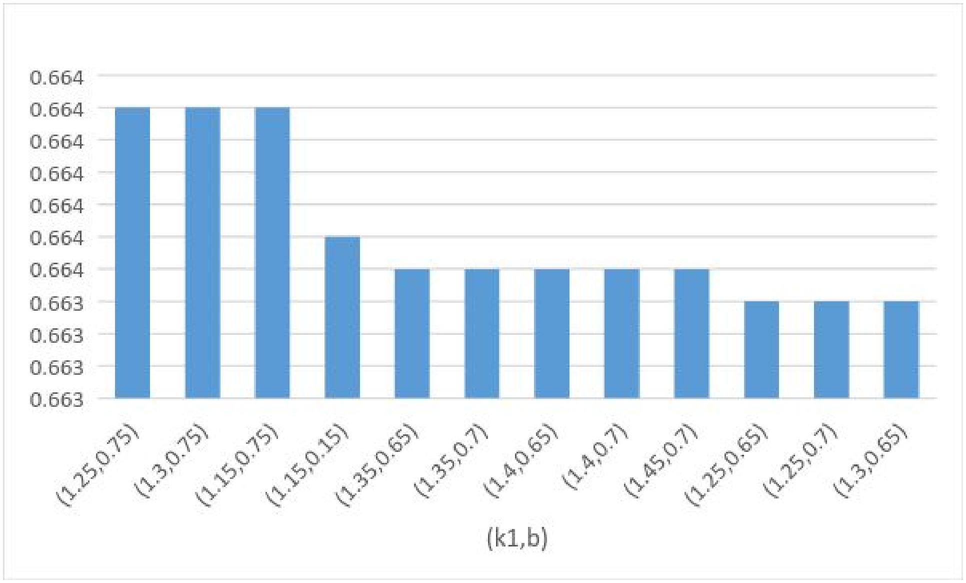
L - how long a document is relative to the average document length
|d|/avgDl - this document length divided by the average document length

The constant b will allow us to finely tune how much influence our L value has on scoring. The value of k denotes TF (term frequency) in the document. The default value of k is 1.2. But we fine-tuned this value to 1.25

as per our system (set of queries and documents). We wrote a script to substitute different values for k1 and b, restart solr, post the documents and then query and give the map values. We then selected values of k1 and b according to the results.

Post Implementation Analysis

The overall map value for the BM25 model increased by 0.004.



Graph: Values of map for different values of k1 and b

4) Boosting Query Fields

Initial Analysis

Next we boosted the language dependent fields. E.g. If the query was originally in English, then the field **text_en** will be boosted compared to text_de and text_ru since intuition tells us that more documents of the same language as the original query will be retrieved. This task was performed using an advanced query parser called **Dismax.** Dismax is designed to process simple phrases entered by users and to search for individual terms across several fields using different weighting (boosts) based on the significance of each field. The **qf** (Query Field) parameter within Dismax specifies the fields in the index on which to perform the query. And so, we made use of qf parameter to boosts fields depending on the language. The boosts that were made is shown below:

If query language -> English
qf = tweet_hashtags^2.04 **text_en^2.25** text_de^2.0 text_ru^2.0

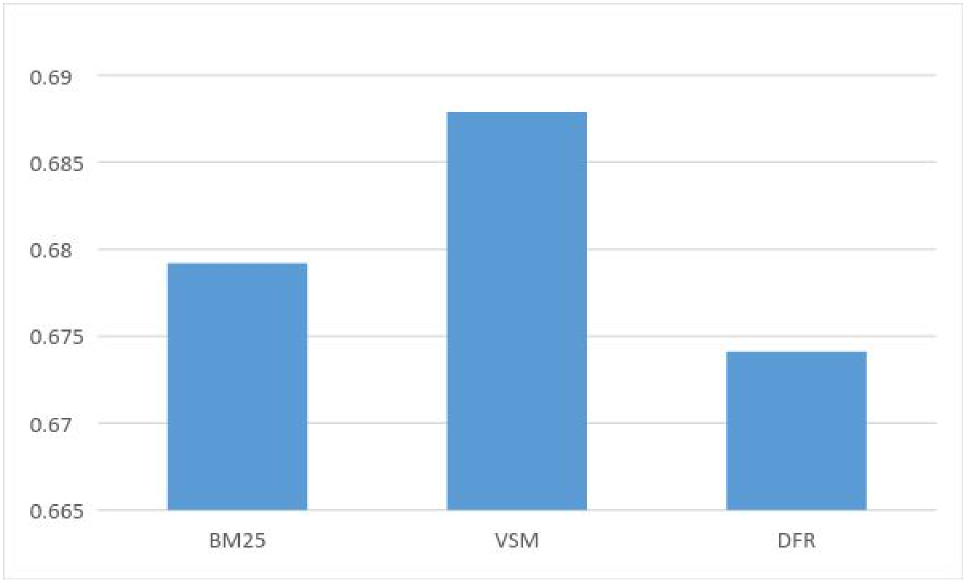
If query language -> German
qf = tweet_hashtags^2.04 text_en^2.0 **text_de^2.25** text_ru^2.0

If query language -> Russian
qf= tweet_hashtags^2.04 text_en^2.0 text_de^2.0 **text_ru^2.25**

Also, it can be seen from the values being assigned to qf that the field **tweet_hashtags** has been boosted since we know that hashtags succinctly summarises the whole tweet in a single word and thus documents finding a match with this field can be ranked higher (given higher score).

Post Implementation Analysis

On an average, query field boosting increased the overall MAP value. Though this improvement wasn’t consistent across MAP values for all the 20 queries since there could have been queries which had lesser number of relevant documents in the same language and more relevant documents in a different language. But, on an average, the overall MAP values improved.



Graph: Map values after boosting query fields

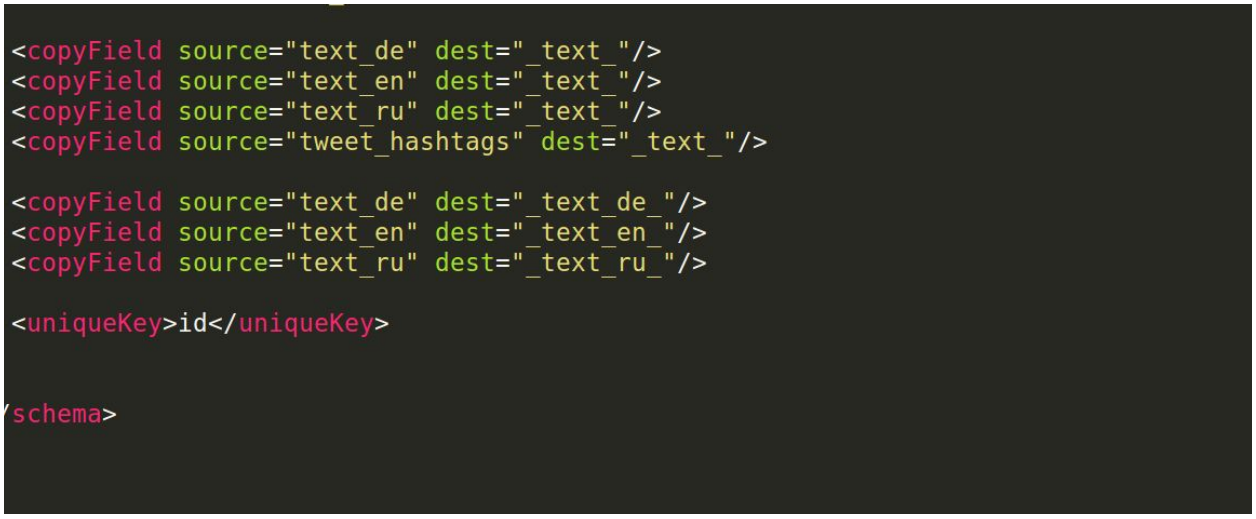
5) <u>Exact Case Matching</u>

<u>Initial Analysis</u>

It made sense that words in a query and a document with the exact case matching must score higher. And thus we added a strategy to index the same field twice, one which had been filtered using Lower Case Filtering and the other in which no lower case filtering had been done. Next we copied these fields using the copyField command in schema to index a single input field multiple times. Finally, the query was applied across both the fields.

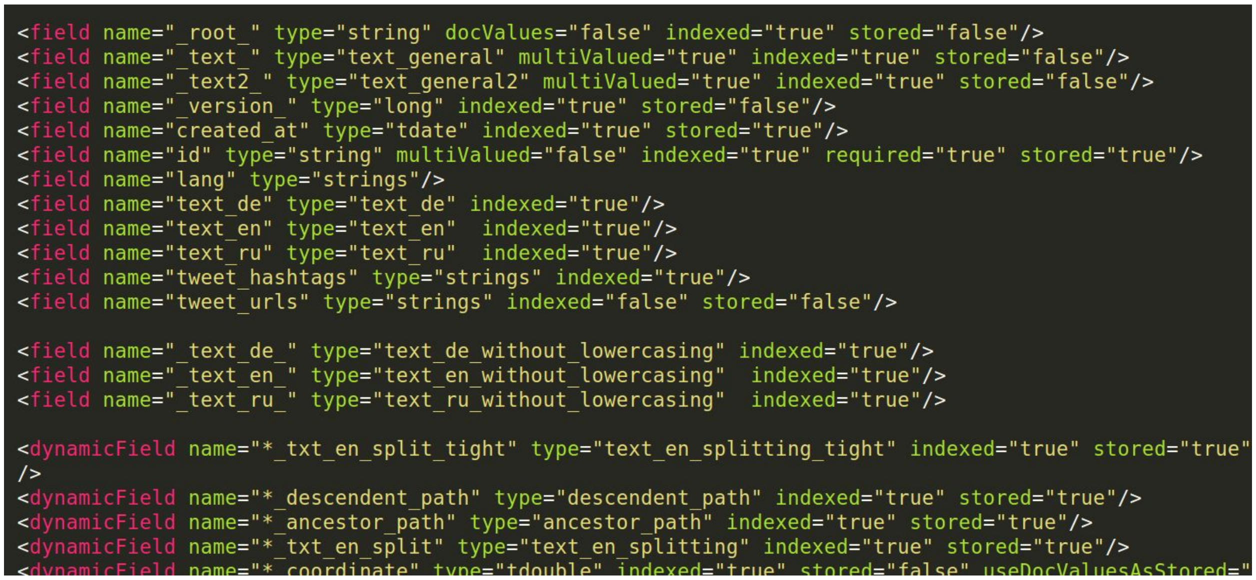
<u>Post Implementation</u>

This implementation resulted in a slight increase in overall MAP value. A screenshot of the implementation is shown below:



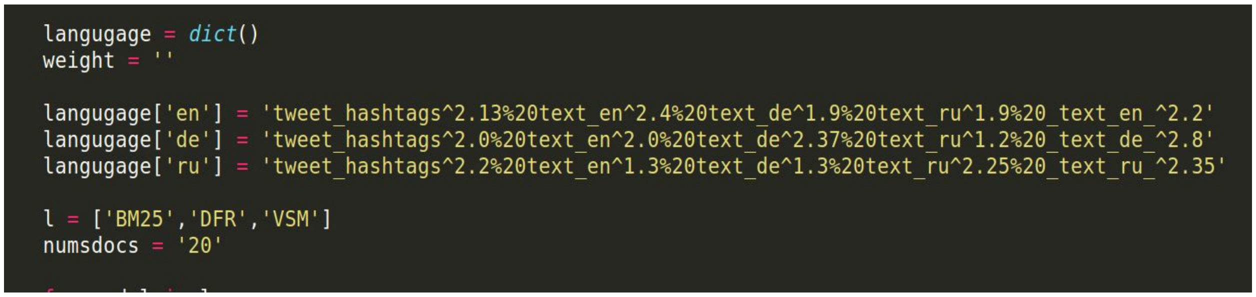
```
<copyField source="text_de" dest="_text_"/>
<copyField source="text_en" dest="_text_"/>
<copyField source="text_ru" dest="_text_"/>
<copyField source="tweet_hashtags" dest="_text_"/>

<copyField source="text_de" dest="_text_de_"/>
<copyField source="text_en" dest="_text_en_"/>
<copyField source="text_ru" dest="_text_ru_"/>

<uniqueKey>id</uniqueKey>


/schema>
```

Fig: Screenshot of copyFields in action



```
<field name="_root_" type="string" docValues="false" indexed="true" stored="false"/>
<field name="_text_" type="text_general" multiValued="true" indexed="true" stored="false"/>
<field name="_text2_" type="text_general2" multiValued="true" indexed="true" stored="false"/>
<field name="_version_" type="long" indexed="true" stored="false"/>
<field name="created_at" type="tdate" indexed="true" stored="true"/>
<field name="id" type="string" multiValued="false" indexed="true" required="true" stored="true"/>
<field name="lang" type="strings"/>
<field name="text_de" type="text_de" indexed="true"/>
<field name="text_en" type="text_en"  indexed="true"/>
<field name="text_ru" type="text_ru"  indexed="true"/>
<field name="tweet_hashtags" type="strings" indexed="true"/>
<field name="tweet_urls" type="strings" indexed="false" stored="false"/>

<field name="_text_de_" type="text_de_without_lowercasing" indexed="true"/>
<field name="_text_en_" type="text_en_without_lowercasing"  indexed="true"/>
<field name="_text_ru_" type="text_ru_without_lowercasing"  indexed="true"/>

<dynamicField name="*_txt_en_split_tight" type="text_en_splitting_tight" indexed="true" stored="true"
/>
<dynamicField name="*_descendent_path" type="descendent_path" indexed="true" stored="true"/>
<dynamicField name="*_ancestor_path" type="ancestor_path" indexed="true" stored="true"/>
<dynamicField name="*_txt_en_split" type="text_en_splitting" indexed="true" stored="true"/>
<dynamicField name="*_coordinate" type="tdouble" indexed="true" stored="false" useDocValuesAsStored="
```

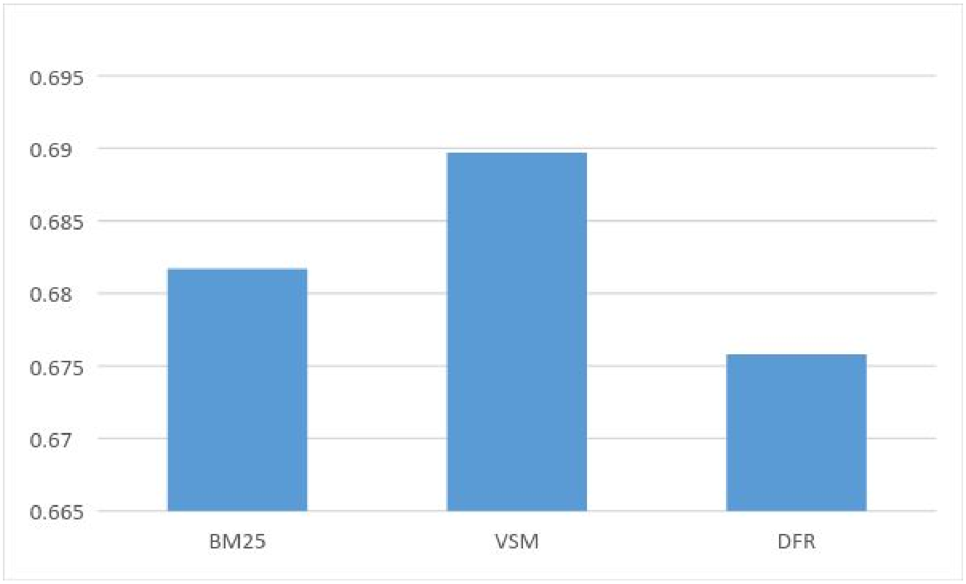
Fig: Screenshot of Exact case matching



```
langugage = dict()
weight = ''

langugage['en'] = 'tweet_hashtags^2.13%20text_en^2.4%20text_de^1.9%20text_ru^1.9%20_text_en_^2.2'
langugage['de'] = 'tweet_hashtags^2.0%20text_en^2.0%20text_de^2.37%20text_ru^1.2%20_text_de_^2.8'
langugage['ru'] = 'tweet_hashtags^2.2%20text_en^1.3%20text_de^1.3%20text_ru^2.25%20_text_ru_^2.35'

l = ['BM25','DFR','VSM']
numsdocs = '20'
```

Fig: Screenshot - Boosting **_text_en_ _text_de_ and _text_ru_**

As seen from the screenshot above, three new fields - **_text_en_, _text_de_ and _text_ru_** were created with the same content as text_en, text_de and text_ru but with a different analyzer. i.e Lower case filtering has been removed from the first three fields. And so during query matching, if the document contains terms that has the same case structure as the original query, then it will assign a relatively higher score.



Graph: Map values after exact case matching

6) <u>Synonyms</u>

<u>Initial Analysis</u>

Since phrases in a language can be worded differently, a document could potentially contain a given query but phrased differently. And so we took up the task of adding synonyms.txt file in solr. And so the query parser then would rephrase the original query as disjunctions of the original terms and its synonyms.

If the original query contains a single word K, the synonyms of the keyword would be added as disjunctions. i.e. if Q = K

Qe = (K) V (Synonym 1) V (Synonym 2)

Post Implementation

The addition of synonyms to the system greatly improved the MAP values in all three models. The improvement is shown as a graph towards the end of this report.

7) Model Specific Improvement - DFR

Initial Analysis

The main idea behind DFR is that 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 the document d.

i.e.

weight(t|d) is proportional to -log(Prob_M(t in d | Collection))

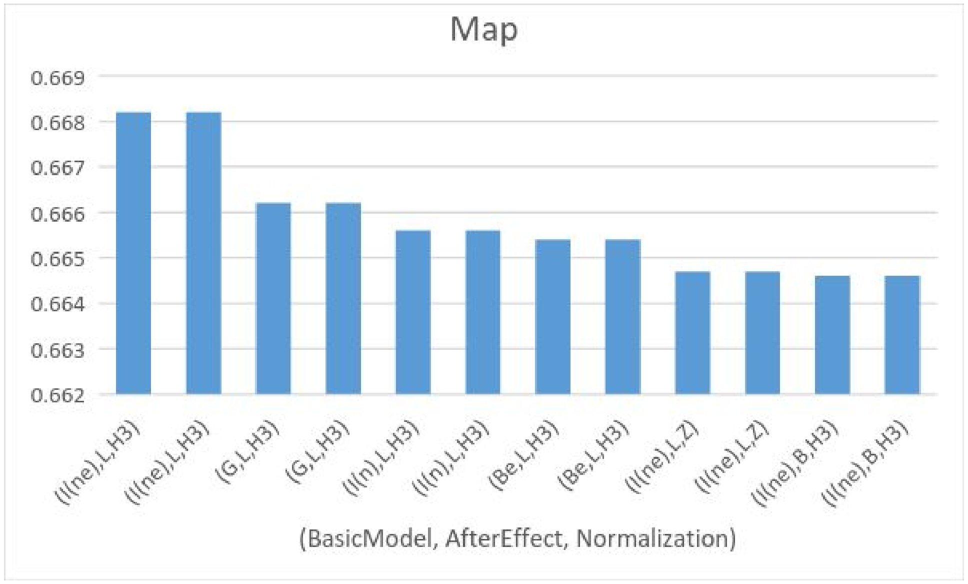
M : type of model of randomness employed to compute the probability

Various potential values for the parameters basicModel, afterEffect and normalization were attempted.

D	Divergence approximation of the binomial
P	Approximation of the binomial
BE	Bose-Einstein distribution
G	Geometric approximation of the Bose-Einstein
I(n)	Inverse Document Frequency model
I(F)	Inverse Term Frequency model
I(ne)	Inverse Expected Document Frequency model

<u>Post Implementation Analysis</u>

After trying various combinations of the three parameters, we found the most compatible combination to be (G,L,H3)



Graph: Map values for different values of BasicModel, AfterEffect, Normalization

<u>Post Implementation</u>

The MAP value for DFR increased by 0.0064.

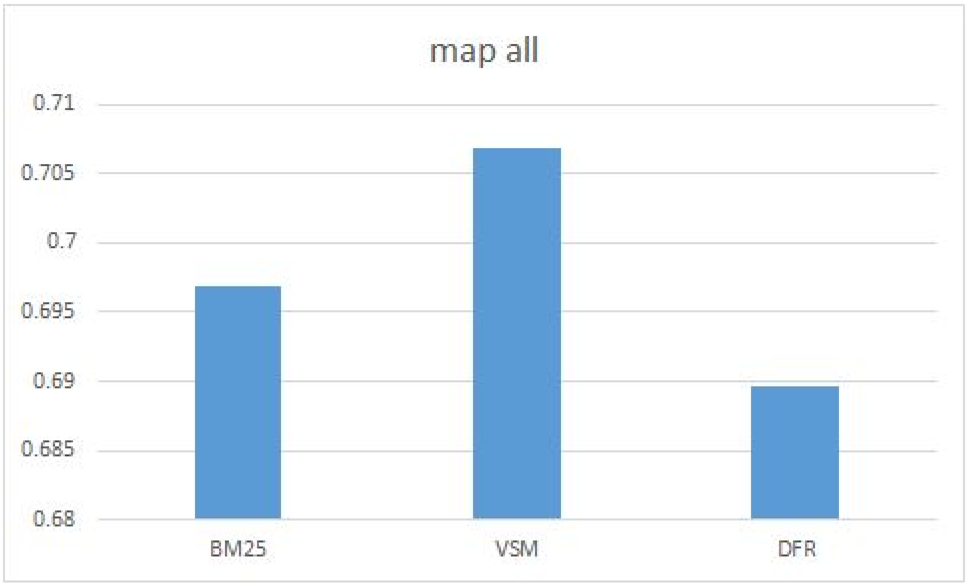
8) <u>Advanced Text Filtering</u>

<u>Initial Analysis</u>

In URL filtering, we were not able to see significant improvements in map for less number of queries. However, after the implementation of all the above steps, we experimented with the regex for the URL Filtering a bit and also replaced ‘:’ with a blank space in the tweet text and were able to fine-tune it further to give an improved map score for 20 queries.

Post Implementation Analysis

There was an increase of 0.0069, 0.0059, 0.0104 in the map values for BM25, VSM and DFR respectively.



Graph: Map values after advanced Text Filtering

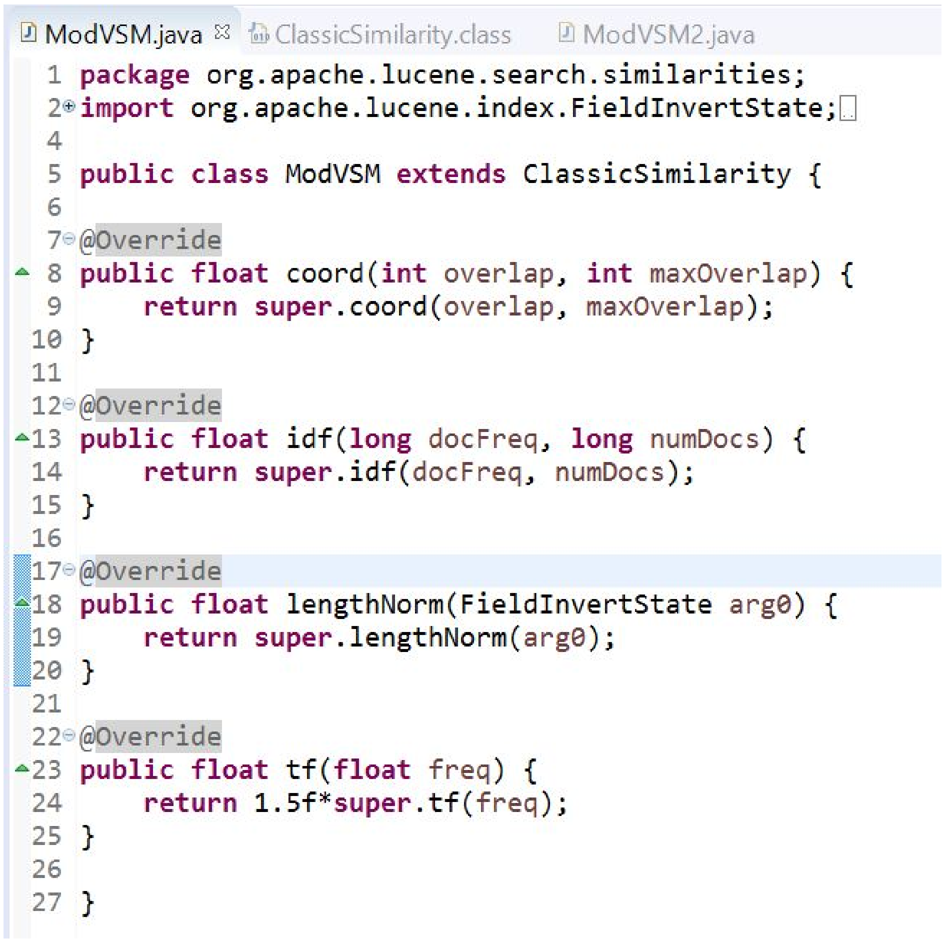
Unsuccessful Implementations

1) Plugin to translate queries online (Query Parser)

Our attempt on writing a java class that would override the QParserPlugin and QParser classes and provide online translation of any incoming query using the request object proved futile because of the difficulty in implementing such a plugin. Lack of proper examples and deprecated functionalities of lucene and solr for this purpose on the internet further made this task difficult. Finally, the same was achieved by translating each of the 20 queries offline.

2) Custom TF IDF Similarity model

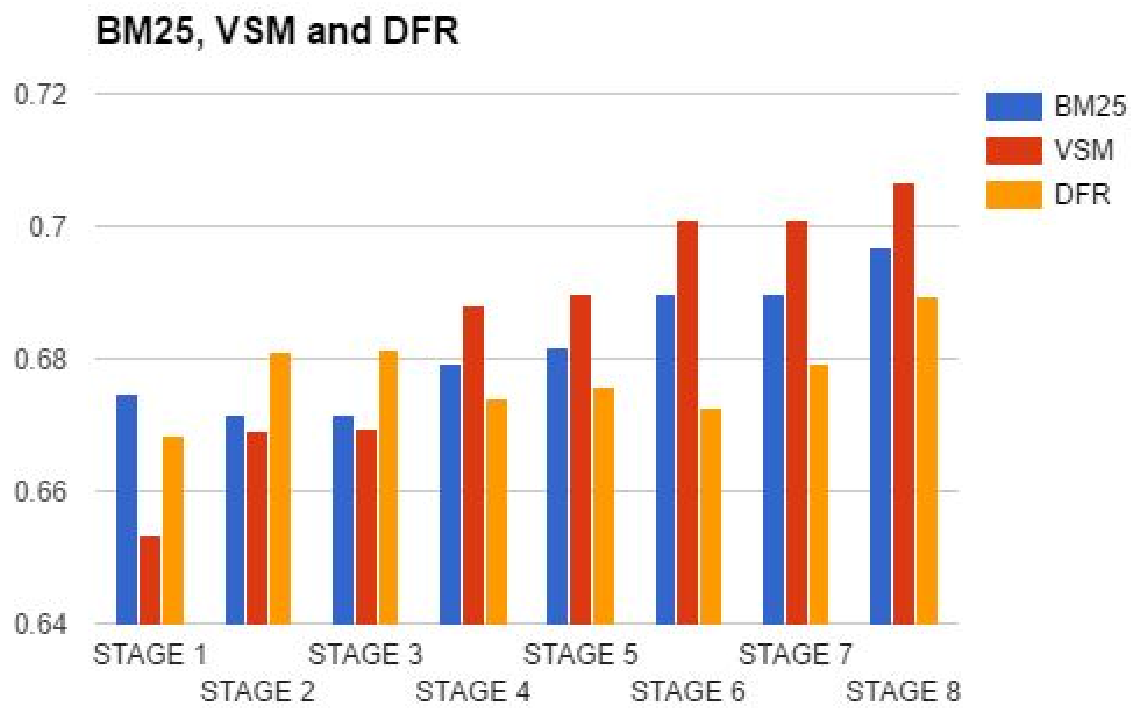
An attempt was made to extend the default VSM similarity model and to override the default tf-idf setting. Various values were tried but none gave an increase in MAP value. A screenshot is provided below of the class we implemented to override the ClassicSimilarity methods.



```java
package org.apache.lucene.search.similarities;
import org.apache.lucene.index.FieldInvertState;

public class ModVSM extends ClassicSimilarity {

@Override
public float coord(int overlap, int maxOverlap) {
    return super.coord(overlap, maxOverlap);
}

@Override
public float idf(long docFreq, long numDocs) {
    return super.idf(docFreq, numDocs);
}

@Override
public float lengthNorm(FieldInvertState arg0) {
    return super.lengthNorm(arg0);
}

@Override
public float tf(float freq) {
    return 1.5f*super.tf(freq);
}

}
```

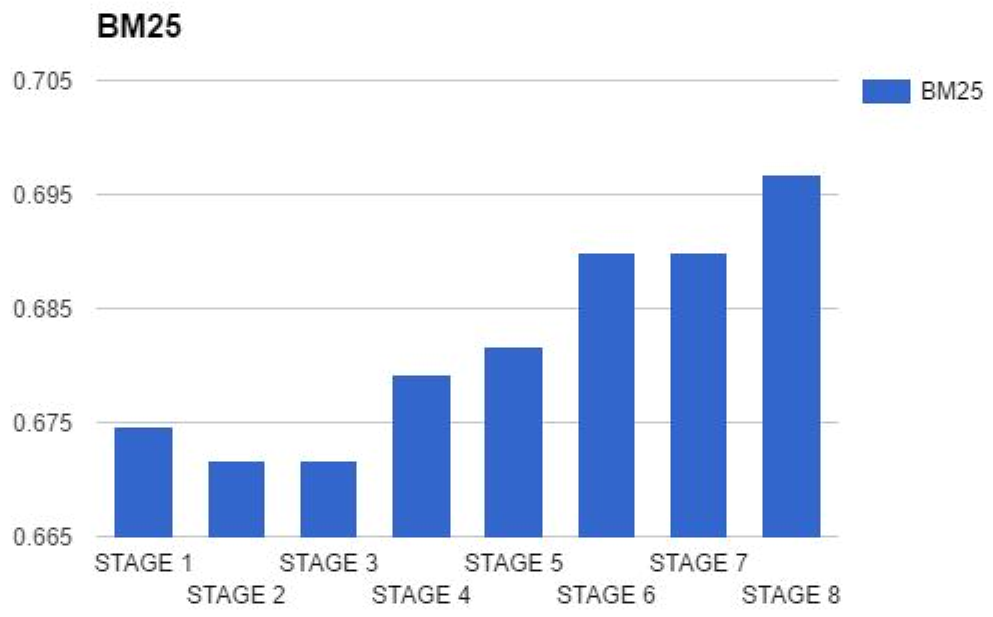
Fig: Implementation of class to override ClassicSimilarity methods

Model Comparison Graph for MAP



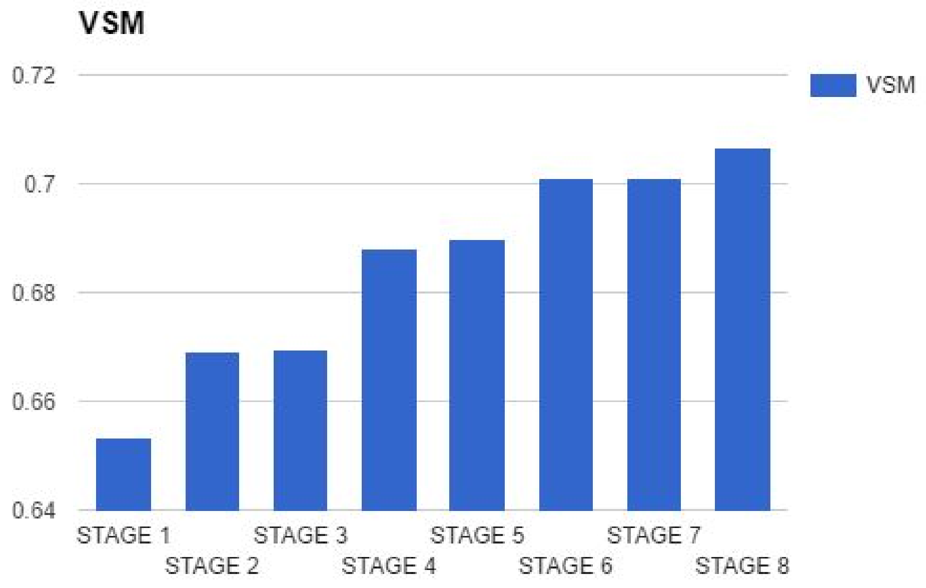
Graph: The graph shows the improvement in MAP value for each model over 8 Stages

<u>BM25 Graph</u>



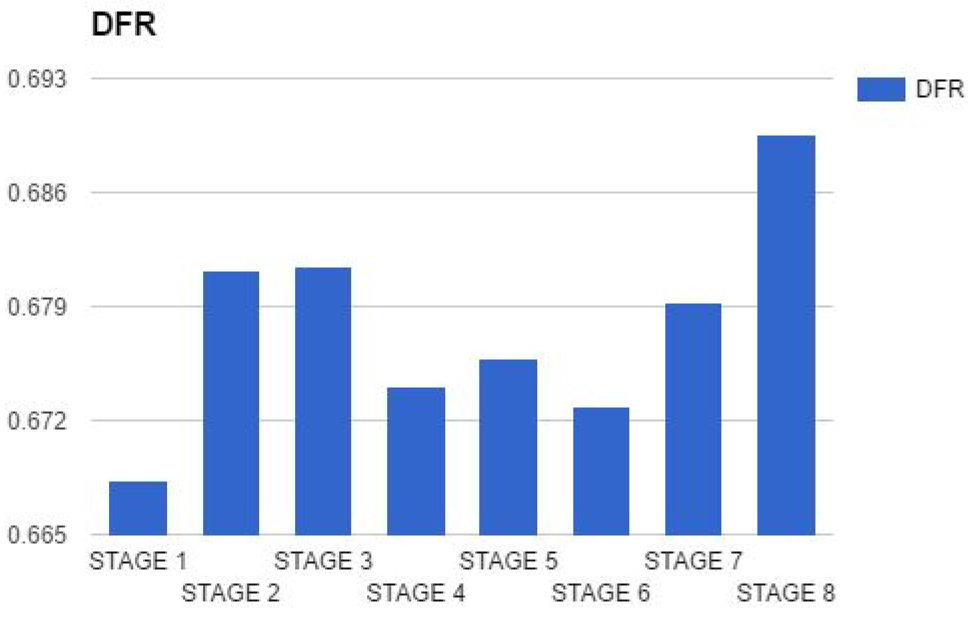
Graph: Improvement in MAP value for BM25 over 8 Stages

<u>VSM Graph</u>



Graph: Improvement in MAP value for VSM over 8 Stages

DFR Graph



Graph: Improvement in MAP value for VSM over 8 Stages

Conclusion

The 8 stages of improvement techniques elucidated above resulted in an overall improvement in MAP value. But in the case of DFR model, boosting query fields impacted the model negatively as it can be seen in the DFR MAP graph. At the end of the 8 stages however, the score has come up for DFR as well.

For a set of 20 queries and 20 rows of results, the most improvement was found using VSM model with a MAP score of 0.7068 from 0.6535, BM25 showed an improvement from 0.6747 to 0.6968 and finally DFR showed an improvement from 0.6683 to 0.6896.