PROJECT 3
INFORMATION RETRIEVAL

UBIT Name: msyed3
By: Mohammed Kamran Syed

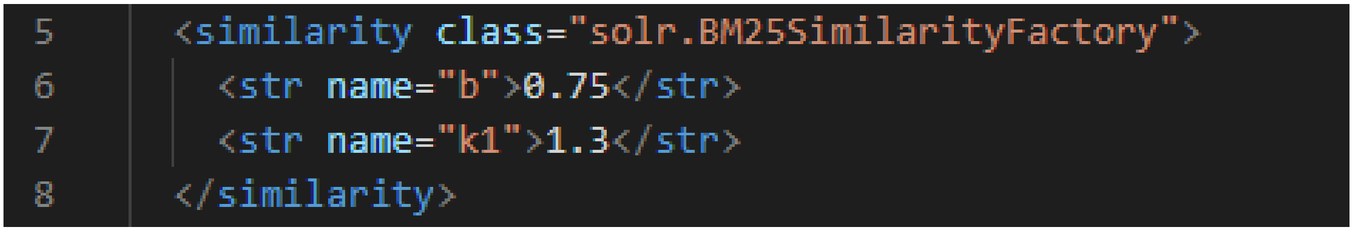
Default Implementations

The three IR model implementations were made on three different cores namely, IRF18P3BM25, IRF18P3VSM and IRF18P3DFR respectively.

For each of the above model implementations, global similarity elements have been declared in the schema configuration files.

BM25

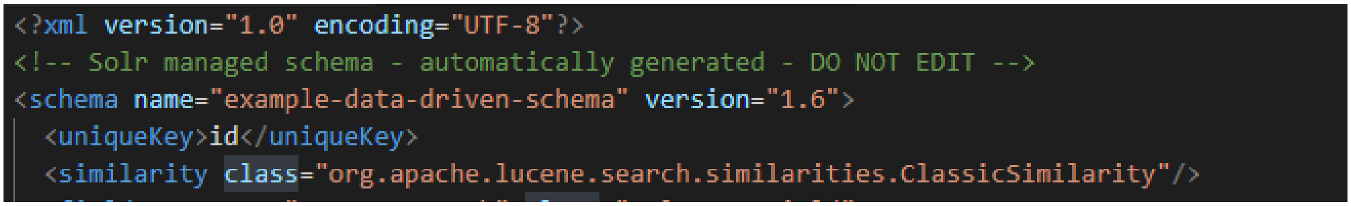
For the Solr version being used (v6.6.5), this is the default scoring model. However, additional parameters (b, k1) have been specified (which would be discussed later) to the global similarity element in the schema.xml file.



```
<similarity class="solr.BM25SimilarityFactory">
  <str name="b">0.75</str>
  <str name="k1">1.3</str>
</similarity>
```

VSM

For VSM, the ClassicSimilarity class of lucene was used in the global similarity element in the schema.xml file. This doesn't support additional parameters like the other two models.

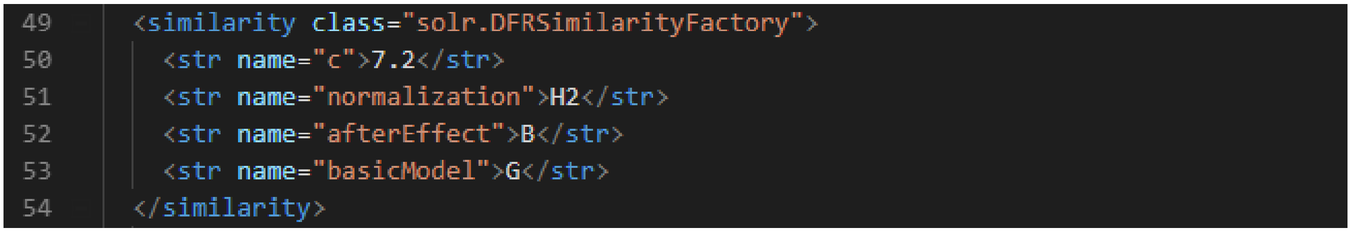


```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Solr managed schema - automatically generated - DO NOT EDIT -->
<schema name="example-data-driven-schema" version="1.6">
  <uniqueKey>id</uniqueKey>
  <similarity class="org.apache.lucene.search.similarities.ClassicSimilarity"/>
```

DFR

For DFR, the DFRSimilarityFactory class of lucene was used in the global similarity element with the following DFR specific parameters:

- Normalization: Specifies the second normalization to be applied. H2 was used in this project. A hyper-parameter called “c” that controls the term frequency normalization with respect to the document 1 was set experimentally (which would be discussed later).
- “afterEffect”: Specifies the first normalization to be applied.
- “basicModel”: Specifies the basic model to be used for DFR, which in this case was Geometric Approximation of Bose-Einstein.

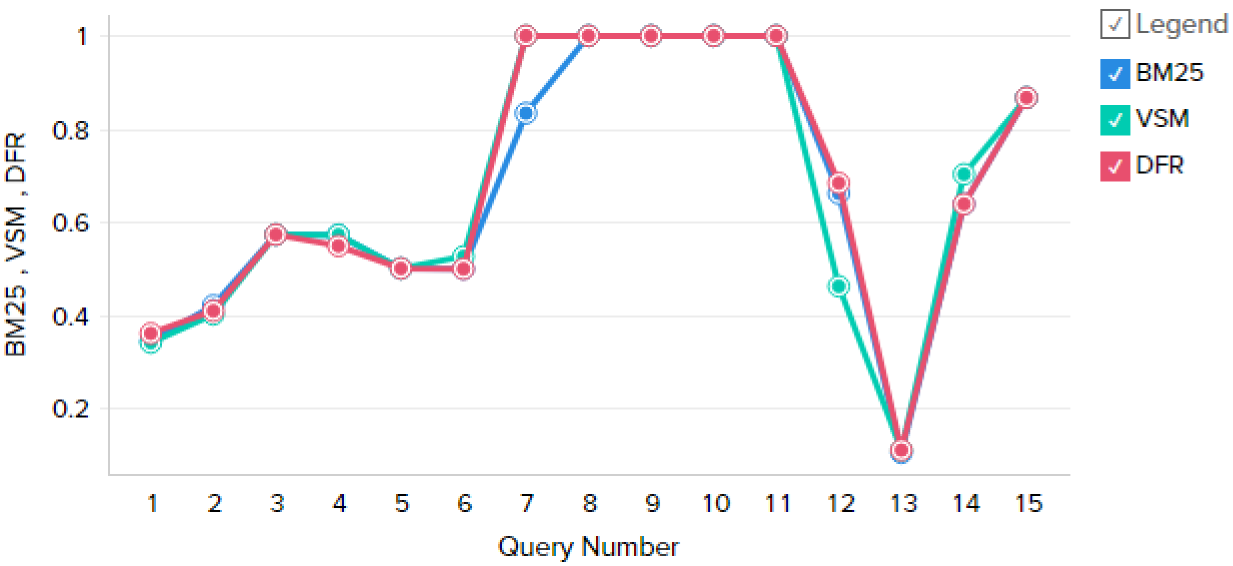


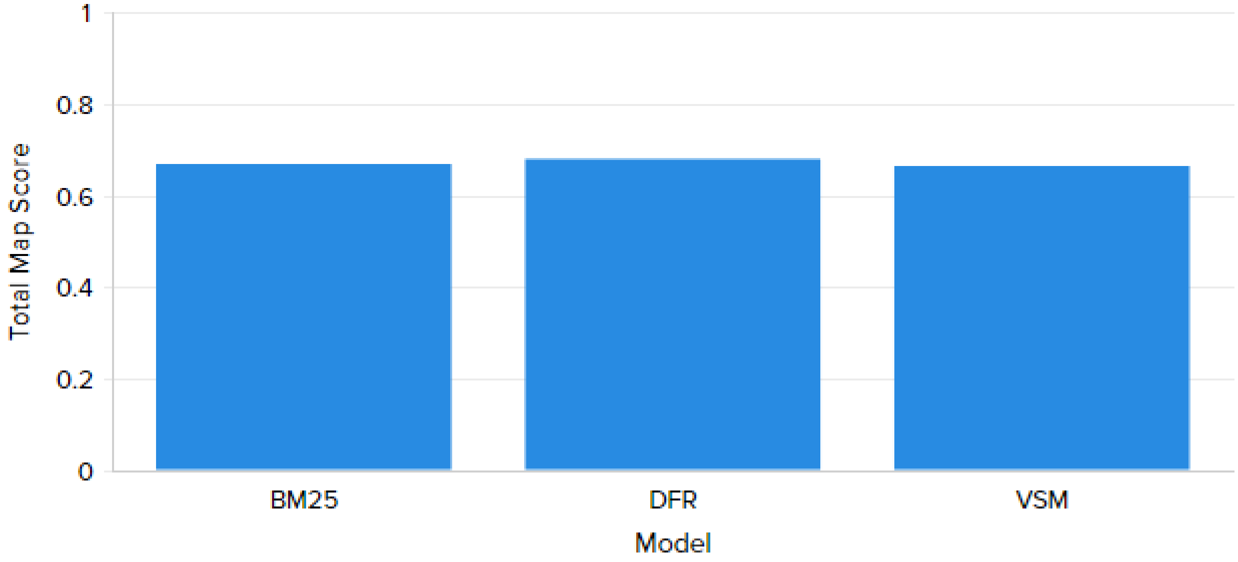
```
<similarity class="solr.DFRSimilarityFactory">
  <str name="c">7.2</str>
  <str name="normalization">H2</str>
  <str name="afterEffect">B</str>
  <str name="basicModel">G</str>
</similarity>
```

Performance Comparison:

With the default implementations of BM25, VSM and DFR models, the following are the map values are obtained:

	Map Scores		
Query Number	BM25	VSM	DFR
1	0.3433	0.3403	0.3601
2	0.4202	0.4011	0.4086
3	0.5729	0.5729	0.5729
4	0.5724	0.5724	0.5484
5	0.5	0.5	0.5
6	0.4991	0.5257	0.4991
7	0.8333	1	1
8	1	1	1
9	1	1	1
10	1	1	1
11	1	1	1
12	0.6616	0.4615	0.684
13	0.1041	0.1098	0.1098
14	0.6386	0.7028	0.6386
15	0.8667	0.8667	0.8667
all	0.6675	0.6639	0.6792





Performance Improvement Steps:

- Query Expansion: Query Language Translation, Synonyms addition
- URL Filtering
- Custom Query Parser with Query boosting
- Exact Query Matching
- Query Refinement
- Model Specific Tweaking

Query Expansion:

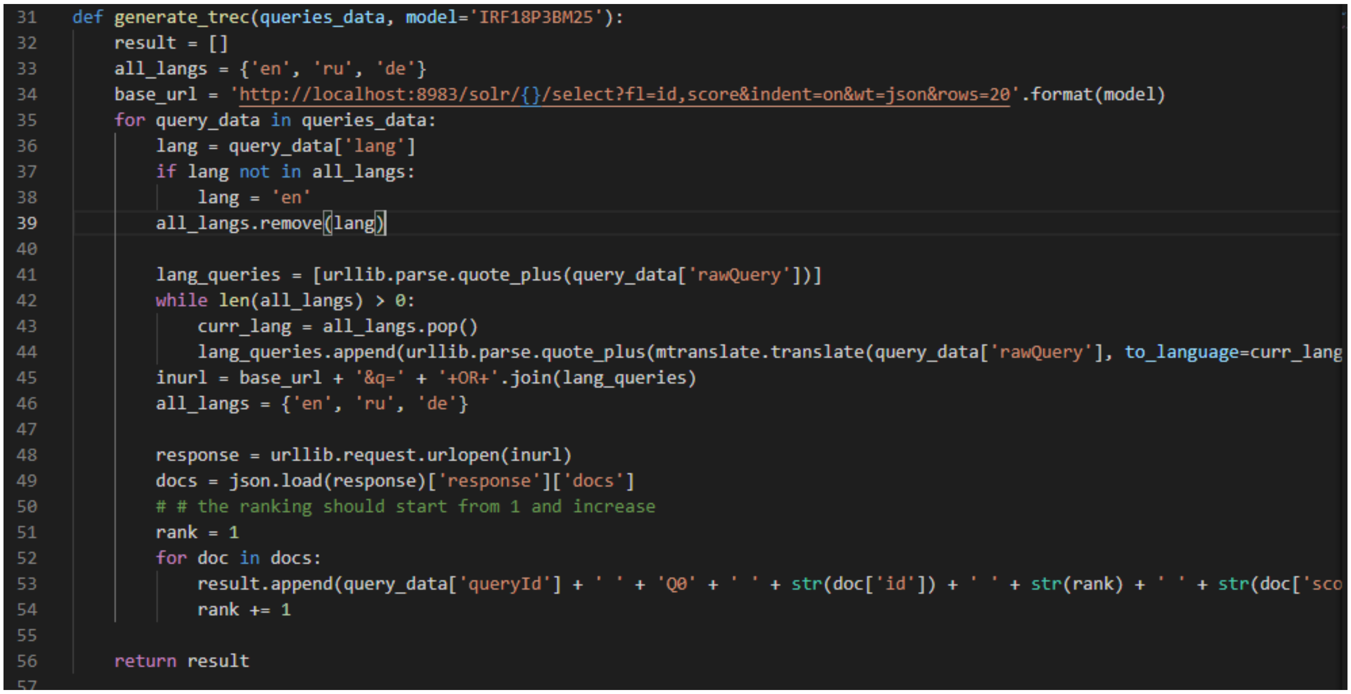
Query Language Translation

One of the major steps to improve the MAP score was to translate queries from original language of the query to the other languages. By default, solr rates the documents with same language as that of the query higher than documents in the other languages. Therefore, to improve the scores of the relevant documents in other languages, language translation was used. This was achieved by adding the query translation logic to all the languages, namely English, German and Russian, and then running the query.

This resulted in the following disjunction of queries in three languages:

Modified Query Q = (Query in English) OR (Query in German) OR (Query in Russian)

Code Snippet for Language Translation:

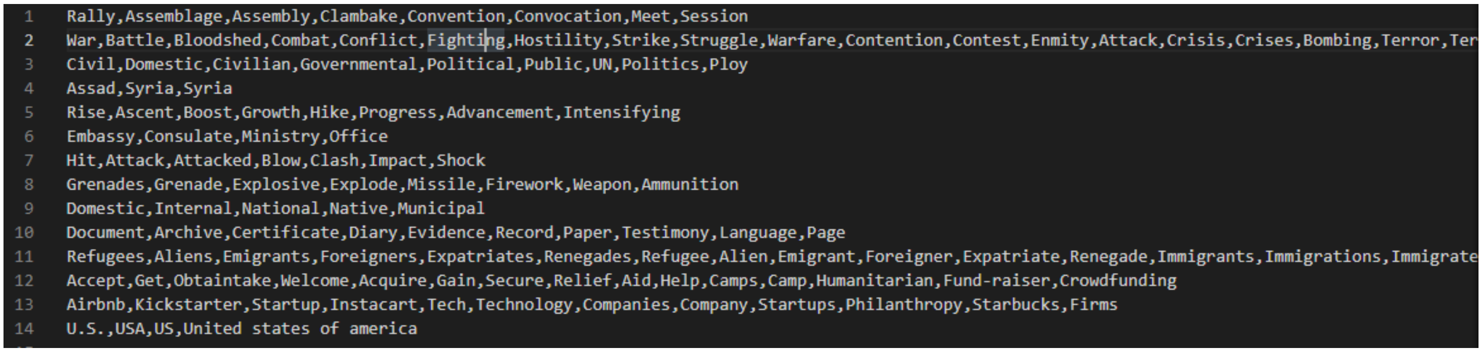


```python
def generate_trec(queries_data, model='IRF18P3BM25'):
    result = []
    all_langs = {'en', 'ru', 'de'}
    base_url = 'http://localhost:8983/solr/{}/select?fl=id,score&indent=on&wt=json&rows=20'.format(model)
    for query_data in queries_data:
        lang = query_data['lang']
        if lang not in all_langs:
            lang = 'en'
        all_langs.remove(lang)

        lang_queries = [urllib.parse.quote_plus(query_data['rawQuery'])]
        while len(all_langs) > 0:
            curr_lang = all_langs.pop()
            lang_queries.append(urllib.parse.quote_plus(mtranslate.translate(query_data['rawQuery'], to_language=curr_lang
        inurl = base_url + '&q=' + '+OR+'.join(lang_queries)
        all_langs = {'en', 'ru', 'de'}

        response = urllib.request.urlopen(inurl)
        docs = json.load(response)['response']['docs']
        # # the ranking should start from 1 and increase
        rank = 1
        for doc in docs:
            result.append(query_data['queryId'] + ' ' + 'Q0' + ' ' + str(doc['id']) + ' ' + str(rank) + ' ' + str(doc['sco
            rank += 1

    return result
```

Synonyms Addition

It was also noticed that the documents (tweets) did not contain the exact same words as that of the queries and so this was impacting the scores of the documents. Thus to further improve the scores, synonyms corresponding to the query terms were added to the synonyms.txt file.

Screenshot of synonyms.txt file:

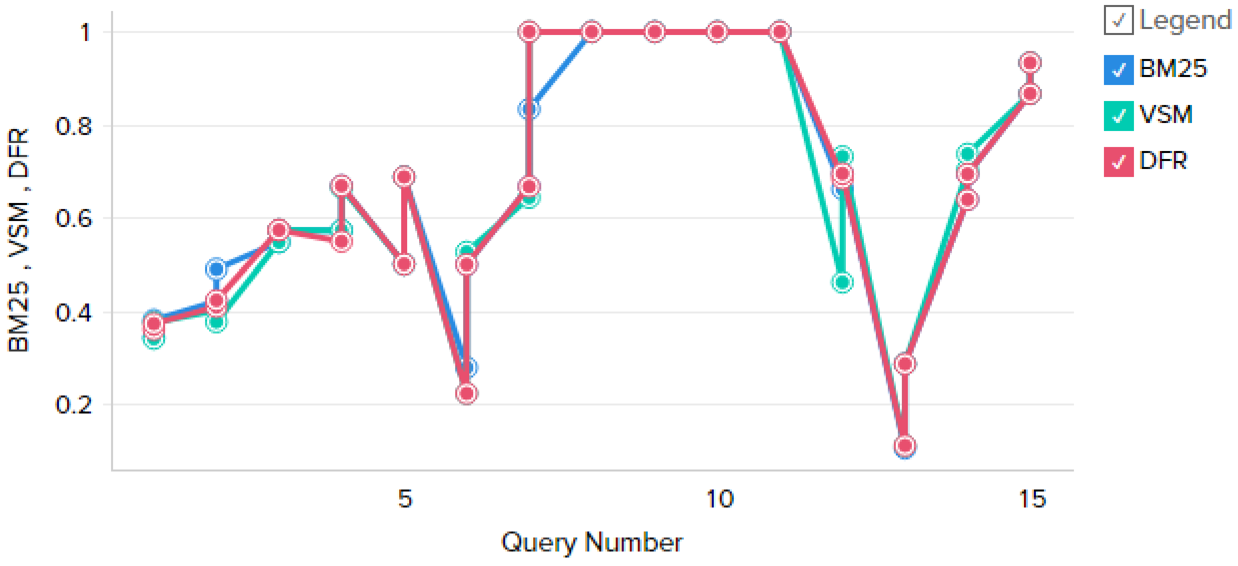


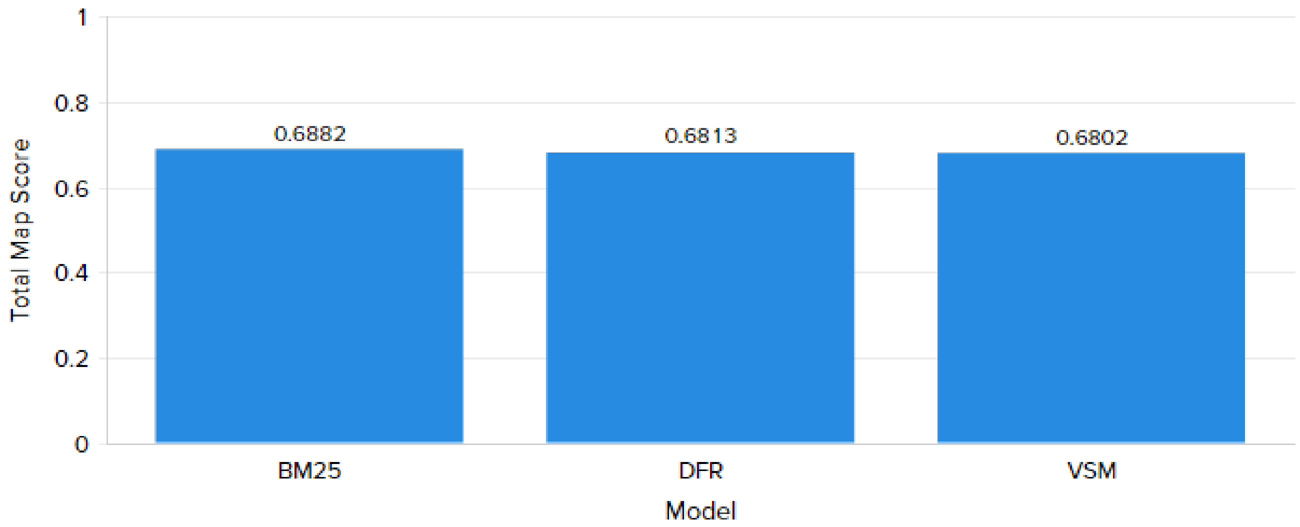
```
Rally,Assemblage,Assembly,Clambake,Convention,Convocation,Meet,Session
War,Battle,Bloodshed,Combat,Conflict,Fighting,Hostility,Strike,Struggle,Warfare,Contention,Contest,Enmity,Attack,Crisis,Crises,Bombing,Terror,Ter
Civil,Domestic,Civilian,Governmental,Political,Public,UN,Politics,Ploy
Assad,Syria,Syria
Rise,Ascent,Boost,Growth,Hike,Progress,Advancement,Intensifying
Embassy,Consulate,Ministry,Office
Hit,Attack,Attacked,Blow,Clash,Impact,Shock
Grenades,Grenade,Explosive,Explode,Missile,Firework,Weapon,Ammunition
Domestic,Internal,National,Native,Municipal
Document,Archive,Certificate,Diary,Evidence,Record,Paper,Testimony,Language,Page
Refugees,Aliens,Emigrants,Foreigners,Expatriates,Renegades,Refugee,Alien,Emigrant,Foreigner,Expatriate,Renegade,Immigrants,Immigrations,Immigrate
Accept,Get,Obtaintake,Welcome,Acquire,Gain,Secure,Relief,Aid,Help,Camps,Camp,Humanitarian,Fund-raiser,Crowdfunding
Airbnb,Kickstarter,Startup,Instacart,Tech,Technology,Companies,Company,Startups,Philanthropy,Starbucks,Firms
U.S.,USA,US,United states of america
```

Post Implementation Analysis:

With the above improvements, the following are the map values obtained:

	Map Scores		
Query Number	BM25	VSM	DFR
1	0.3801	0.3753	0.3723
2	0.4892	0.3763	0.4226
3	0.5471	0.5471	0.5729
4	0.6683	0.6644	0.6683
5	0.6875	0.6875	0.6875
6	0.2778	0.2222	0.2222
7	0.6667	0.6429	0.6667
8	1	1	1
9	1	1	1
10	1	1	1
11	1	1	1
12	0.6946	0.7313	0.6946
13	0.2857	0.2857	0.2857
14	0.6931	0.7369	0.6936
15	0.9333	0.9333	0.9333
all	0.6882	0.6802	0.6813



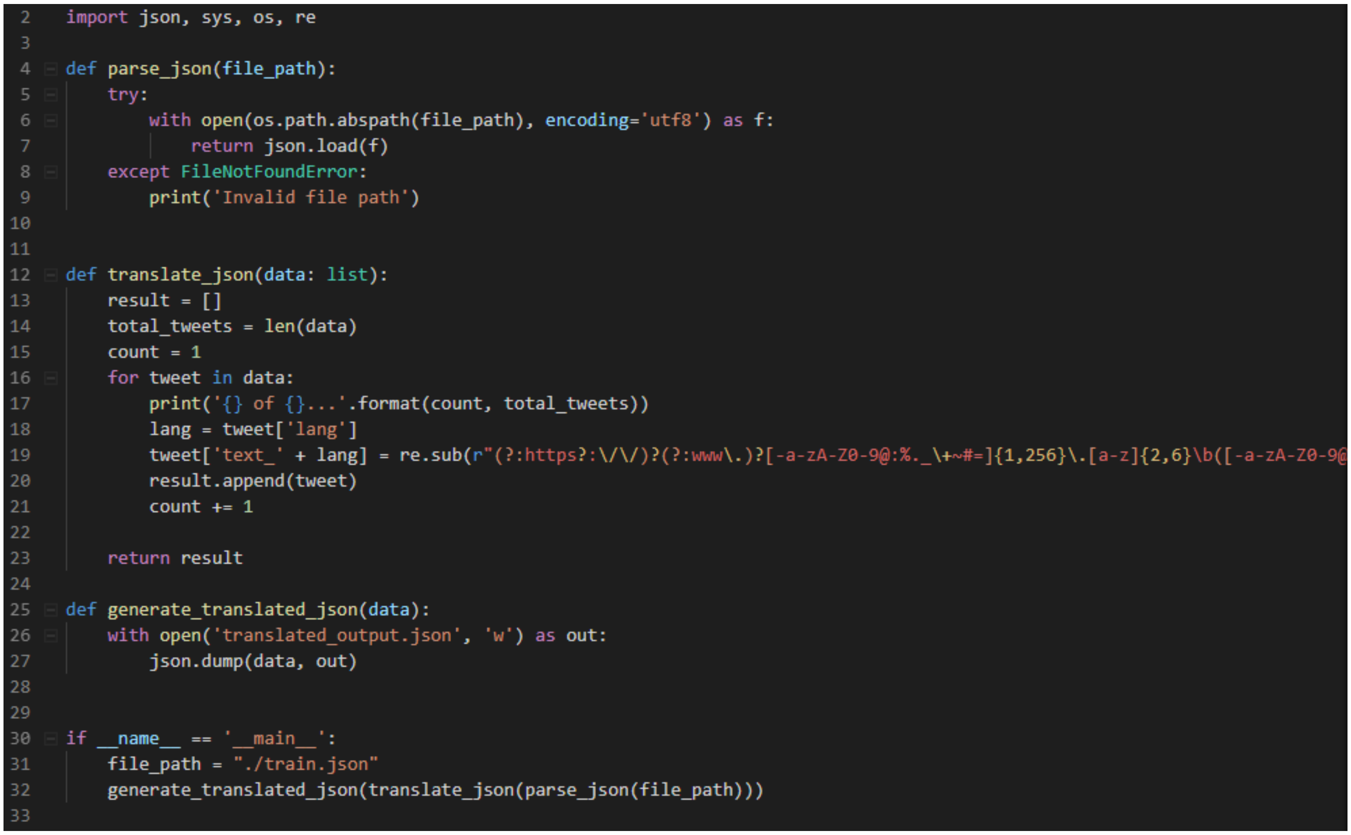


On an average, there was an improvement of 0.02 MAP scores with Query expansion.

URL Filtering:

It was also observed that the URLs in the text fields did not add relevant information thereby decreasing the scores of the documents. A preprocess script was used to remove the URLs from the text_en, text_de and text_ru fields respectively. The newly generated json file was re-indexed which improved the MAP scores.

Screenshot from the tweet_translator.py script:

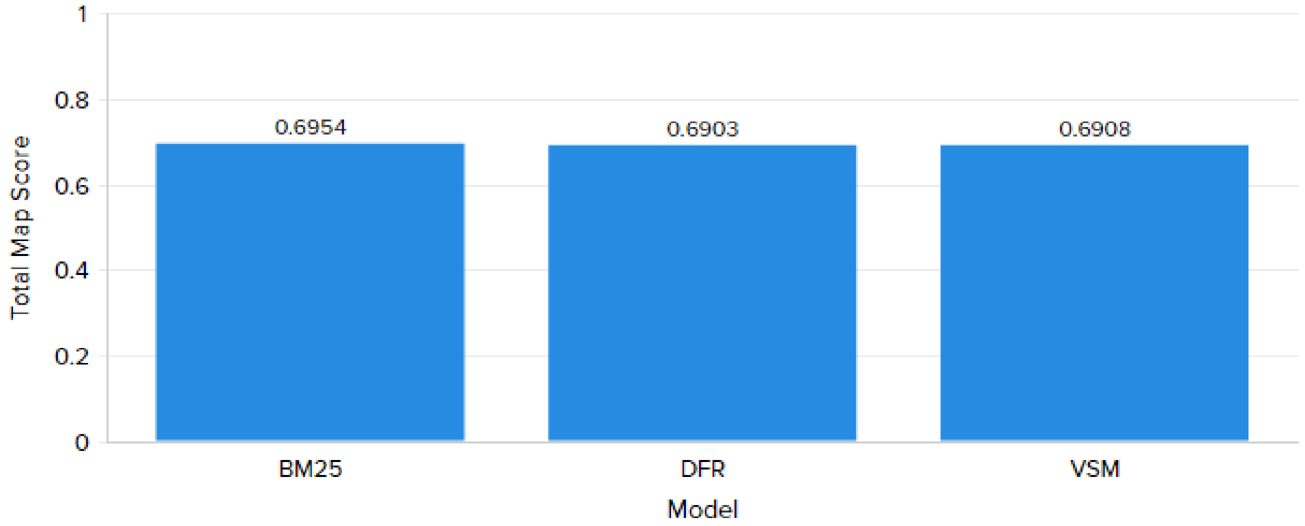


```
import json, sys, os, re

def parse_json(file_path):
    try:
        with open(os.path.abspath(file_path), encoding='utf8') as f:
            return json.load(f)
    except FileNotFoundError:
        print('Invalid file path')


def translate_json(data: list):
    result = []
    total_tweets = len(data)
    count = 1
    for tweet in data:
        print('{} of {}...'.format(count, total_tweets))
        lang = tweet['lang']
        tweet['text_' + lang] = re.sub(r"(?:https?:\/\/)?(?:www\.)?[-a-zA-Z0-9@:%._\+~#=]{1,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@
        result.append(tweet)
        count += 1

    return result

def generate_translated_json(data):
    with open('translated_output.json', 'w') as out:
        json.dump(data, out)


if __name__ == '__main__':
    file_path = "./train.json"
    generate_translated_json(translate_json(parse_json(file_path)))
```

Post Implementation Analysis:

With this change, a further improvement of 0.01 overall MAP score was achieved.

Model	Map Score
BM25	0.6954
VSM	0.6908
DFR	0.6903

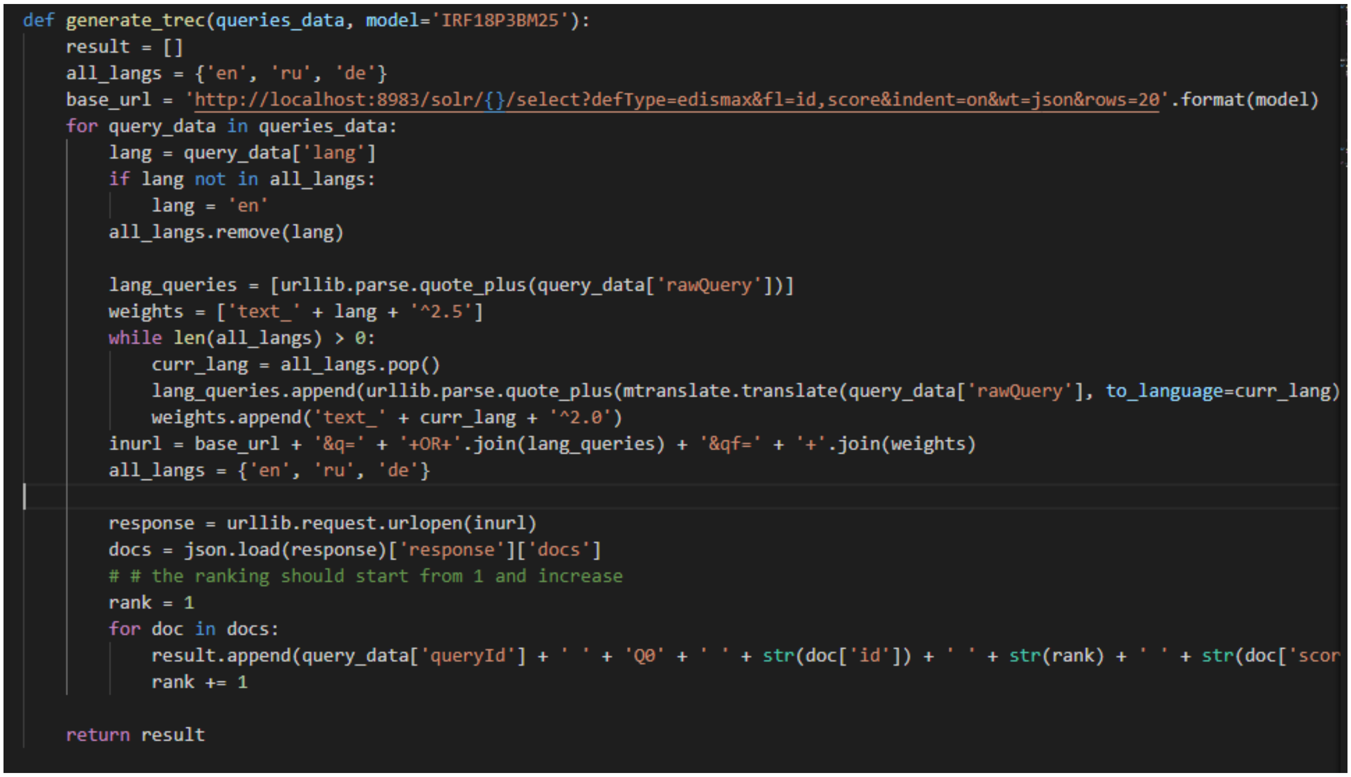


Custom Query Parser and Query Boosting:

Another idea to improve the MAP score was to give score boosting to different fields based on some criteria. In solr, this can be achieved by using edisMax query parser. The query boosting was based on the following criteria:

- Boost the scores of documents with the same language as that of the query's original language.
  For ex. Queries in English would give a boost of 2.5 to documents in English language and 2.0 for documents in other languages.
- Boost the scores of documents whose hashtags match the query terms.

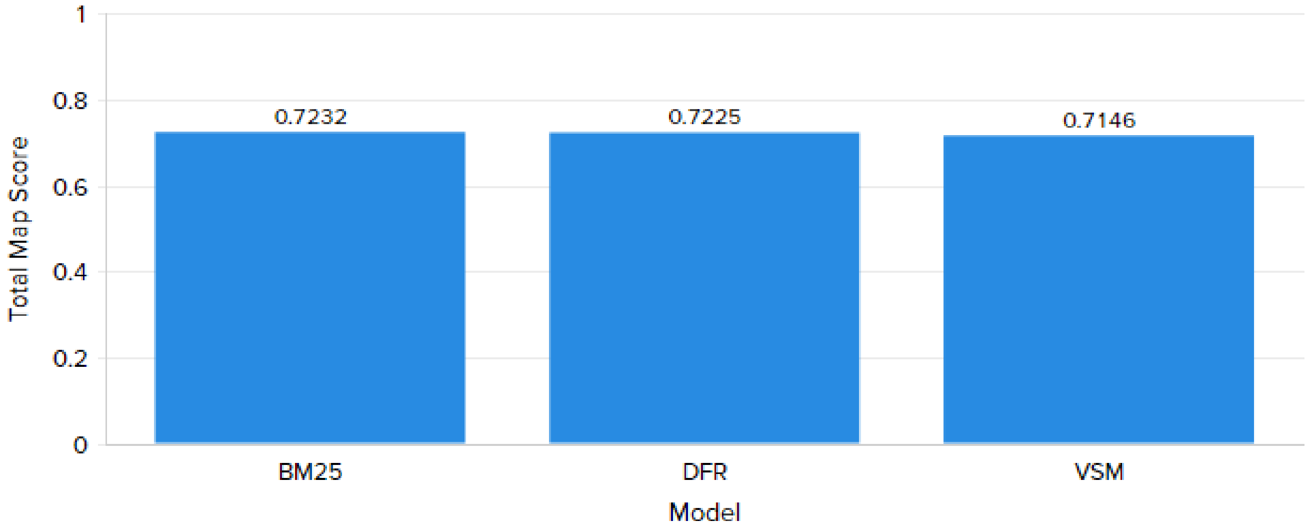
This was implemented using the qf parameter of the edisMax, which allows us to give different boost to different fields. This was included in the query building process in the json_to_trec.py script as shown below:



```
def generate_trec(queries_data, model='IRF18P3BM25'):
    result = []
    all_langs = {'en', 'ru', 'de'}
    base_url = 'http://localhost:8983/solr/{}/select?defType=edismax&fl=id,score&indent=on&wt=json&rows=20'.format(model)
    for query_data in queries_data:
        lang = query_data['lang']
        if lang not in all_langs:
            lang = 'en'
        all_langs.remove(lang)

        lang_queries = [urllib.parse.quote_plus(query_data['rawQuery'])]
        weights = ['text_' + lang + '^2.5']
        while len(all_langs) > 0:
            curr_lang = all_langs.pop()
            lang_queries.append(urllib.parse.quote_plus(mtranslate.translate(query_data['rawQuery'], to_language=curr_lang)
            weights.append('text_' + curr_lang + '^2.0')
        inurl = base_url + '&q=' + '+OR+'.join(lang_queries) + '&qf=' + '+'.join(weights)
        all_langs = {'en', 'ru', 'de'}

        response = urllib.request.urlopen(inurl)
        docs = json.load(response)['response']['docs']
        # # the ranking should start from 1 and increase
        rank = 1
        for doc in docs:
            result.append(query_data['queryId'] + ' ' + 'Q0' + ' ' + str(doc['id']) + ' ' + str(rank) + ' ' + str(doc['scor
            rank += 1

    return result
```

Post Implementation Analysis:

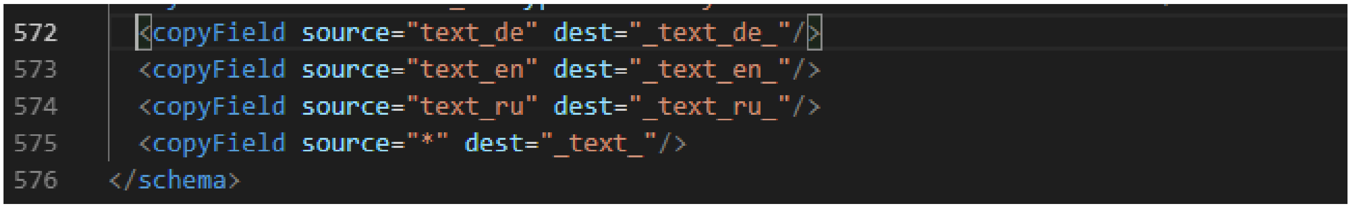
Query boosting	
Model	Map Score
BM25	0.7232
VSM	0.7146
DFR	0.7225



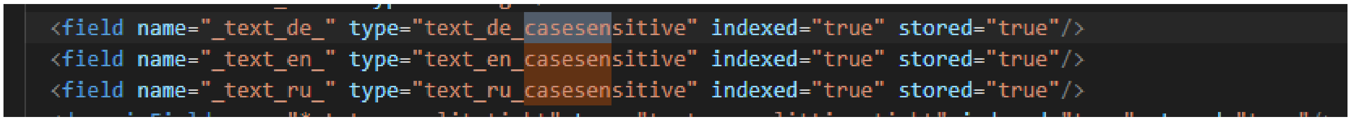
Query boosting had a significant impact on the MAP scores of the documents because it not only helped judging the relevant documents correctly but also improved the ranking of the documents.

Exact Query Matching:

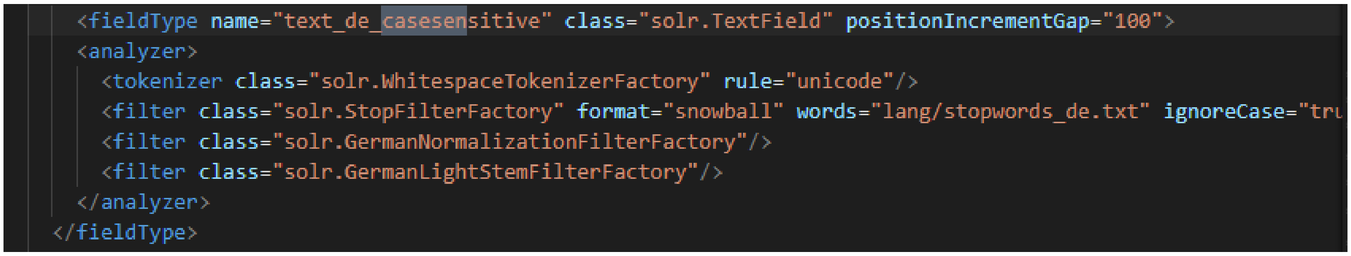
It also makes sense to rate the documents (tweets) higher with case sensitive match than the case-insensitive matches. This was achieved by creating additional field types similar to 'text_en', 'text_ru' and 'text_de' field types but without the LowerCaseFilterFactory. New copy fields were added with the new field types and documents were re-indexed. At query time, extra boost was given to documents with exact match (Case-sensitive match).



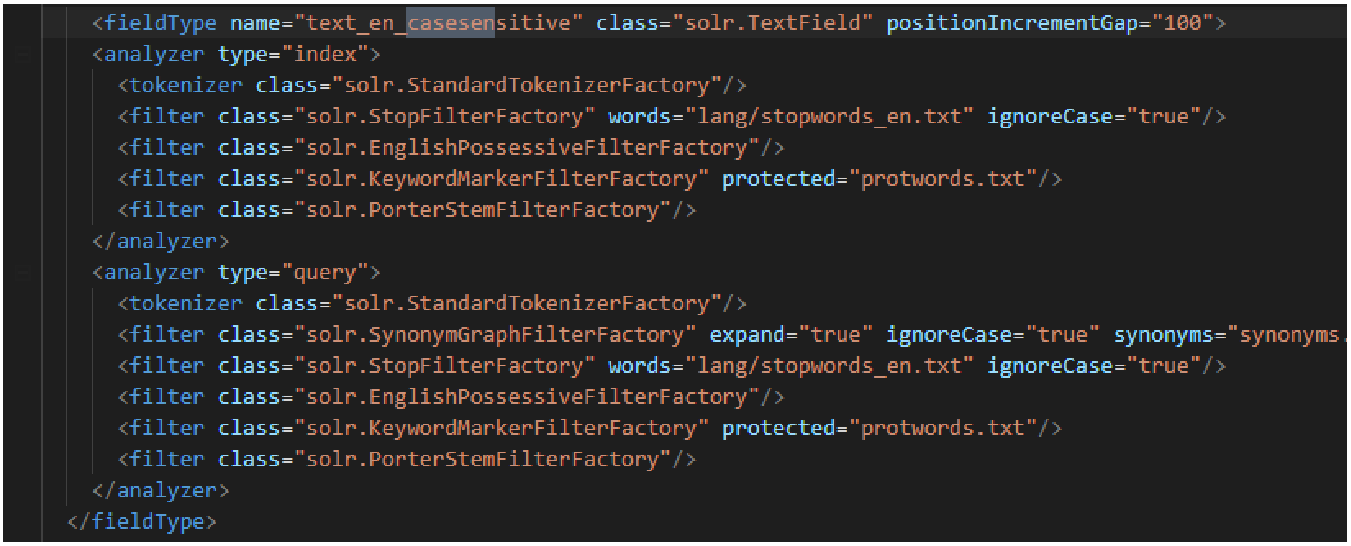
```
    <copyField source="text_de" dest="_text_de_"/>
    <copyField source="text_en" dest="_text_en_"/>
    <copyField source="text_ru" dest="_text_ru_"/>
    <copyField source="*" dest="_text_"/>
</schema>
```



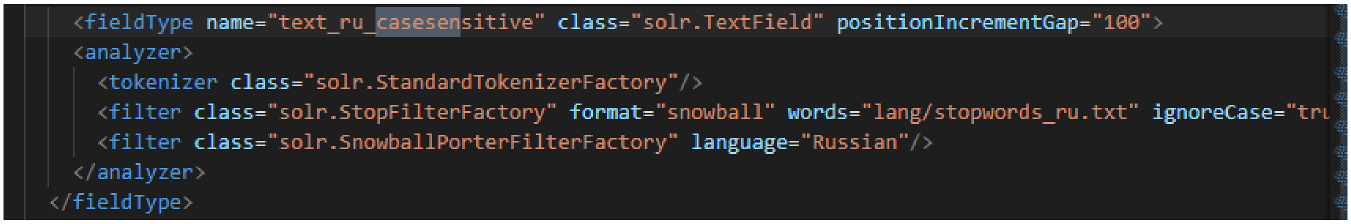
```
  <field name="_text_de_" type="text_de_casesensitive" indexed="true" stored="true"/>
  <field name="_text_en_" type="text_en_casesensitive" indexed="true" stored="true"/>
  <field name="_text_ru_" type="text_ru_casesensitive" indexed="true" stored="true"/>
```



```
  <fieldType name="text_de_casesensitive" class="solr.TextField" positionIncrementGap="100">
    <analyzer>
      <tokenizer class="solr.WhitespaceTokenizerFactory" rule="unicode"/>
      <filter class="solr.StopFilterFactory" format="snowball" words="lang/stopwords_de.txt" ignoreCase="tru
      <filter class="solr.GermanNormalizationFilterFactory"/>
      <filter class="solr.GermanLightStemFilterFactory"/>
    </analyzer>
  </fieldType>
```



```
  <fieldType name="text_en_casesensitive" class="solr.TextField" positionIncrementGap="100">
    <analyzer type="index">
      <tokenizer class="solr.StandardTokenizerFactory"/>
      <filter class="solr.StopFilterFactory" words="lang/stopwords_en.txt" ignoreCase="true"/>
      <filter class="solr.EnglishPossessiveFilterFactory"/>
      <filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt"/>
      <filter class="solr.PorterStemFilterFactory"/>
    </analyzer>
    <analyzer type="query">
      <tokenizer class="solr.StandardTokenizerFactory"/>
      <filter class="solr.SynonymGraphFilterFactory" expand="true" ignoreCase="true" synonyms="synonyms.
      <filter class="solr.StopFilterFactory" words="lang/stopwords_en.txt" ignoreCase="true"/>
      <filter class="solr.EnglishPossessiveFilterFactory"/>
      <filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt"/>
      <filter class="solr.PorterStemFilterFactory"/>
    </analyzer>
  </fieldType>
```

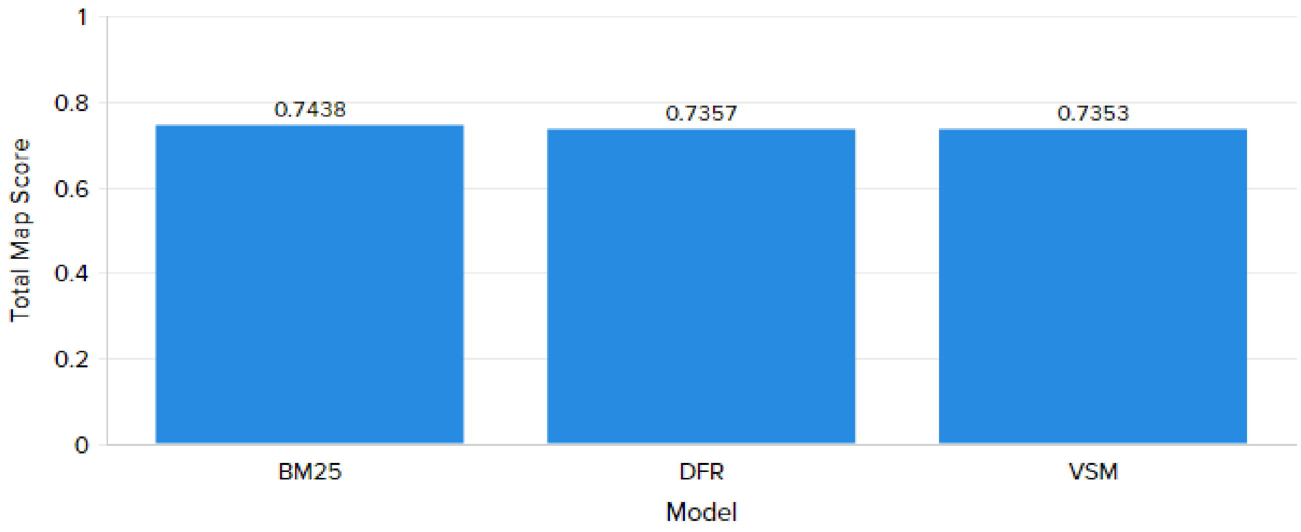


```
  <fieldType name="text_ru_casesensitive" class="solr.TextField" positionIncrementGap="100">
    <analyzer>
      <tokenizer class="solr.StandardTokenizerFactory"/>
      <filter class="solr.StopFilterFactory" format="snowball" words="lang/stopwords_ru.txt" ignoreCase="tru
      <filter class="solr.SnowballPorterFilterFactory" language="Russian"/>
    </analyzer>
  </fieldType>
```

Post Implementation Analysis:

With this enhancement, an average boost of 0.1 was observed.

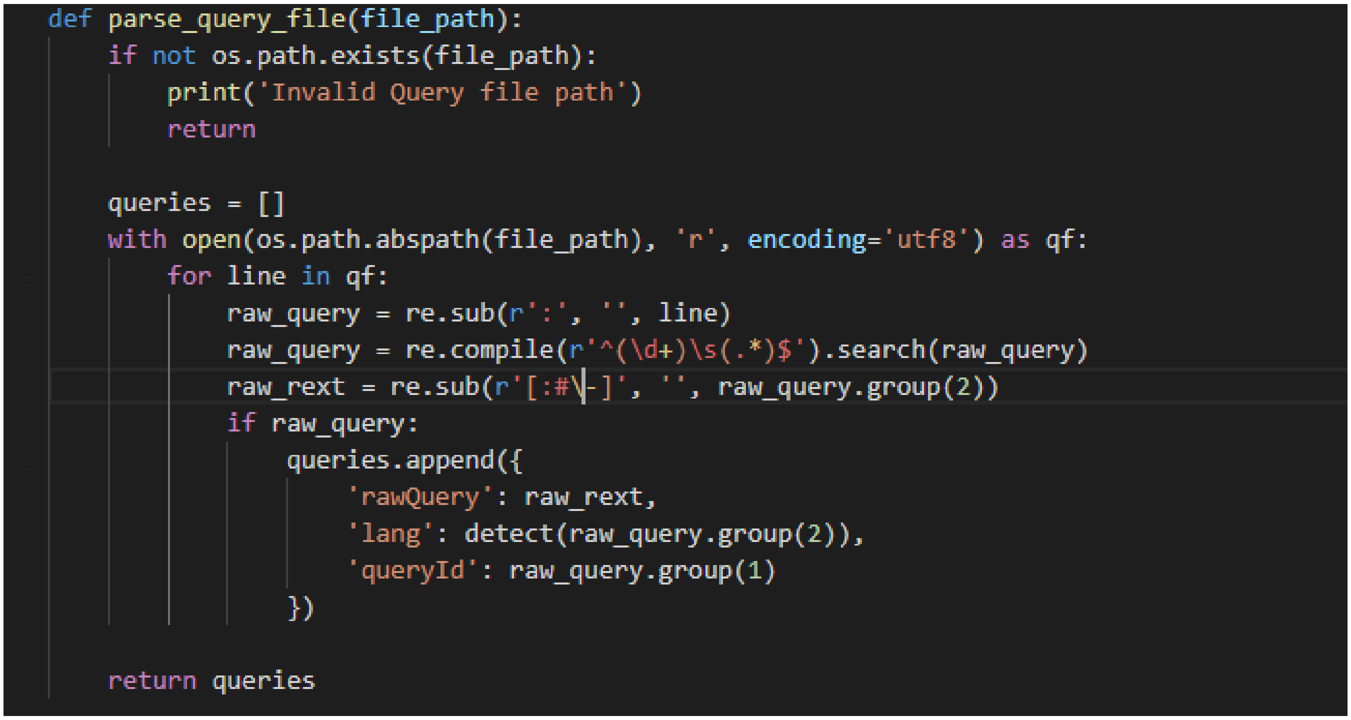
Exact Match	
Model	Map Score
BM25	0.7438
VSM	0.7353
DFR	0.7357



.

Query Refinement:

A few minor tweaks like removing hashtags, colons, hyphens etc. characters were removed during the query time to help improve the MAP score.

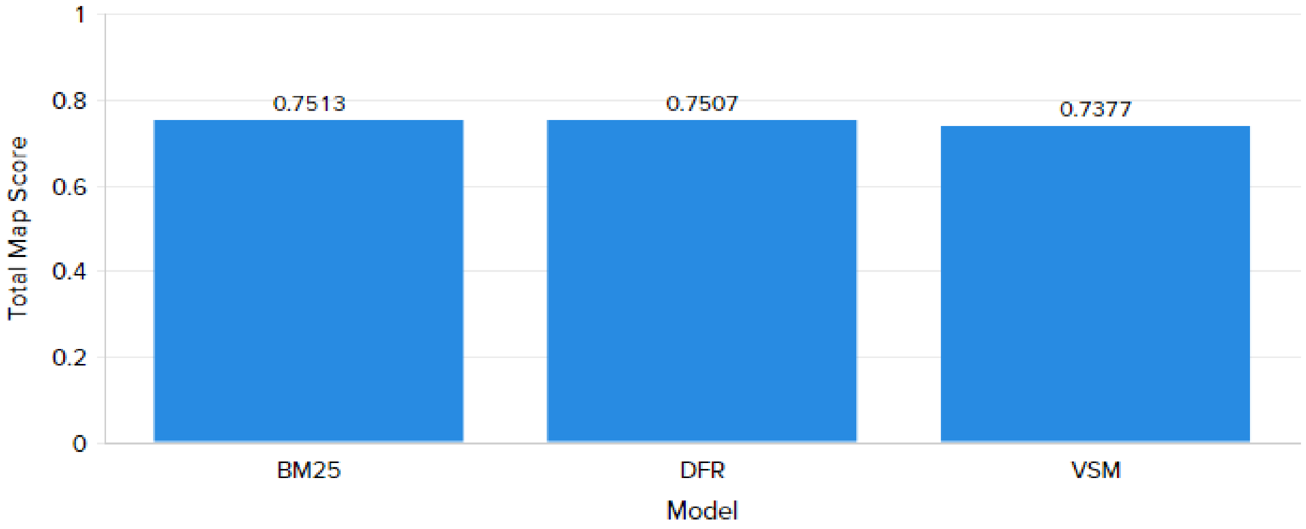


```python
def parse_query_file(file_path):
    if not os.path.exists(file_path):
        print('Invalid Query file path')
        return

    queries = []
    with open(os.path.abspath(file_path), 'r', encoding='utf8') as qf:
        for line in qf:
            raw_query = re.sub(r':', '', line)
            raw_query = re.compile(r'^(\d+)\s(.*)$').search(raw_query)
            raw_rext = re.sub(r'[:#\-]', '', raw_query.group(2))
            if raw_query:
                queries.append({
                    'rawQuery': raw_rext,
                    'lang': detect(raw_query.group(2)),
                    'queryId': raw_query.group(1)
                })

    return queries
```

Post Implementation Analysis:

On Query refinement, on average there was an improvement of 0.1 MAP score as shown below:

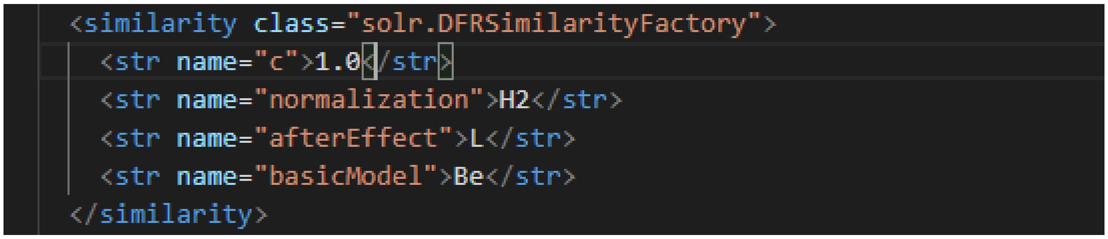
Minor tweaks	
Model	Map Score
BM25	0.7513
VSM	0.7377
DFR	0.7507



Model Specific Tweaking:

- ### DFR Parameters Tweaking

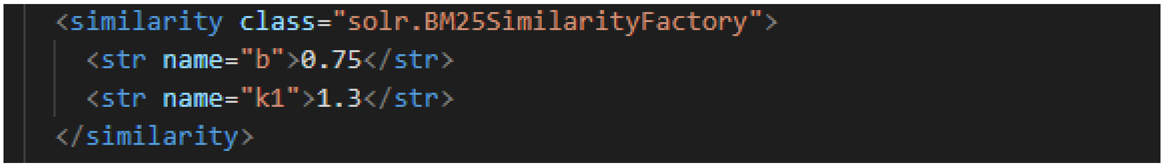
By changing the default parameters to the following parameters, an increase in MAP score was observed:



```
<similarity class="solr.DFRSimilarityFactory">
  <str name="c">1.0</str>
  <str name="normalization">H2</str>
  <str name="afterEffect">L</str>
  <str name="basicModel">Be</str>
</similarity>
```

- ### BM25 Parameters Tweaking

Similar to the above change, changing the following parameters of the BM25, MAP score is increased:

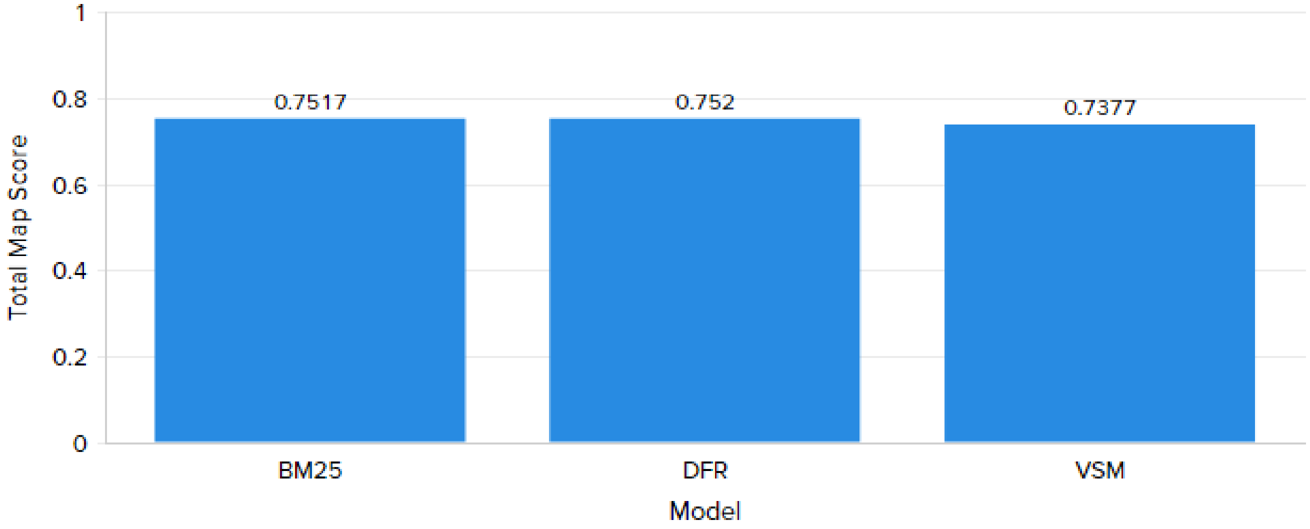


```
<similarity class="solr.BM25SimilarityFactory">
  <str name="b">0.75</str>
  <str name="k1">1.3</str>
</similarity>
```

Post Implementation Analysis:

With the above model specific tweaks, the models were performing better and on an average an increase of 0.001 was observed

DFR & BM25 Specific	
Model	Map Score
BM25	0.7517
VSM	0.7377
DFR	0.752

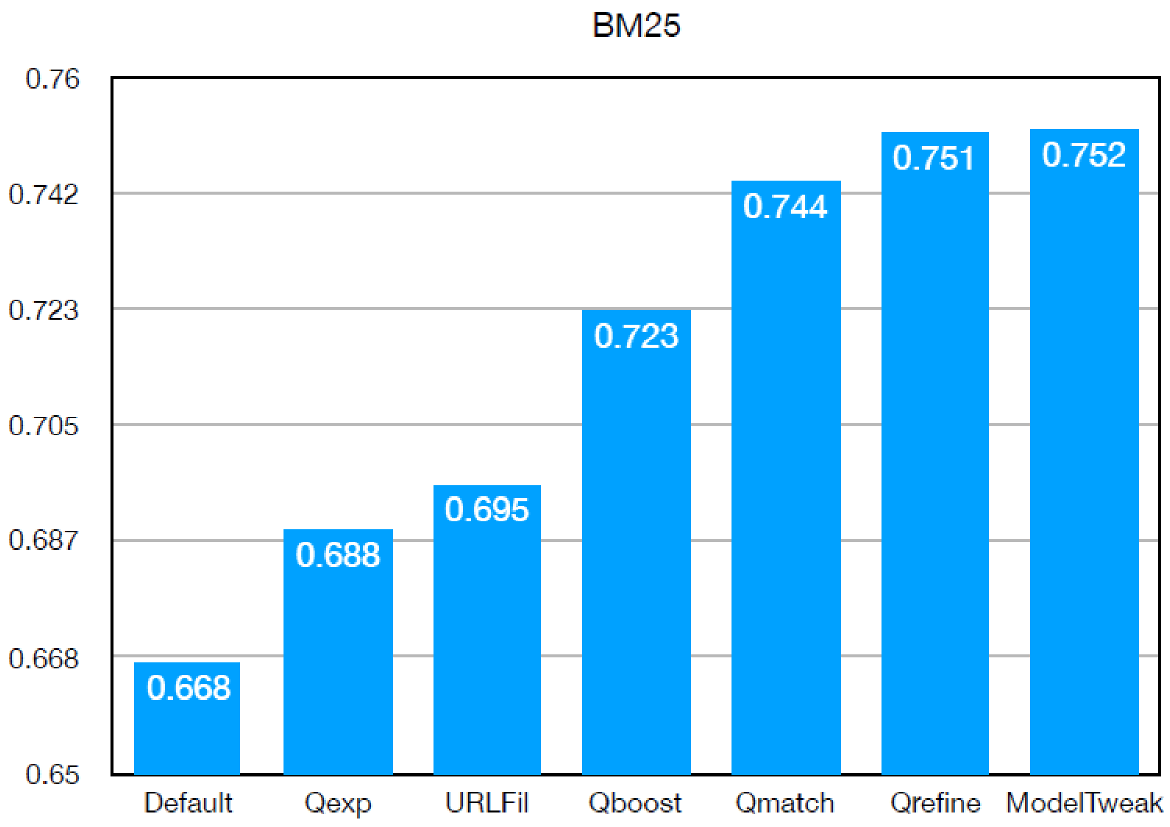


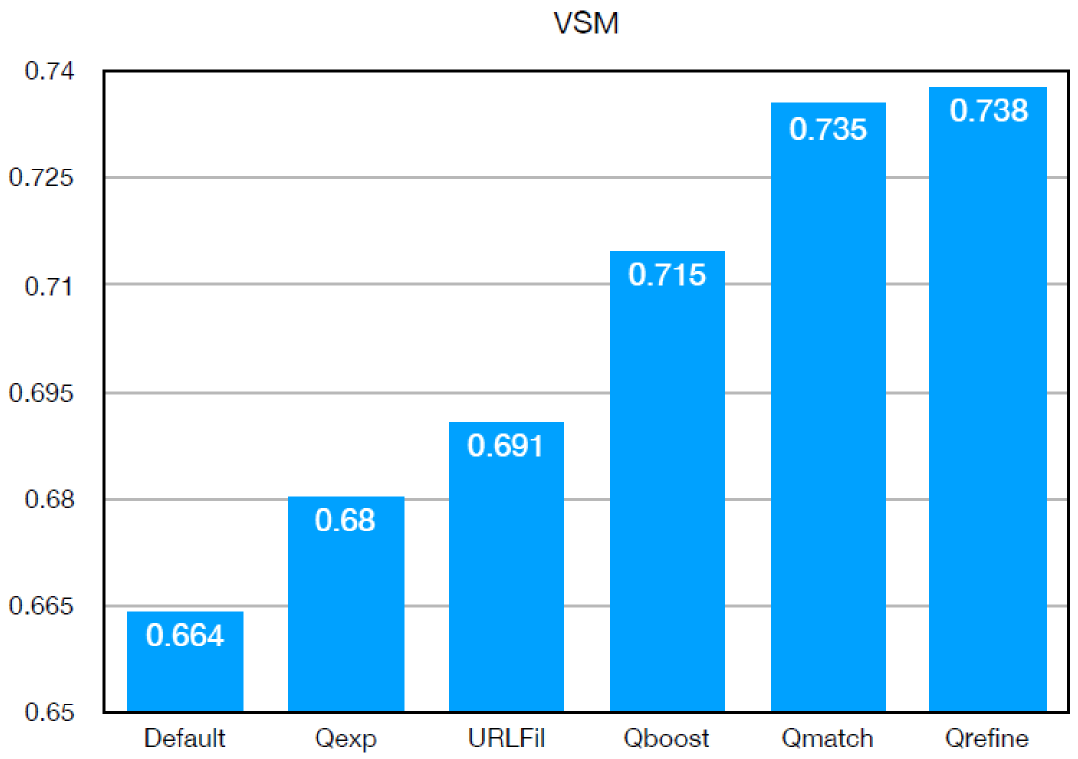
Conclusion:

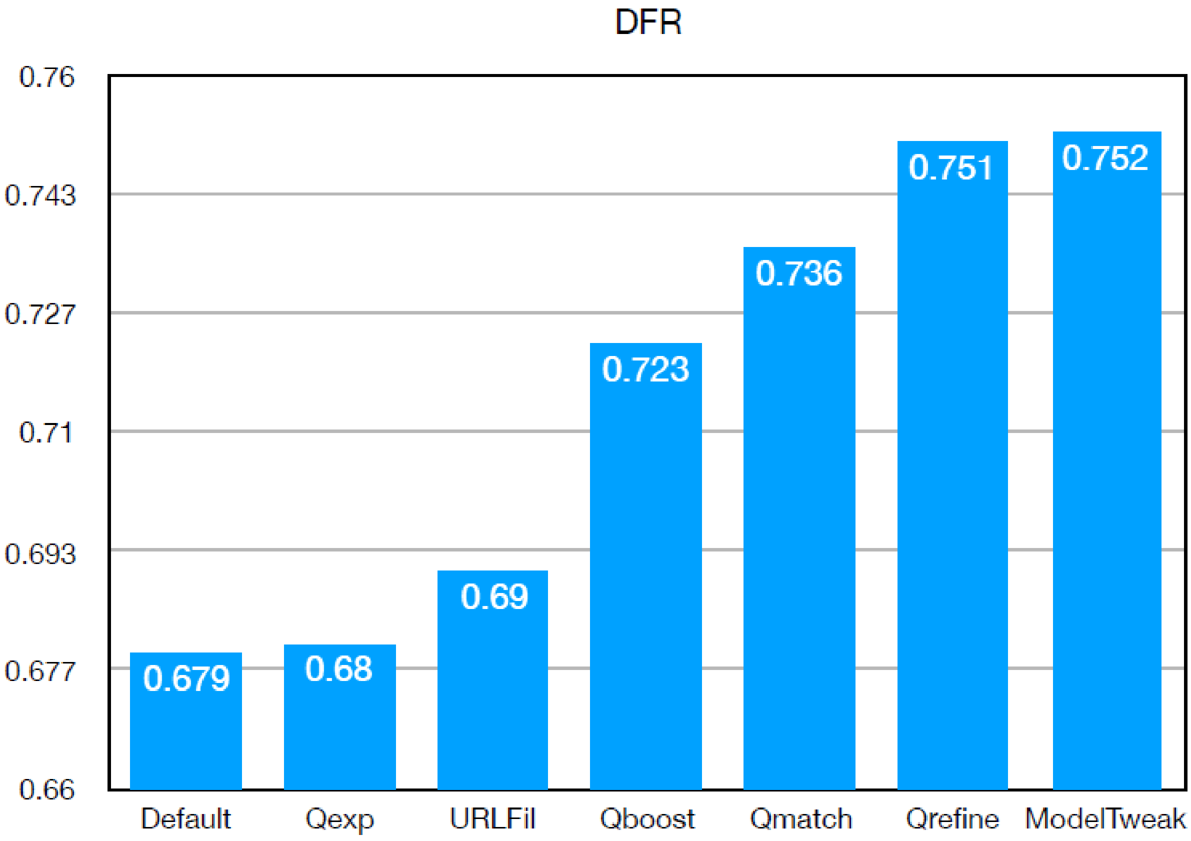
A summary of stepwise improvement statistic graphs is shown below which gives a better idea on the improvement of MAP score with each step on a per model basis and an overall comparison graph of all the models.

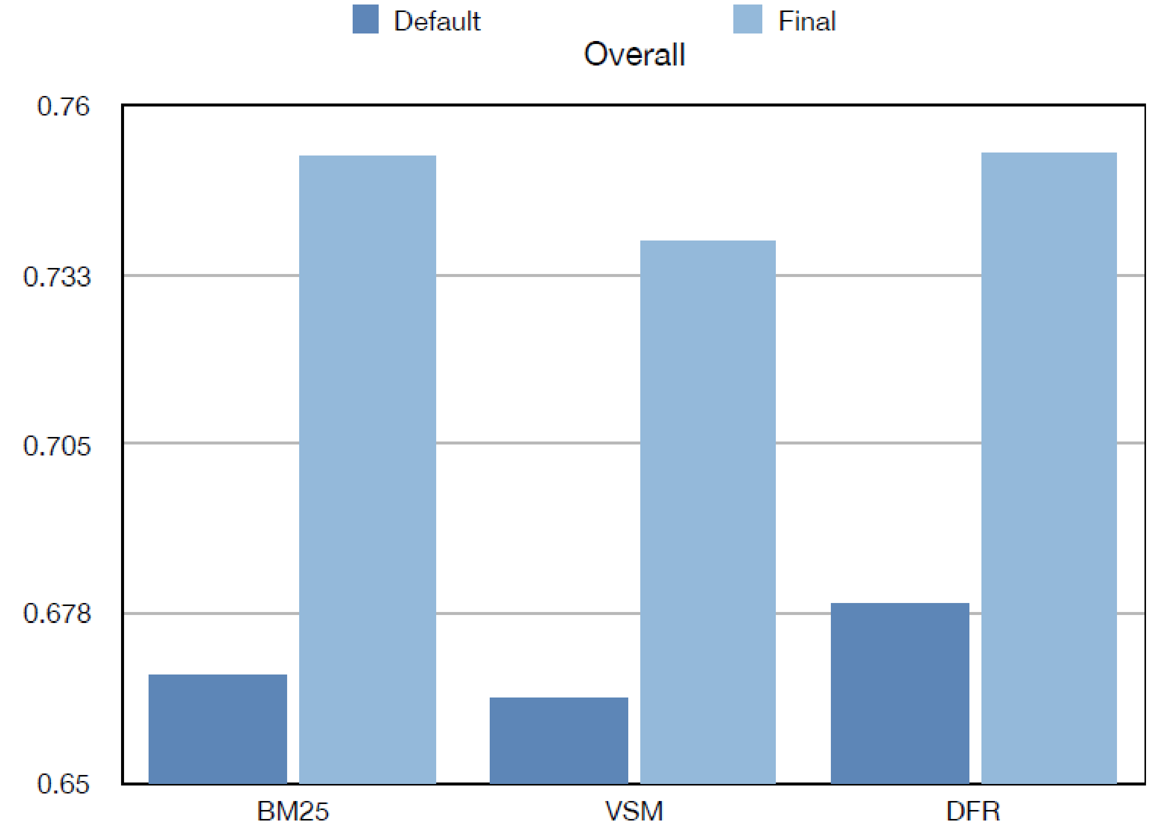
Submitted files include:

1. Modified json_to_trec.py
2. Tweets_translator.py
3. Schema files for each model
4. Synonyms.txt


BM25
0.76
0.742
0.723
0.705
0.687
0.668
0.65
0.668
0.688
0.695
0.723
0.744
0.751
0.752
Default
Qexp
URLFil
Qboost
Qmatch
Qrefine
ModelTweak


VSM
0.74
0.725
0.71
0.695
0.68
0.665
0.65
0.664
0.68
0.691
0.715
0.735
0.738
Default
Qexp
URLFil
Qboost
Qmatch
Qrefine


DFR
0.76
0.743
0.727
0.71
0.693
0.677
0.66
0.679
0.68
0.69
0.723
0.736
0.751
0.752
Default
Qexp
URLFil
Qboost
Qmatch
Qrefine
ModelTweak


Default
Final
Overall
0.76
0.733
0.705
0.678
0.65
BM25
VSM
DFR