CSE535: Information Retrieval (Fall'18)

Project 3: Evaluation of IR models

Vishal Shivaji Gawade
Department of Computer Science
University at Buffalo
Buffalo, NY,14226
vgawade@buffalo.edu
vgawade(UBIT Name)

Abstract

The goal of this project is to implement various IR models, evaluate the IR system and improve the search result based on understanding of the models, the implementation and the evaluation.

1. Implementing IR models

We have given three IR models to evaluate. First, we are going to use trec evaluation on default setting of these IR models. Together with our training queries, query results, ground truth judgements and the TREC_eval result, we gain an intuition on the performance of our IR system. We choose the measure **MAP** as main objective to improve. For our evaluation we are going to consider **top 20** results. We will be going to use below techniques to improve our models and evaluation score.

1. Tuning parameters of different models to see if it improves the MAP value.
2. Using SynonymGraphFilterFactory on fields
3. Using different tokenizer and analyzer on fields to check if it improves the results.
4. Trying query expansion by converting query into different languages, which might help to improve evaluation score.
5. Boosting the query based on different fields
6. Implementing different query parsers and check whether it helps to improve score.

2. Models with default setting

First, let's create three cores for three different models. If we just use <similarity> tag in schema.xml file and mention similarity class name as BM25/VSM/DFR, all of them have their different model parameters set to some default value. We are going to compare the results of these default setting.

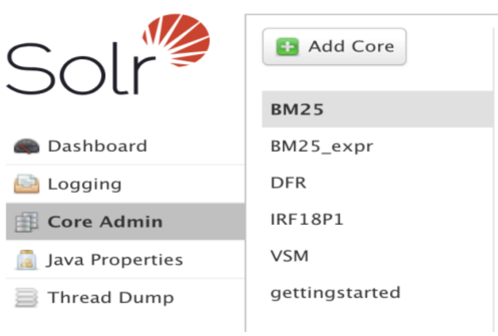

Figure 1: Cores in Solr

We have created modified query for each given query. For example, given below,

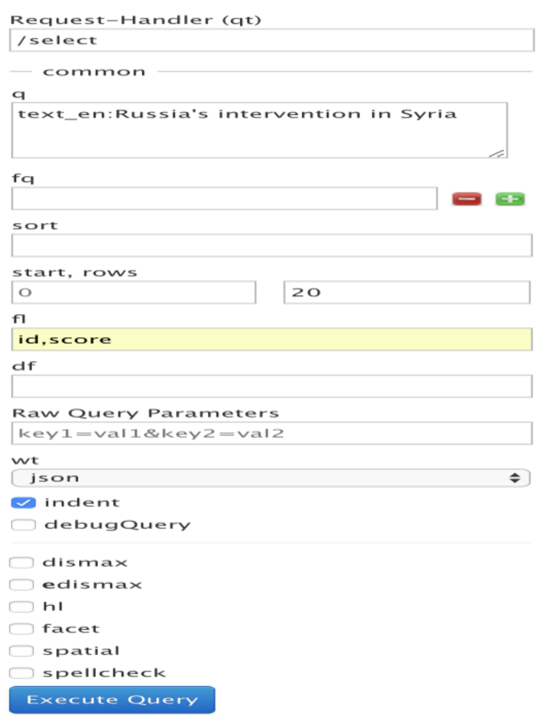

Figure 2: Querying in Solr

2.1 Vector Space Model (VSM)

Schema.xml-

```
<?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="solr.ClassicSimilarityFactory"/>
  <fieldType name="ancestor_path" class="solr.TextField">
    <analyzer type="index">
      <tokenizer class="solr.KeywordTokenizerFactory"/>
    </analyzer>
```

Figure 3: Schema.xml for default VSM

Let’s run all the queries on our model and execute json to trec python script to get top 20 results. Below is the screenshot of first query’s top 20 result.

```
001 Q0 654266183605063680 1 0.6274246 VSM
001 Q0 653941482882134016 2 0.39189237 VSM
001 Q0 653278466788487168 3 0.29482386 VSM
001 Q0 653941285045276672 4 0.2691989 VSM
001 Q0 654268898083254272 5 0.26405197 VSM
001 Q0 653278355677184000 6 0.25270617 VSM
001 Q0 653278331278913536 7 0.25270617 VSM
001 Q0 653278024876576768 8 0.23339117 VSM
001 Q0 653275762678743040 9 0.2302975 VSM
001 Q0 654268899165372416 10 0.2302975 VSM
001 Q0 654276097811566592 11 0.21469462 VSM
001 Q0 654276021919838208 12 0.21469462 VSM
001 Q0 654275700166410240 13 0.21469462 VSM
001 Q0 654270208929058816 14 0.21469462 VSM
001 Q0 653935275261980672 15 0.19803898 VSM
001 Q0 653927254888718337 16 0.19803898 VSM
001 Q0 653272979074322432 17 0.1926561 VSM
001 Q0 653270880588574725 18 0.1926561 VSM
001 Q0 653278536707506176 19 0.1833275 VSM
001 Q0 653941510665338880 20 0.18305814 VSM
```

Figure 4: json to trec result for query 001 in default VSM

Now we will store all 15 queries top 20 results in txt file.
We are going to run trec_eval command to compare our results with trec's qrel.txt

```
runid                   all     VSM
num_q                   all     15
num_ret                 all     281
num_rel                 all     225
num_rel_ret             all     114
map                     all     0.6599
gm_map                  all     0.5756
Rprec                   all     0.6662
bpref                   all     0.6688
recip_rank              all     1.0000
iprec_at_recall_0.00    all     1.0000
iprec_at_recall_0.10    all     0.9722
iprec_at_recall_0.20    all     0.9333
iprec_at_recall_0.30    all     0.9155
iprec_at_recall_0.40    all     0.7325
iprec_at_recall_0.50    all     0.6641
iprec_at_recall_0.60    all     0.5673
iprec_at_recall_0.70    all     0.5100
iprec_at_recall_0.80    all     0.3600
iprec_at_recall_0.90    all     0.3600
iprec_at_recall_1.00    all     0.2933
P_5                     all     0.8400
P_10                    all     0.6400
P_15                    all     0.4756
P_20                    all     0.3800
P_30                    all     0.2533
P_100                   all     0.0760
P_200                   all     0.0380
P_500                   all     0.0152
P_1000                  all     0.0076
```

Figure 5: trec_eval for default VSM

2.2 Divergence from Randomness Model (DFR)

Schema.xml-

```
<?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="solr.DFRSimilarityFactory">
    <str name="c">7.0</str>
    <str name="normalization">H2</str>
    <str name="afterEffect">B</str>
    <str name="basicModel">G</str>
  </similarity>
```

Figure 6: Schema.xml for default DFR

Let's run all the queries on our model and execute json to trec python script to get top 20 results. Below is the screenshot of first query's top 20 result.

```
001 Q0 654266183605063680 1 16.774292 DFR
001 Q0 653941482882134016 2 12.550833 DFR
001 Q0 653278024876576768 3 11.821917 DFR
001 Q0 654268898083254272 4 11.739952 DFR
001 Q0 653278466788487168 5 11.609203 DFR
001 Q0 653941285045276672 6 11.397218 DFR
001 Q0 653275762678743040 7 11.387619 DFR
001 Q0 654268899165372416 8 11.387619 DFR
001 Q0 654276097811566592 9 10.736669 DFR
001 Q0 654276021919838208 10 10.736669 DFR
001 Q0 654275700166410240 11 10.736669 DFR
001 Q0 654270208929058816 12 10.736669 DFR
001 Q0 653278355677184000 13 10.529008 DFR
001 Q0 653278331278913536 14 10.529008 DFR
001 Q0 653272979074322432 15 10.370556 DFR
001 Q0 653270880588574725 16 10.370556 DFR
001 Q0 653935275261980672 17 9.927791 DFR
001 Q0 653927254888718337 18 9.927791 DFR
001 Q0 653278536707506176 19 9.667354 DFR
001 Q0 654279454278197248 20 9.463333 DFR
```

Figure 7: json to trec result for query 001 in default DFR

Now we will store all 15 queries top 20 results in txt file.

We are going to run trec_eval command to compare our results with trec's qrel.txt

```
runid                   all     DFR
num_q                   all     15
num_ret                 all     281
num_rel                 all     225
num_rel_ret             all     117
map                     all     0.6572
gm_map                  all     0.5628
Rprec                   all     0.6700
bpref                   all     0.6798
recip_rank              all     1.0000
iprec_at_recall_0.00    all     1.0000
iprec_at_recall_0.10    all     0.9692
iprec_at_recall_0.20    all     0.9028
iprec_at_recall_0.30    all     0.8736
iprec_at_recall_0.40    all     0.7592
iprec_at_recall_0.50    all     0.6658
iprec_at_recall_0.60    all     0.5993
iprec_at_recall_0.70    all     0.5167
iprec_at_recall_0.80    all     0.3600
iprec_at_recall_0.90    all     0.3452
iprec_at_recall_1.00    all     0.2933
P_5                     all     0.8267
P_10                    all     0.6133
P_15                    all     0.4756
P_20                    all     0.3900
P_30                    all     0.2600
P_100                   all     0.0780
P_200                   all     0.0390
P_500                   all     0.0156
P_1000                  all     0.0078
```

Figure 8: trec_eval for default DFR

2.3 BM25

Schema.xml-

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

Figure 9: Schema.xml for default BM25

Let's run all the queries on our model and execute json to trec python script to get top 20 results. Below is the screenshot of first query's top 20 result.

```
001 Q0 654266183605063680 1 16.81431 BM25
001 Q0 653941482882134016 2 13.494686 BM25
001 Q0 653278466788487168 3 12.358061 BM25
001 Q0 653941285045276672 4 12.000957 BM25
001 Q0 654268898083254272 5 11.96763 BM25
001 Q0 653278024876576768 6 11.934417 BM25
001 Q0 653275762678743040 7 11.405806 BM25
001 Q0 654268899165372416 8 11.405806 BM25
001 Q0 653278355677184000 9 10.9446 BM25
001 Q0 653278331278913536 10 10.9446 BM25
001 Q0 654276097811566592 11 10.619978 BM25
001 Q0 654276021919838208 12 10.619978 BM25
001 Q0 654275700166410240 13 10.619978 BM25
001 Q0 654270208929058816 14 10.619978 BM25
001 Q0 653272979074322432 15 10.114389 BM25
001 Q0 653270880588574725 16 10.114389 BM25
001 Q0 653278536707506176 17 9.89045 BM25
001 Q0 653935275261980672 18 9.710419 BM25
001 Q0 653927254888718337 19 9.710419 BM25
001 Q0 654279454278197248 20 9.597064 BM25
```

Figure 10: json to trec result for query 001 in default BM25

Now we will store all 15 queries top 20 results in txt file.
We are going to run trec_eval command to compare our results with trec's qrel.txt

runid	all	BM25
num_q	all	15
num_ret	all	281
num_rel	all	225
num_rel_ret	all	114
map	all	0.6557
gm_map	all	0.5631
Rprec	all	0.6525
bpref	all	0.6706
recip_rank	all	1.0000
iprec_at_recall_0.00	all	1.0000
iprec_at_recall_0.10	all	0.9644
iprec_at_recall_0.20	all	0.9333
iprec_at_recall_0.30	all	0.9067
iprec_at_recall_0.40	all	0.7568
iprec_at_recall_0.50	all	0.6597
iprec_at_recall_0.60	all	0.5836
iprec_at_recall_0.70	all	0.5159
iprec_at_recall_0.80	all	0.3600
iprec_at_recall_0.90	all	0.3452
iprec_at_recall_1.00	all	0.2933
P_5	all	0.8400
P_10	all	0.6200
P_15	all	0.4711
P_20	all	0.3800
P_30	all	0.2533
P_100	all	0.0760
P_200	all	0.0380
P_500	all	0.0152
P_1000	all	0.0076

Figure 11: trec_eval for default BM25

3. Model Improvements

We tweaked the model as mentioned in the topic 1

3.1 Model parameter tuning

A) BM25

Let’s see map value by keeping k1=2 as a constant.

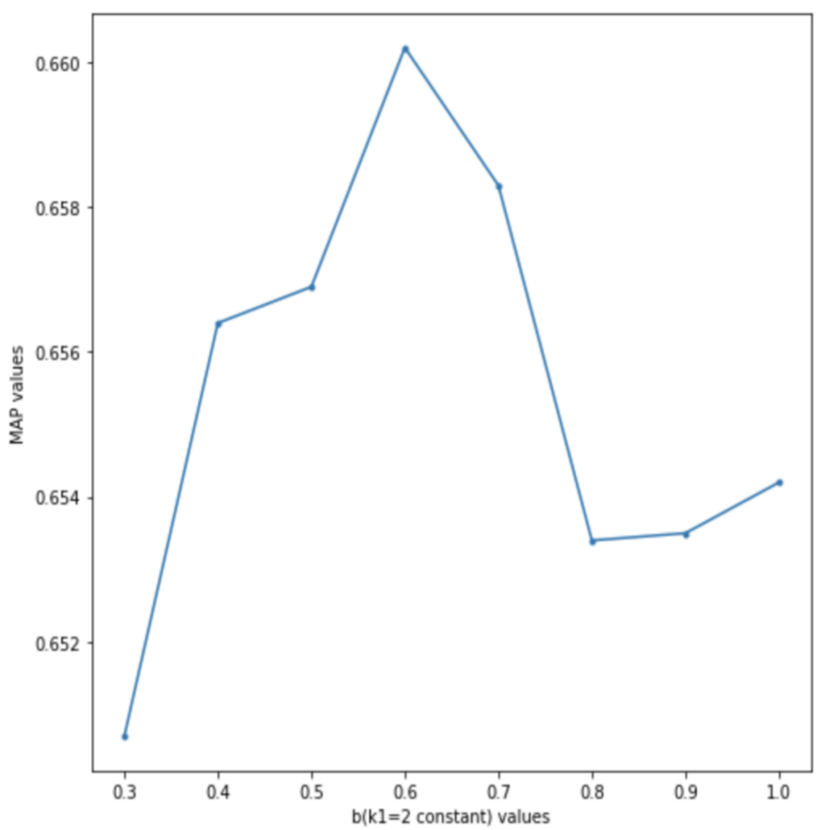

Figure 12: b vs MAP for BM25

As we can see if we keep increasing value of b from 0.3 to 1 we get max map value at b=0.6
From now onwards for other tweaking's we are going to use b=0.6 for all BM25 improvement.

Now let's keep b=0.6 constant and increase k1's value.

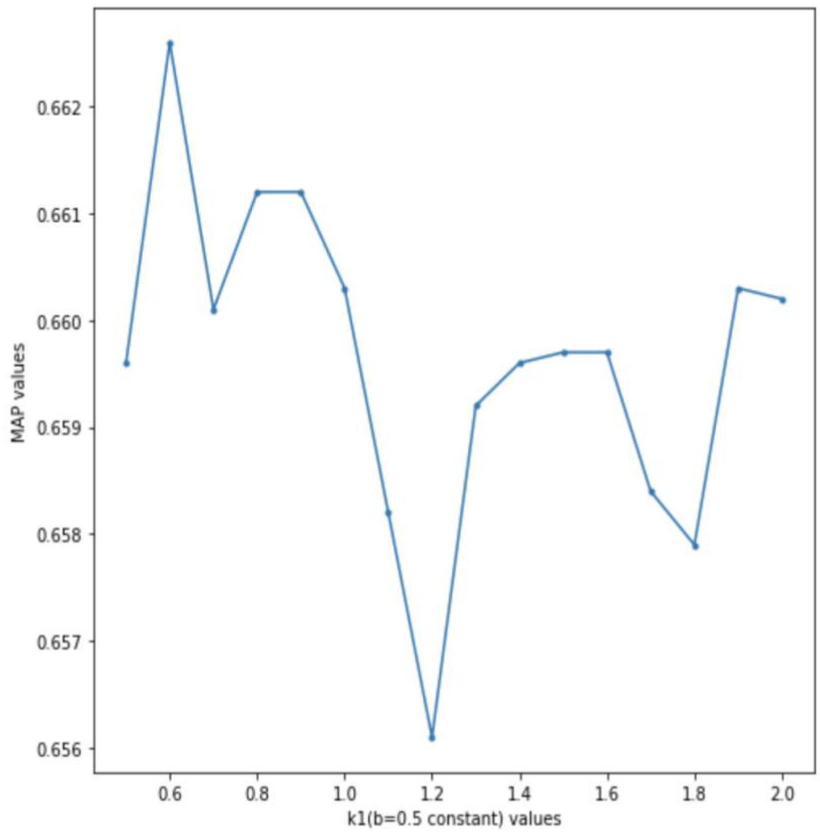

Figure 13: k1 vs Map for BM25

As we can see if we keep increasing value of k1 from 0.5 to 2 we get max map value at k1=0.6
From now onwards for other tweaking's we are going to use k1=1.2 and b=0.6 for all BM25 improvements

B) DFR

There are main for parameters to tweak. Let's keep c=7.0 (it's default value) and change other fields and let see how map value gets affected.

Table 1: Parameter tuning results for DFR

Basic Model	After Effect	Normalization	Map
G	B	H2	0.6572
I(F)	B	H2	0.6561
I(ne)	B	H2	0.6521
I(n)`	L	H2	0.6553
P	L	H2	0.6568
G	B	Z	0.6473
G	B	H3	0.6538
G	B	H1	0.6459

As we can see we are getting maximum value for default model of DFR.

Therefore, from now on we are going to use "BasicModelG" plus "Bernoulli" first normalization plus "H2" second normalization for DFR model. Changing C's value to 1.0 gives best result with above combination.

C) VSM

Using SweetSpotSimilarityFactory-

SweetSpotSimilarityFactory similarity is subclass of ClassicSimilarityFactory. If we use SweetSpotSimilarityFactory instead of ClassicSimilarityFactory we don't see any improvements in result.

Table 2: Parameter tuning results for VSM

Similarity Factory name	MAP
ClassicSimilarityFactory	0.6599
SweetSpotSimilarityFactory	0.6577

Therefore, from now onwards we are going to stick with ClassicSimilarityFactory for model improvement.

3.2 Using SynonymGraphFilterFactory on fields

I added SynonymGraphFilterFactory to schema.xml file for text_en, text_ru and text_de.

```
</fieldType>
<fieldType name="text_ru_i" class="solr.TextField" positionIncrementGap="100" multiValued="true">
  <analyzer>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.SynonymGraphFilterFactory" expand="true" ignoreCase="true" synonyms="synonyms_ru.txt"/>
    <filter class="solr.PatternReplaceFilterFactory" pattern="(_)" replace="all" replacement=" "/>
  </analyzer>
</fieldType>
<fieldType name="text_en_i" class="solr.TextField" positionIncrementGap="100" multiValued="true">
  <analyzer>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.SynonymGraphFilterFactory" expand="true" ignoreCase="true" synonyms="synonyms_en.txt"/>
    <filter class="solr.PatternReplaceFilterFactory" pattern="(_)" replace="all" replacement=" "/>
  </analyzer>
</fieldType>
<fieldType name="text_de_i" class="solr.TextField" positionIncrementGap="100" multiValued="true">
  <analyzer>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.SynonymGraphFilterFactory" expand="true" ignoreCase="true" synonyms="synonyms_de.txt"/>
    <filter class="solr.PatternReplaceFilterFactory" pattern="(_)" replace="all" replacement=" "/>
  </analyzer>
</fieldType>
<fieldType name="text_el" class="solr.TextField" positionIncrementGap="100">
```

Figure 14: Schema.xml for SynonymGraphFilterFactory

Also created customized synonyms txt files for different languages. But it did not help in increasing the map value and relevancy. Also, this synonym mapping caused to get more false positives in our results. That's why instead of it help in increasing our score value it decreased it. Therefore, I am going to use query expansion instead of SynonymGraphFilterFactory.

```
launch, fire, barrage, bombard, bung, fling, dispatch, set in motion, propel, drive,
project
Intervention, interventions, intrusion, interruption, obtrusion, invasion, attack,
Intervention, Interventionen, Intrusion, Unterbrechung, Aufdringung, Invasion, Angriff,
Вмешательство, вмешательство, вторжение, прерывание, навязчивость, вторжение, нападение
ammo, ammunition, arms, missile, missiles, weapons, bombed, bombarded, боеприпасы, оружие,
ракета, ракеты, оружие, бомбардировка, бомбардировка, Munition, Munition, Waffen, Raketen,
Raketen, Waffen, bombardiert, bombardiert
refugee, immigrant, беженец, иммигрант, Flüchtling, Einwanderer
crisis, emergency, Krise, Notfall, кризис, чрезвычайная ситуация
kills, kill, killed, murders, murder, murdered, executes, tötet, tötet, tötet, tötet,
ermordet, ermordet, vollstreckt,убивает, убивает, убивает, убивает, убивает, исполняет
vulnerable, weak, unsafe, уязвимый, слабый, небезопасный, anfällig, schwach, unsicher
campaigns, movement, Kampagnen, Bewegung, кампании, движение, fight, movement, operation
delegation, delegate, representative, representatives, ambassador, делегация, делегат,
представитель, представители, посол, Delegation, Delegierter, Vertreter, Vertreter,
Botschafter
died, die, dies, death, deceased, succumbed, starb, stirbt, stirbt, stirbt, ist gestorben,
умер, умирал, умирал, умирал, умер, скончался
photojournalist, journalists, journalist, photographer, фотожурналист, журналисты,
журналист, фотограф, Fotojournalist, Journalist, Journalist, Fotograf
umfragen, survey
terrorist, militant, террорист, боевик
Russian, Russia, Putin, Россия, Россия, Путин, Russisch, Russland
German, Germany, Merkel, Deutsch, Deutschland, Германия, Германия
Palestine, palestinian, Палестина, палестинец, Palästina, Palästinenser
European, European, European Union, EU, Europäische, Europäische, Europäische Union, EU,
Европейский, Европейский, Европейский Союз, ЕС
US, USA, America, American, Trump, США, США, Америка, Америка, Трамп, Amerika, Amerikaner
```

Figure 15: synonyms_en.txt snippet

I tried NGramTokenizerFactory tokenizer for Russian and German tweets but it did not perform better than our previous implementation. Also, I tried UAX29URLEmailTokenizerFactory for text_en, text_ru, text_de fields but it did not help.

3.3 Adding tokenizer and analyzer and query expansion

Let's add type of fields text_en, text_ru and text_de as shown in below schema.xml snippet.
We are using standard field types text_en, text_ru and text_de present in schema.xml without any modification.

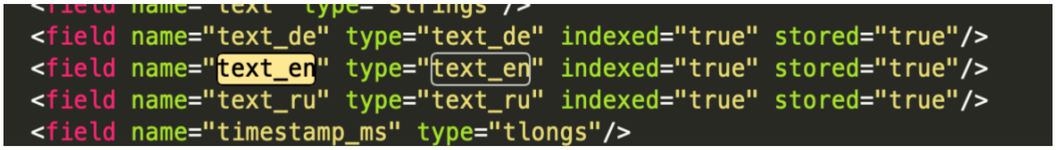

Figure 16: Schema.xml snippet

We expanded query by translating each query in German, English and Russian and then performing query operation. For example, let's look at below expansion of query 001
<u>Query</u>- 001 Russia's intervention in Syria

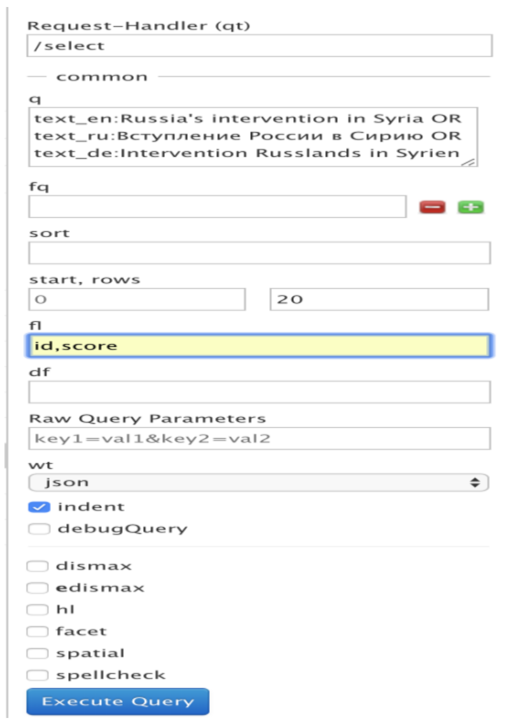

Figure 17: Query expansion in Solr

<u>Expanded query-</u>
http://localhost:8983/solr/BM25/select?q=Russia%27s%5C%20intervention%5C%20in%5C%20Syria%20OR%20Intervention%5C%20Russlands%5C%20in%5C%20Syrien%20OR%20%D0%92%D1%81%D1%82%D1%83%D0%BF%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%5C%20%D0%A0%D0%BE%D1%81%D1%81%D0%B8%D0%B8%5C%20%D0%B2%5C%20%D0%A1%D0%B8%D1%80%D0%B8%D1%8E&fl=id%2Cscore&wt=json&indent=true&rows=20

In above link just replace BM25 by VSM/DFR for those models
Now we will see how each model get affected by this change,

Table 3: Model comparison after query expansion

Model	Total number of relevant docs	Number of relevant docs retrieved		MAP	
		Before this step	After this step	Before this step	After this step
BM25	225	114	115	0.6557	0.6661

DFR	225	117	119	0.6572	0.6636
VSM	225	114	117	0.6599	0.6711

As we can see in each model, number of retrieved relevant tweets are more than before applying tokenizer, analyzer and query expansion. Also map value also gets increased because of number of relevant tweets. This advanced query processing helped to increase map value because previously we are just searching tweet in their respective language. But now we expanded our scope to tweets which are present in other languages also. Tokenizer and analyzer helped to remove stop words and stemmed the query.
If we compare all the models from above table we see that VSM model took most of the advantage of this step because of its tf-idf evaluation.

3.4 Boosting the query based on different fields

Previous Expanded query 001 -
http://localhost:8983/solr/BM25/select?q=Russia%27s%5C%20intervention%5C%20in%5C%20Syria%20OR%20Intervention%5C%20Russlands%5C%20in%5C%20Syrien%20OR%20%D0%92%D1%81%D1%82%D1%83%D0%BF%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%5C%20%D0%A0%D0%BE%D1%81%D1%81%D0%B8%D0%B8%5C%20%D0%B2%5C%20%D0%A1%D0%B8%D1%80%D0%B8%D1%8E&fl=id%2Cscore&wt=json&indent=true&rows=20

After adding term boost for 001 query-
http://localhost:8983/solr/DFR/select?q=Russia^1.2%27s%5C%20intervention^0.7%5C%20in%5C%20Syria%20OR%20Intervention^0.7%5C%20Russlands^1.2%5C%20in%5C%20Syrien%20OR%20%D0%92%D1%81%D1%82%D1%83%D0%BF%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5^0.7%5C%20%D0%A0%D0%BE%D1%81%D1%81%D0%B8%D0%B8^1.2%5C%20%D0%B2%5C%20%D0%A1%D0%B8%D1%80%D0%B8%D1%8E&fl=id%2Ctext_*%2Cscore&wt=json&indent=true&rows=20

In above link just replace DFR by VSM/BM25 for those models
Below table is for query 001 only.

Table 4: Parameter tuning results for query 001

Model	Total number of relevant docs in 001	Number of relevant docs retrieved		MAP value of 001	
		Before this step	After this step	Before this step	After this step
BM25	20	8	6	0.3723	0.2620
DFR	20	8	6	0.3652	0.3000
VSM	20	8	6	0. 3656	0. 3000

For query 001 after adding term boost for Russia^1.2 and Intervention^0.7 words we don't see any improvement in our model. Our map score decreases if we implement term boost. I tried adding term boost for other queries but it did not help in improving map value. Therefore, we are not adding term boost for our implementation.

3.5 Using DisMax query parser

Till now we were using Standard query parser for our queries. The DisMax query parser 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. Additional options enable users to influence the score based on rules specific to each use case independent of user input.

I used qf parameter of dismax parser which queries the specifies fields in the index on which to perform the query.
qf=text_en^2.8%20text_de^1.8%20text_ru^1.4%20tweet_hashtags

Previous query 001-
http://localhost:8983/solr/BM25/select?q=Russia%27s%5C%20intervention%5C%20in%5C%20Syria%20OR%20Intervention%5C%20Russlands%5C%20in%5C%20Syrien%20OR%20%D0%92%D1%81%D1%82%D1%83%D0%BF%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%5C%20%D0%A0%D0%BE%D1%81%D1%81%D0%B8%D0%B8%5C%20%D0%B2%5C%20%D0%A1%D0%B8%D1%80%D0%B8%D1%8E&fl=id%2Cscore&wt=json&indent=true&rows=20

After adding dismax parser and using qf parameter with boost-
http://localhost:8983/solr/BM25/select?defType=dismax&q=Russia%27s%5C%20intervention%5C%20in%5C%20Syria%20OR%20Intervention%5C%20Russlands%5C%20in%5C%20Syrien%20OR%20%D0%92%D1%81%D1%82%D1%83%D0%BF%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%5C%20%D0%A0%D0%BE%D1%81%D1%81%D0%B8%D0%B8%5C%20%D0%B2%5C%20%D0%A1%D0%B8%D1%80%D0%B8%D1%8E&qf=text_en^2.8%20text_de^1.8%20text_ru^1.4%20tweet_hashtags&fl=id%2Cscore&wt=json&indent=true&rows=20

In above link just replace BM25 by VSM/DFR for those models

Now let's see how this parser and query field boost affects our result.

Table 5: Model comparison after adding dismax parser

Model	Total number of relevant docs	Number of relevant docs retrieved		MAP	
		Before this step	After this step	Before this step	After this step
BM25	225	115	132	0. 6661	0.6910
DFR	225	119	130	0. 6636	0.6840
VSM	225	117	132	0. 6711	0.7140

As we can see number of relevant tweets got increased after applying dismax parser. We see improvement for all the models. In dismax parser it creates disjunction max queries from each term from the user input which helped us to increase map value. I tried different combination of boost. If we change the boost value like below,
qf=text_en^1.8%20text_de^1.2%20text_ru^1.4%20tweet_hashtags
It does help in increasing score, but above implemented combination gave max result. Finally, we come to conclusion, for our dataset above combination works the best.
If we compare all three models with each other we see VSM again takes the lead. So, we can say that on default setting VSM performed best after applying dismax parser. Also gain in increment of map value for VSM is more compare to other two models since VSM takes advantage of tf-idf evaluation while scoring.BM25 is probabilistic model, its probabilistic approach did help when we apply dismax parser but map score improvement is not significant as of VSM model. Also, in DFR model term weights are computed by measuring the divergence between a term distribution produced by a random process and the actual term distribution, which helped in increasing map value but VSM have an advantage of tf-idf scoring for our implementation.

4.Model's result comparison

Table 6: Model comparison after improvements

Model name	MAP (Default setting)	MAP (After tuning)	% increase in MAP
BM25	0.6572	0.6910	5.14 %
DFR	0.6557	0.6840	4.32 %
VSM	0.6599	0.7140	8.20 %

5.Conclusion

For given dataset **VSM** performs better than other two models when we implement above improvements in Solr and query expansion for respective models. VSM increased map value by 8.2 %.
We learnt how different IR models work and how they differ from each other in calculating relevancy of tweets(documents). We understood how Solr works and indexes the data. We used different techniques like model tuning, query parsers, query expansion and checked how it affects the relevancy score of tweets (documents). We understood how different tokenizers, analyzers work and affect the query and its results. We also learnt how to tune our model for multilingual search support with optimum performance.

References

[1] https://lucene.apache.org/solr/4_0_0/solr-core/org/apache/solr/schema/SimilarityFactory.html
[2 https://wiki.apache.org/solr/SolrRelevancyFAQ#How_can_I_make_exact-%20case_matches_score_higher