CSE 535 PROJECT 3

Report by:

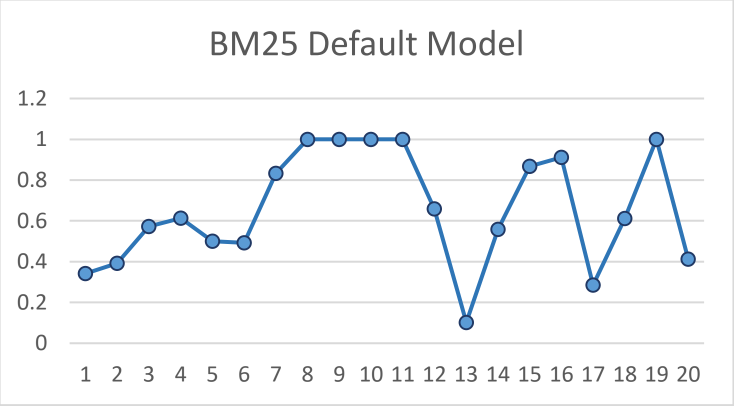
MITALI VIJAY BHIWANDE (mitalivi)
TEJASVI BALARAM SANKHE (tejasvib)

GROUP NO. 23

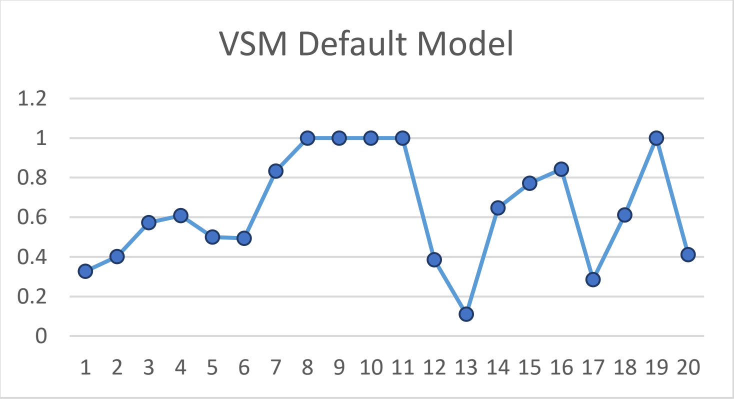
We had to implement three Information Retrieval models, evaluate the respective IR systems and improve the search result based on our understanding of the models. The models to be implemented were BM25 (Boolean Model), Vector Space Model and Divergence from Randomness (DFR) Model based on Solr

Implementing IR Models:

1. BM25:
    - For Solr version 6.2.0, Lucene combines Boolean model (BM) and Vector Space Model (VSM). Hence to implement basic BM25 model we declared the similarity class as <similarity class="solr.BM25SimilarityFactory"> and set the values of k1 and b as 1.2 and 0.75 respectively, which are the default values for BM25.
    - Run the queries in basic settings and calculate MAP.
    - Map for whole system is 0.6575

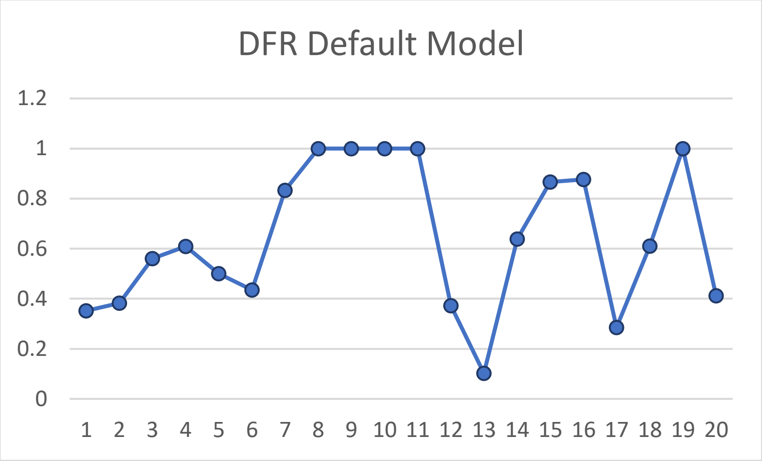


2. Vector Space Model:
    - To implement VSM in default setting we declare the similarity class as <similarity class="solr.ClassicSimilarityFactory"/>.
    - Run the queries in basic settings and calculate MAP.
    - Map for whole system is 0.6402



3. Divergence from Randomness (DFR) Model:
    - To implement DFR model we declare similarity class as
      <similarity class="solr.DFRSimilarityFactory ">
      <str name=”basicModel”>G</str>
      <str name=”afterEffect”>B</str>
      <str name=”normalization”>H2</str>
      </similarity>

- Run the queries in basic settings and calculate MAP.
- Map for whole system is 0.6419



There were no language specific stemmers applied for the above three basic implementations.

Improving IR System:

The main aim in this project was to improve the IR system performance in terms of Mean Average Precision (MAP). Precision is the fraction of the retrieved documents that are relevant, average precision is a single value obtained by averaging the precision values at each new relevant document observed i.e. highly relevant documents are more useful when appearing earlier in a search engine result list and have higher ranks.

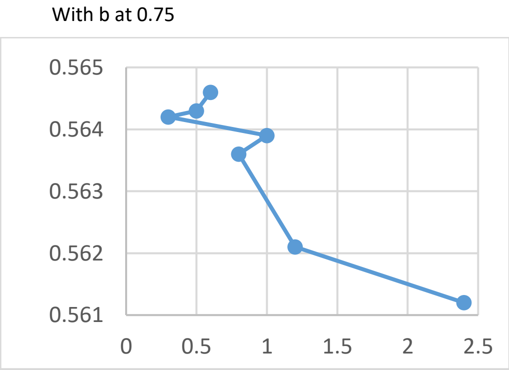
We try the following:

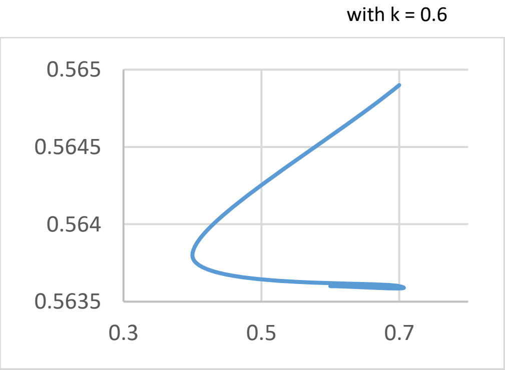
1. Tuning Parameters of Models:

BM25:

Since there are two values for BM25 model, the first one is k1, a float which controls non-linear term frequency normalization (saturation). The second, b, controls to what degree document length normalizes term frequency. We try to tune them to check if the MAP improves. We will first keep b at 0.75 which is the default value and try to tune k from 0.101 to 2 and then once we find an optimum value for K we will tune b.

We tried various values but the MAP was highest in range of k [0-1]





Hence we choose K=0.6 and b=0.7 as optimum.

DFR:

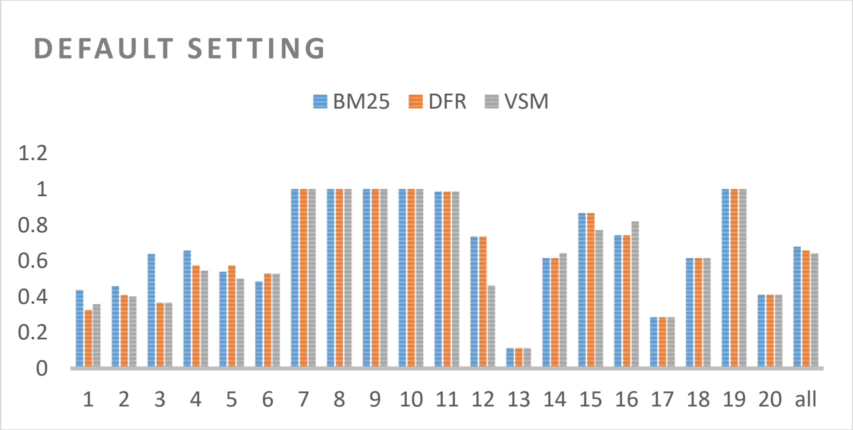
For DFR we have three parameters, basic model and normalization 1 and 2. We tried different combinations of models and normalizations as follows:

Basic	N1	N2	Constant	Map
I(ne)	L	H2	2	0.671
I(ne)	L	H2	3	0.6709
I(F)	L	Z	1	0.6677
I(ne)	L	H2	7	0.667
P	L	H2		0.6644
G	B	H2	3	0.6642
I(ne)	B	H2	500	0.666

Hence now we choose DFR configuration with basic model Inverse Expected Document Frequency model, with Laplace(L) and H2 normalization and constant c as 2.

Base Setup:

Let us now test the base setup. In this setup we will look at MAP and see how the modifications fair using different models.



For MAP some queries have very small amount of recall so they manage perfect scores. The rest fair pretty well across models. BM25 seems to perform significantly better than VSM on most queries. The algorithm itself is a mixture between the probabilistic model as well as the vector space model, and this perhaps explains why the mix of both worlds has contributed to a more robust sort of result.

2. Query Expansion using Stemmers:

Just tweaking the parameters of all models did not generate much significant improvements hence now we will try processing the query. We try query expansion using stemmers.

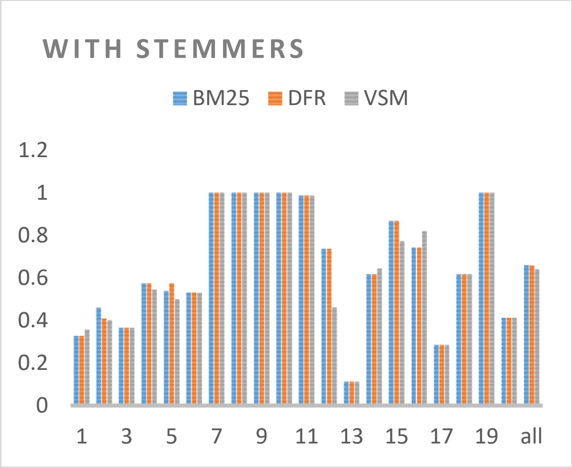
To improve the MAP of our IR system we can apply stemming filters at indexing time to generate tokens of input so that the documents can be indexed as their base words, which helps during querying.

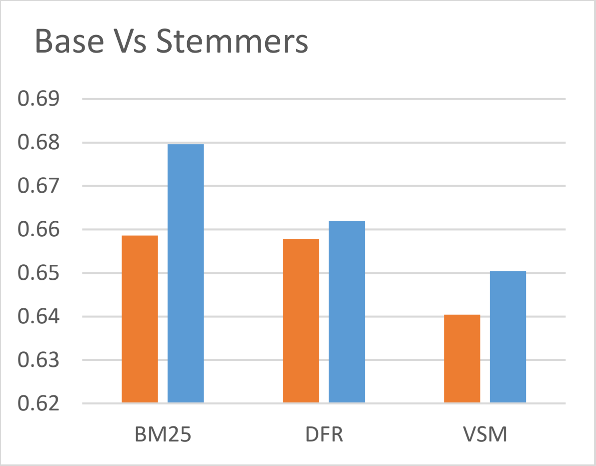
We apply the following stemmers to our fields as per the languages:

- Porter Stemmer (English) - solr.PorterStemFilterFactory
- German Light Stemmer (German) - solr.GermanLightStemFilterFactory
- Snowball Porter Stemmer (Russian) - solr.SnowballPorterStemFilterFactory

We also apply the solr.porterStemFilterFactory to the query in schema.

Results after applying Stemmers:





3. Choosing an alternative query parser:

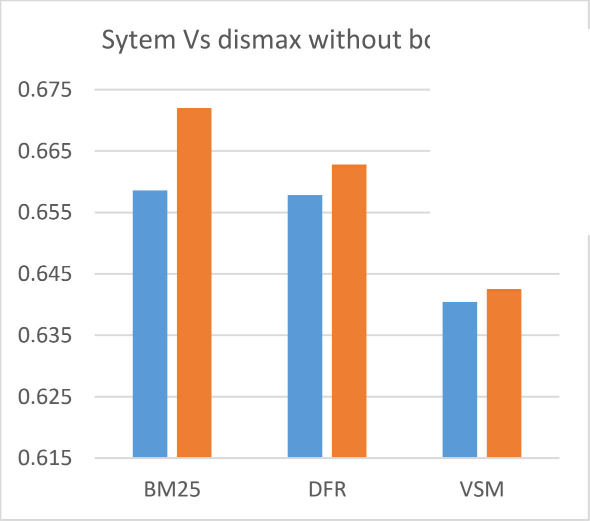
Till now we were using Solr standard query parser, however we decided to try different query parser. We chose the dismax query parser as it allows for more natural free-text queries as well as field boosting for particular fields

Query: select?&fl=id%2Cscore&wt=json&indent=true&debugQuery=true&
defType=dismax&pf=text_en%5E2+text_de%5E2+text_ru%5E2+tweet_hashtags%5E1.5&
qf=text_en+text_de+text_ru+tweet_hashtags'

Here the pf field means “phrase fields” which can be used to "boost" the score of documents in cases where all of the terms in the "q" param appear in close proximity
qf field is used to define the fields we want solr to search in. We gave all languages an equal boost of 2 and boost of 1.5 to tweet_hashtags.

The MAP improved on using dismax, hence we tried different boost combination for the fields. Map increased when we boosted tweet_hashtags field. We decided to come back later to dismax, and for now limit the dismax query to only specify the query field.

The MAP of the system till this point is:

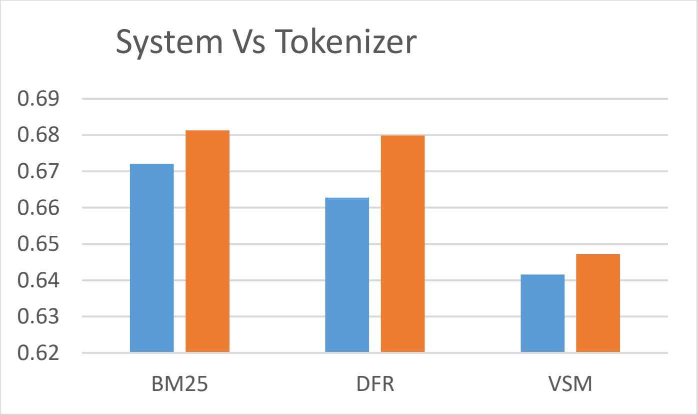


4. Trying Different Tokenizers:

Solr provides an abundance of predefined tokenizers, we decided to try out a few and check how the system performs.

First we tried the N gram tokenizer, by keeping the minLength as 3 and maxLength as 5 since we had German and Russian data in our index. Using fixed length tokens could improve the query results. The MAP did increase, but it increased significantly on using UAX29URLEmailTokenizerFactory. It could be because many top k terms were from url such as http, RT, @ etc.

We decided to keep UAX29URLEmailTokenizerFactory. The MAP results after changing from standard tokenizer:



5. Query Parser using Minimum Match:

The mm field tells us the “minimum” must “match” among the optional clauses in the q field. Specifying mm=25% indicates that at least 25% of the optional clauses must match.

Specifying mm as 25% decreased the MAP maybe as the result documents are limited to 20.

Hence we decided to not use mm parameter of dismax.

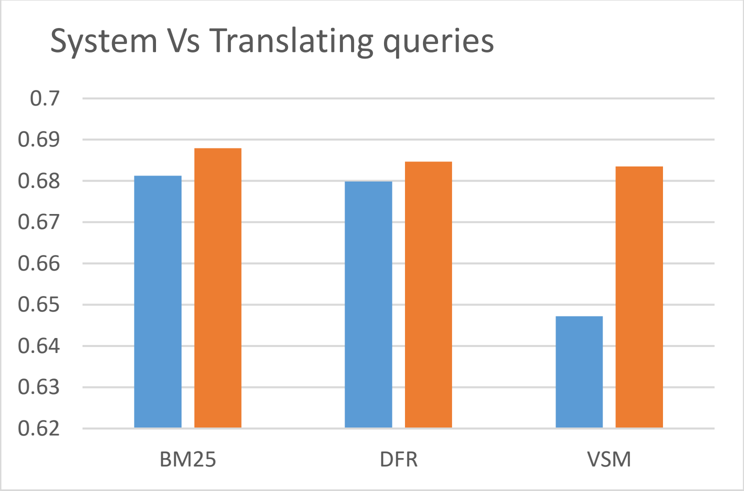
6. Translating the queries:

As the training data was cross lingual, we thought the queries to the system could contain relevant documents in another language. Hence we decided to translate the queries in all languages present in all documents i.e. English <-> Russian <-> German.

On querying the system using translated queries the MAP improved from the earlier setup.

Possibly as lot of Russian queries had relevant documents in other languages, translating the query has provided the biggest improvement in the System. The improvement is most notable for the VSM model.

MAP of system until this point:



7. Exact case Matching:

We tried to attempt and see if making exact case matches score higher would result in improved performance. To achieve this we copied the fields into fields that didn't get lowercase filtered while being tokenized in the analyser.

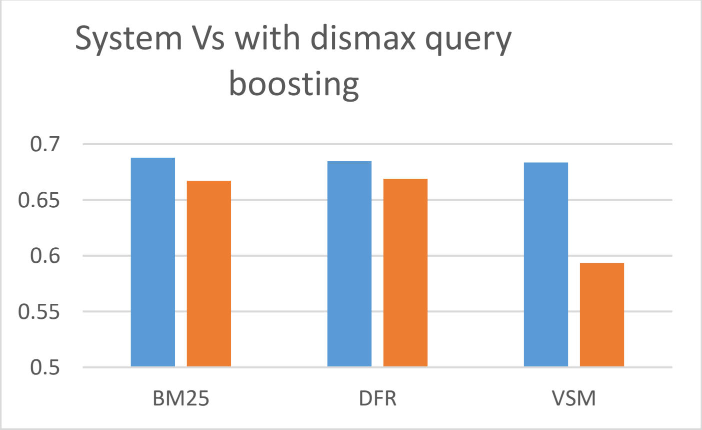
Unfortunately there was no improvement. Due to this we will not be using Exact Case matching to improve our performance as it decreases our performance overall

8. Query Boosting:

We tried query boosting after translating across languages and applying a higher boost to the original language of the query. We noticed that often times the native language of a query is favoured over translations. Since in our setup we translate the original query and query the different language fields we thought it would be of benefit to boost the native language of a query.

As per each query observation if query is in English then English docs have higher weightage over rest of the languages. If query weightage in German, German doc will have highest weightage, next highest should be English If query weightage in Russian, Russian doc will have highest weightage, next highest should be English.

German Query: qf = text_de^1.5+text_en^0.25+text_ru^0.02
Russian Query: qf = text_ru^1.5+text_en^0.25 +text_de^0.02
English Query: qf =text_en^2+text_ru^0.5+text_de^0.75



This worked when we were working with 200 rows, but query boosting reduced MAP when we were working with 20 queries. That's why we decided not to apply boost for this result.

9. Filters - Remove Duplicates

We thought it'd be interesting to see if removing duplicates has any impact on performance. To attempt this we had to fiddle with filters. The most important filter in question here is solr.RemoveDuplicatesTokenFilterFactory. Let us look at the results: It appears this change had not difference as well. This is perhaps due to there not being many duplicates in the collection and queries themselves.

10. Re-tuning the Model's Parameters:

Lot has been changed since we last tuned the parameters of our models. Hence we re-tune them to see if MAP can be further improved.

BM25:

We retune k and b

K	b	Map
0.4	0.8	0.6881
0.6	0.8	0.6879
1.4	0.8	0.6866
0.4	0.6	0.6894
0.4	0.4	0.6811

Hence we choose k=0.4 and b=0.6

DFR:

Basic	N1	N2	Constant	Map
I(ne)	L	H2	2	0.6847
I(ne)	L	H2	2.5	0.6877
I(ne)	L	H2	3.5	0.6874
I(ne)	L	H2	1.5	0.6867

We change the constant c of H2 as 2.5 from 2.

Conclusion:

At this point we have chosen the best improvements that would maximize our performance across many different metrics.

Efficient Configurations:

1. **BM25**

a) Using URLTokenizer instead of standard tokenizer
b) Parameter Values: k1=0.4, b=0.6
c) Using dismax query parser.
d) Translating the query across languages.

2. DFR

a) Using URLTokenizer instead of standard tokenizer
b) Parameter Values: I(ne), L, H2 and constant c=2.5
c) Using dismax query parser.
d) Translating the query across languages.

3. VSM

a) Using URLTokenizer instead of standard tokenizer
b) Using dismax query parser with equal boost to all fields
c) Translating the query across languages.