EVALUATION OF IR MODELS

Project 3

Abstract

One of the most important aspect of search engine is returning relevant results out of the entire document set. Furthermore, the engine should be optimized to select the most relevant documents. Most of our effort in this project is in the direction of improving the recall and the average precision values. Designing an optimum weighted query, and tuning the respective model to improve precision is the challenge. In this document we try to optimize these values using custom designed scripts and propose an algorithm to assign weights according to the provided relevancy. We discuss in detail about the implementation and the outcome of our implementation in the document ahead.

Sushmita Sinha
UBITname: ssinha7
ssinha7@buffalo.edu

Mohammad Abuzar
UBITname: mshaikh2
mshaikh2@buffalo.edu

Table of Contents

Information Retrieval Model

The IR model heavily depends upon the type of query user inputs, for the model to provide optimum results. Usually the queries are in plain text format containing hash-tags and key terms. The user inputs the query in raw format, which if directly fed to the IR Model would yield low precision. We now discuss our strategy to build the model

Optimizing the input Queries on default model BM25

We have used Solr 6.2.1 to illustrate our experiment in this project. By default, this version of Solr implements **BM25** with default configuration to return the results. In this section we will highlight facts about getting query expansion and optimization. Our observations with respect to expanding queries and providing weights to key terms is enlisted below:

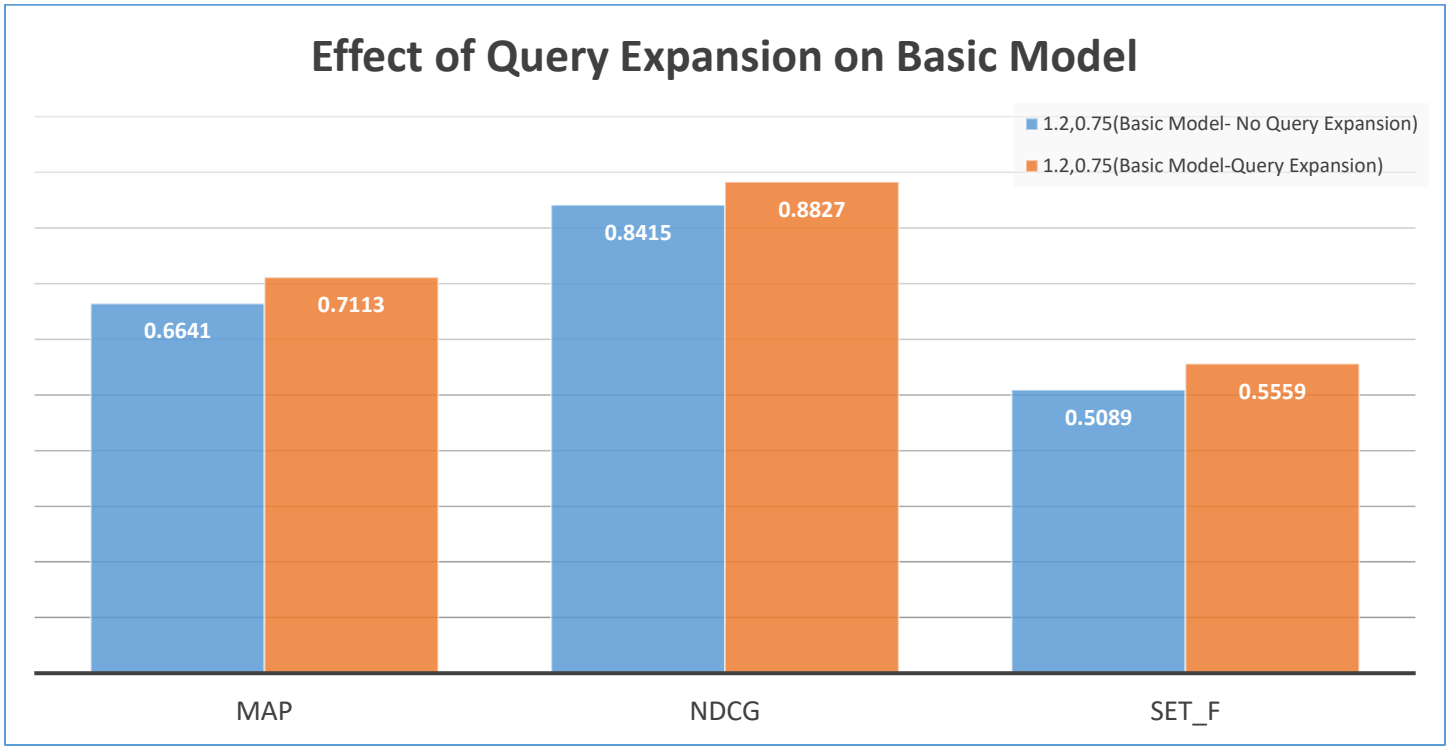

Fig. 1.1

1. Unexpanded or raw queries are free text queries; For Example: Russia's intervention in Syria; which gives equal importance to each term in the query.
2. By judgement we know that the word Russia and Syria are important and should be present in the sentence for it to be relevant. We thus introduce a proximity rule in the query statement, for only English language, which made the query look like as follows:

   text_en:(“Russia Syria”~5)
3. We also observed that some relevant documents were present in Russian and Germany and so it was now required to translate the queries in these languages and optimize them as well. This made our query look like as follows:

   text_en:(“Russia Syria”~5) OR text_de:(“Russlands Syrien”~5) OR text_ru:(“России Сирии”~5)
4. We noticed that some important words of the queries can be found in tweet_hashtag and they are more relevant in grouping the valid documents together
5. This made it obvious that we are missing the word intervention and its synonyms, which led us to include this word with higher importance (Intervention^2).
6. We also added the synonyms of the word “intervention”, say, interference, arbitration, etc. to get more relevant documents. Post tweaking, the query to displays as follows:

text_en:(“Russia Syria”~5 OR Russia's intervention^2 in Syria) OR text_de:(“Russlands Syrien”~5 OR Russlands Intervention^2 in Syrien)^1 OR text_ru:(“России Сирии”~5 OR Вмешательство России в Сирии)

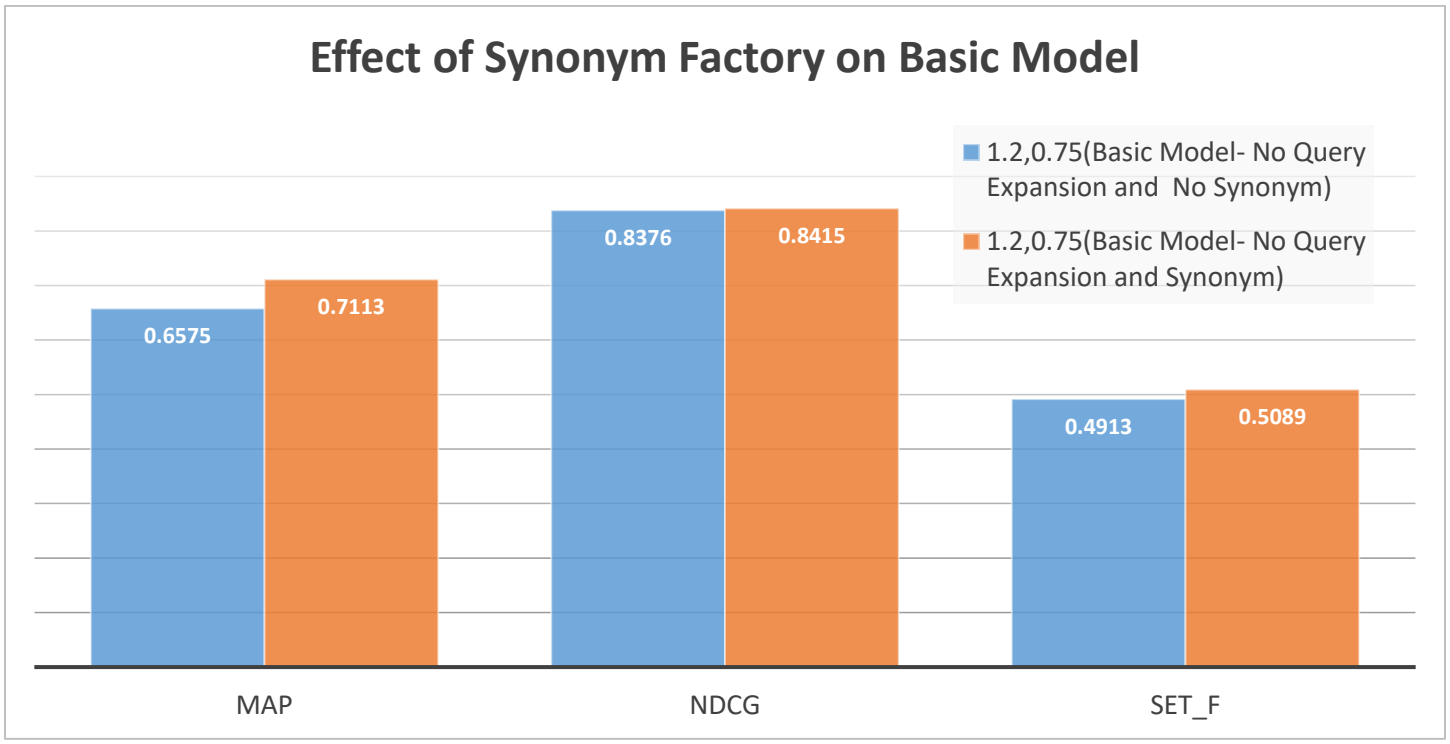

Fig. 1.2

7. Since the original query was in English we assumed that the relevant result should be in the same language as the user will expect to see more results in the language of his query, hence we provide more weight to the english text, and our final query looks like as follows:

   text_en:(“Russia Syria”~5 OR Russia's intervention^2 in Syria)^3 OR text_de:(“Russlands Syrien”~5 OR Russlands Intervention^2 in Syrien)^1 OR text_ru:(“России Сирии”~5 OR Вмешательство России в Сирии)
8. The graph (fig. 1) displays the before and after effect of query expansion on MAP.

Building the Okapi BM25 model

The steps used to implement DFR Model is illustrated in the following steps:

1. Initially we implement the model in “Managed-Schema” of Solr, by adding <similarity> tag as a global parameter.
2. Next we invoke the DFR Similarity class by adding, **class="solr. BM25SimilarityFactory",** attribute in the <similarity>
3. BM25 has two free parameters, namely k1 and b, as displayed in the formula below.
4. As we know, if the value of b = 0 then the model becomes BM11, and the value of b = 1 then the model becomes BM15, and BM25 is a combination of both of these Models, we incremented the value of b from 0.3 to 0.9 in steps of 0.05 and values k1 from 1.2 to 2 in steps of 0.05.
5. Finally, we chose the best combination of k1, b that gives the highest MAP value, decided from the graph (Fig. 1.3) below.

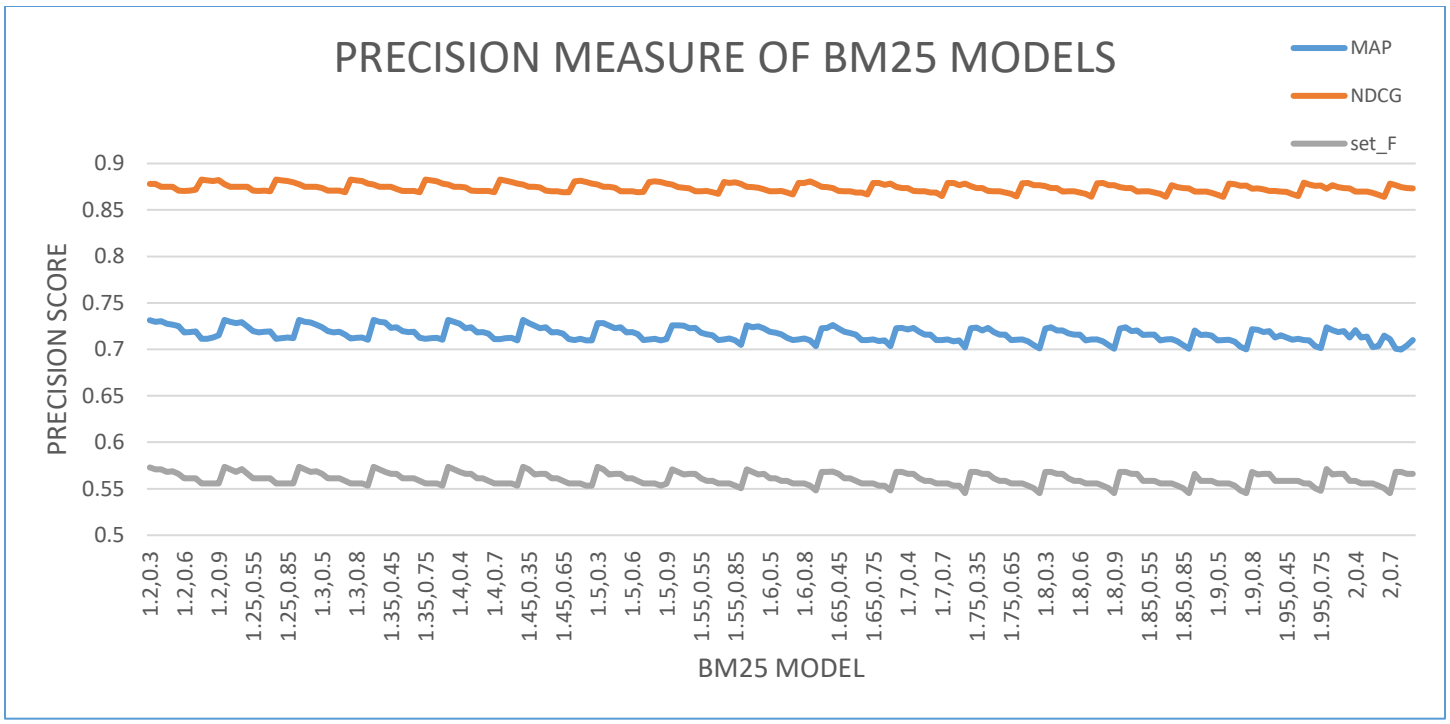

Fig. 1.3

6. Post choosing the combination, that is, k1 = 1.35 and b = 0.3, we achieved the best MAP score, and we display the comparison of the basic model and the selected model, both with expanded queries, in the figure 1.3 below.

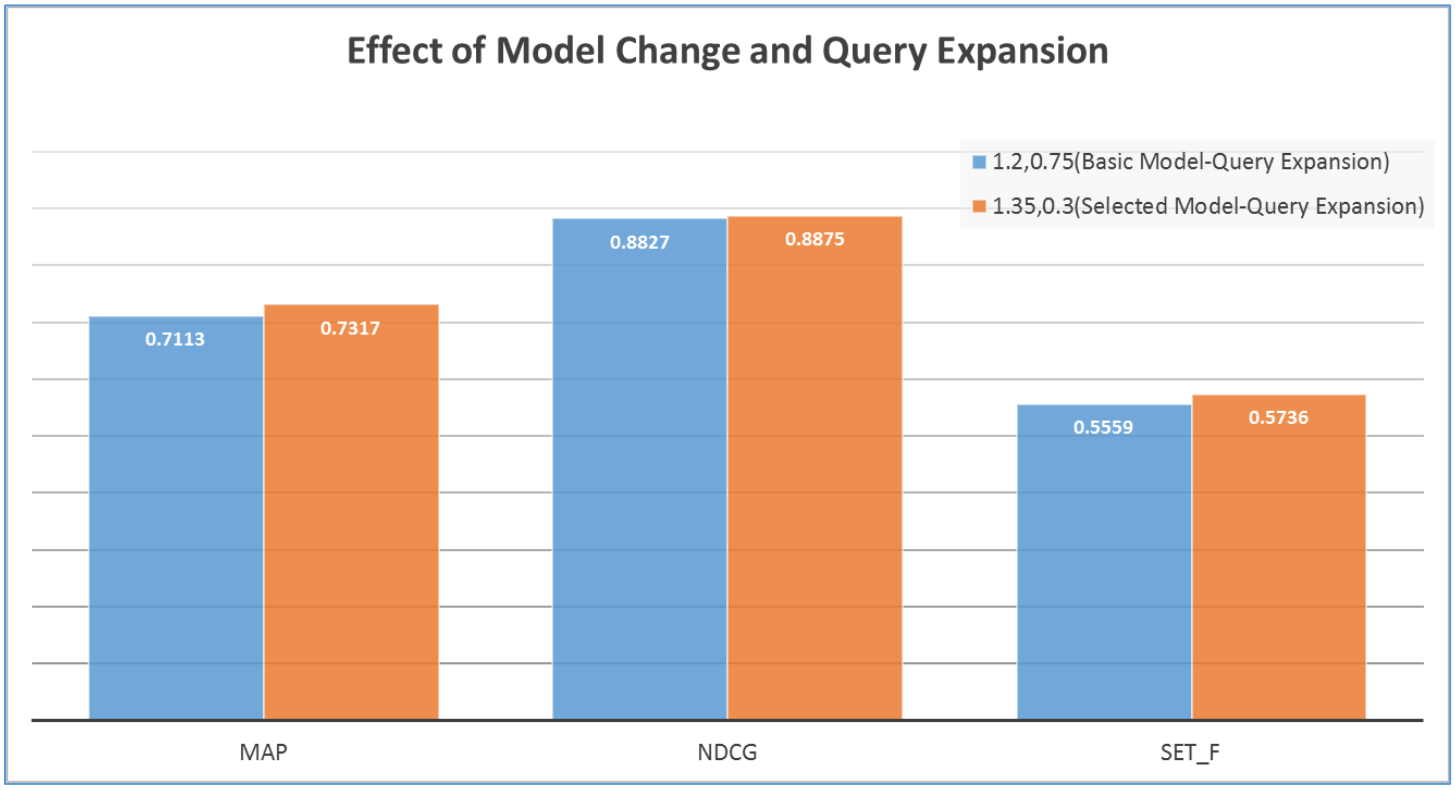

Fig. 1.4

Building the Divergence from Randomness(DFR) model

The steps used to implement DFR Model is illustrated in the following steps:

1. Initially we implement the model in “Managed-Schema” of Solr, by adding <similarity> tag as a global parameter.
2. Next we invoke the DFR Similarity class by adding, **class="solr.DFRSimilarityFactory",** attribute in the <similarity>
3. We firstly used un-expanded queries with basic model and compared results with the effect of query expansion on the basic model to gain a better MAP score as illustrated in Fig. 1.5

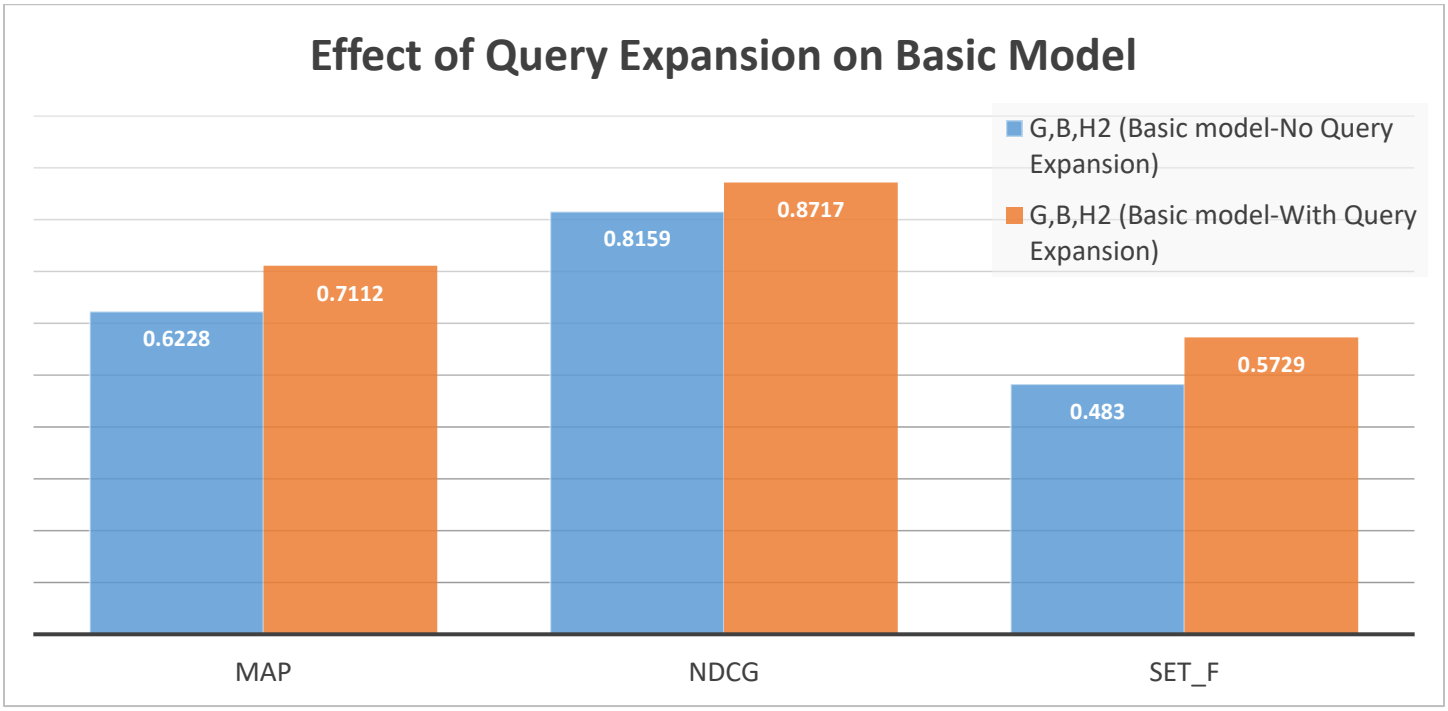

Fig. 1.5

4. Then we tune BasicModel, AfterEffect and Normalization parameters, added within the <similarity>, by iterating over each combination of their respective values.
5. We then plot a graph (Fig. 1.6) of the values of MAP, nDCG and F_Measure for each combination.

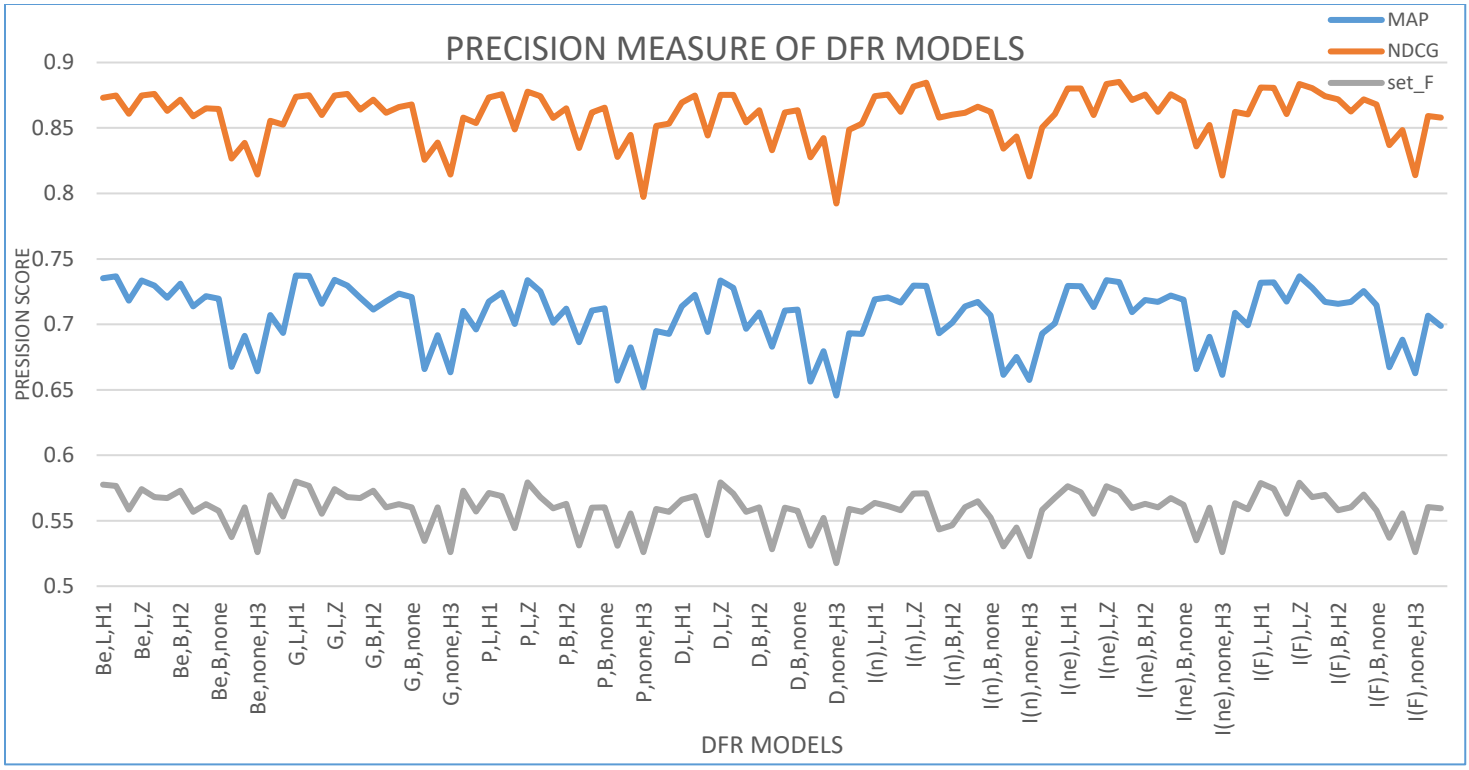

Fig. 1.6

6. Post choosing the combination, that is, BasicModel = G, AfterEffect=L and Normalization=H1, we achieved the best MAP score, and we display the comparison of the basic model and the selected model, both with expanded queries, in the figure 1.7 below.

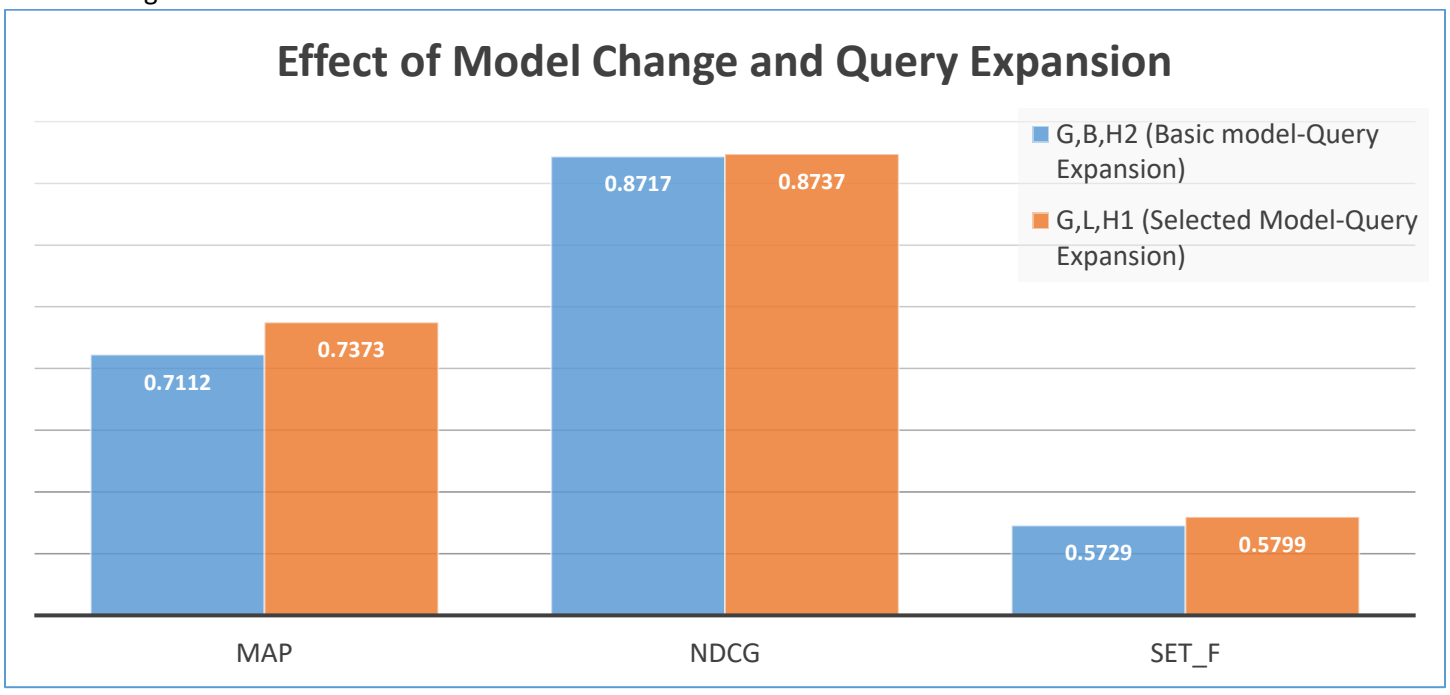

Fig. 1.7

Building the Vector Space Model(VSM)

The steps used to implement the Vector Space Model is illustrated in the following steps:

1. Initially we implement the model in “Managed-Schema” of Solr, by adding <similarity> tag as a global parameter.
2. Next we invoke the DFR Similarity class by adding, **class="solr. ClassicSimilarityFactory",** attribute in the <similarity>
3. We firstly used un-expanded queries with basic model and compared results with the effect of query expansion on the basic model to gain a better MAP score as illustrated in Fig. 1.8

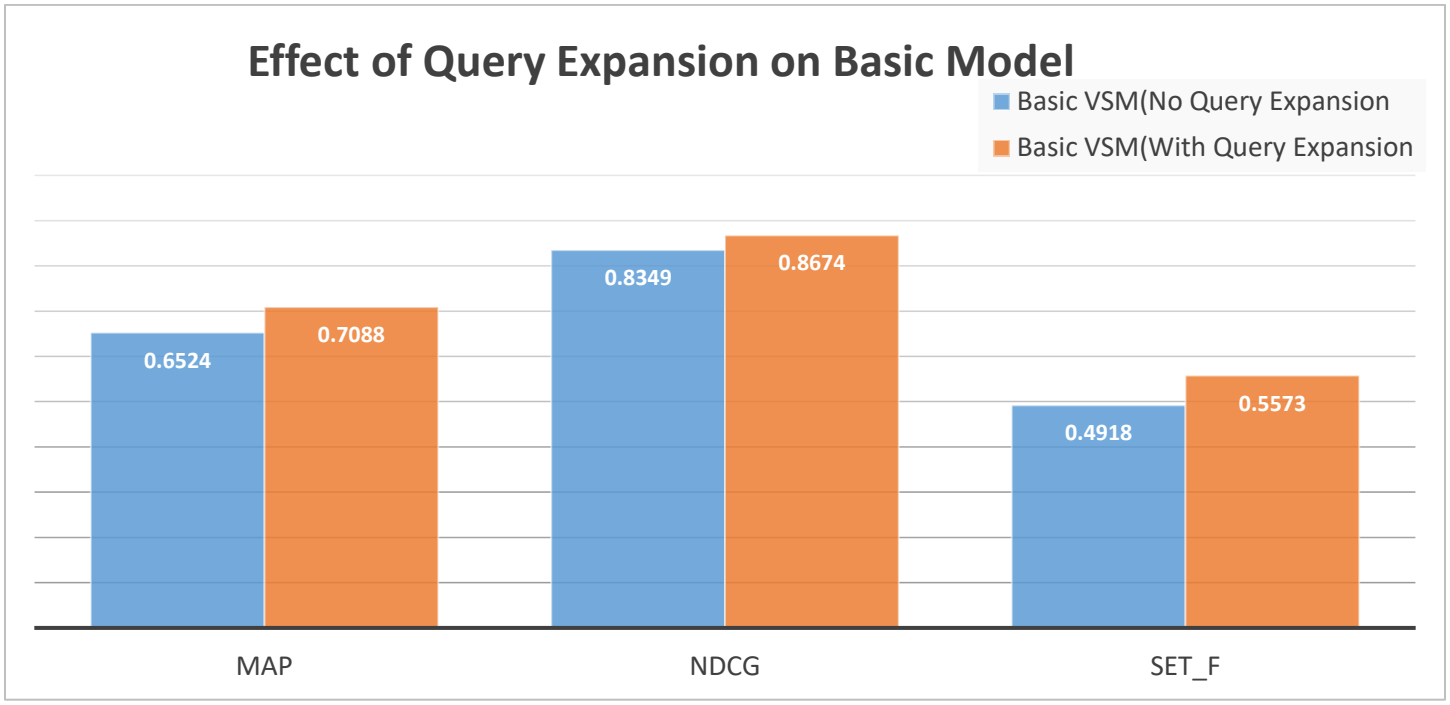

Fig. 1.8

4. Then we extended the ClassicSimilarityFactory class :- class **VSMSimilarityFactory** extends ClassicSimilarity
   Next we invoke the DFR Similarity class by adding, **class=VSMSimilarityFactory ",** attribute in the <similarity>
5. We exported the the jar into dist folder of Solr and made the following change in solrconfig.xml in-order to get the effect of the extended class <lib dir="${solr.install.dir:../../../..}/dist/" regex=".*\.jar" />
6. We then plot a graph (Fig. 1.9) of the values of MAP, nDCG and F_Measure before and after addition of the extended class

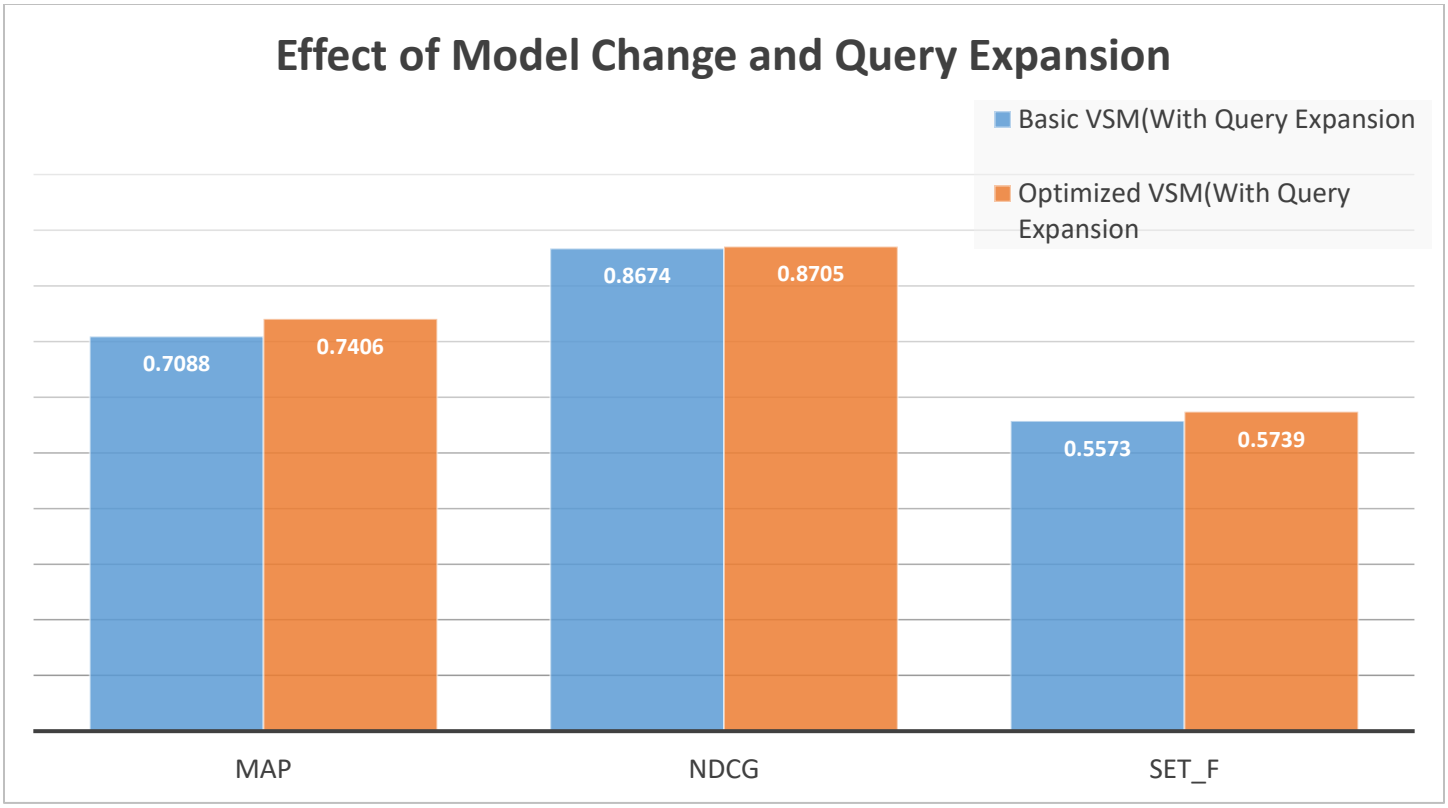

Fig. 1.9

Automation – Powershell

1. We wrote a custom script to automate the process of changing the parameters in the managed schema, restarting the solr and running the python file to generate trec eval input
2. This saved us a lot of time and we could calculate over an exhaustive range of values.
3. The file is present in the src folder for reference.

Conclusion

From the above experiment we have concluded the following:

1. The optimization is dependent on the nature of the data, and has to be an iterative process.
2. The best output model is obtained for DFR model
3. The MAP score, nDCG and F_measure changes as per the configuration of the respective model.
4. Weighing the query terms accurately plays a very important factor in increasing the precision