CSE 535 – Information Retrieval

Project 2: IR Models, Query Processing and Evaluation

Introduction:

There are various IR Models available to perform scoring and ranking namely the Vector Space Model, BM25. These are different models which give results that vary with the collection that they are implemented on and the way the scoring and ranking are done. The project compares these models for their performance on the TREC 4,5 collections. Once the models are compared for their performance and a baseline is obtained, query processing is done to improve the performance and see how the queries which leverage more of the information need actually improve the performance. This is then compared with the baseline performance to understand the different query expansion techniques and how they can improve performance.

Section 1:

This section describes the step wise procedure to execute the baseline code and how the performance metrics are compared. Comparisons give an idea of which is the better model given an expected IR system, the model that is having better early precision, the model that has better results for varying lengths of the query and for out of vocabulary terms present in the query.

This section also tries to explain how the code can be modified to accommodate Boolean queries and how the parameters of the various models can be modified to improve the performance.

Baseline code execution steps:

The project consists of two parts, one being the java code that is capable of producing the ranked results for both the models and another is the evaluation program called trec_eval that is responsible for comparing the given relevance judgments against the produced results and provide with statistics for each of the scenarios executed.

The output of the first part is always stored into result/result.out and then used while running trec_eval against test-data/qrels.trec6-8.nocr.

1. BatchSearch.java:

This is the program used to perform a search on the given index and obtain results for a given set of queries at a time. The file can be run in the following manner:

java BatchSearch [-index dir] [-simfn similarity] [-field f] [-queries file]

where the similarity function can be specified as either “default” for Vector Space Model similarity computation or ‘’bm25” for BM25 model similarity computation. The field gives the filed which has to be searched for (“contents” is the only field that is present in the index so it is hardcoded in the program) and the index file/ directory, file that contains the queries is specified.

The purpose of doing batch search is to obtain the first 1000 results of all the 150 queries that are being executed from the test-data/title-queries.301-450. If there aren’t as many as 1000 results for the given query then whatever has been found is printed as output.

2. Trec_eval:

The output of the above program should be in a format as follows:

Qid	iter	docno	rank	sim	run_id
301	Q0	FBIS4-41991	0	17.554089	bm25

where the similarity and qid-docno pair are used to compare with the available relevance judgments in order to come up with statistics about the IR model over the indexed collection. Here the ranking is done by sorting the similarity scores but rows with same scores are ranked based upon the earliness of retrieval i.e. the earlier document is ranked above the later one.

Comparison of trec_eval results for both the models:

Initially when the trec_eval is done on the output of VSM and BM25 models, the output is obtained as given by Figure 1 – Vector Space Model and Figure 2 – BM25 respectively. It is observed that the VSM model has slightly more performance for the given collection which is evident from the Mean Average Precision value which can be considered as a single value measure that estimates the precision of the model on the relevant documents retrieved averaged over all the information needs. The VSM has a MAP value of 0.1730 which is slightly greater than that of BM25 which has a MAP estimate of 0.1722. But, in case of early precision values: BM25 performs better when compared with VSM given that the precision @ K for K = 5,10,15,20,30,100 (where K represents the document retrieved) ,the values of precision is higher for BM25 model. It can be found that the R-Precision values are same for both the

models which explain that they have the top |Relevant| number of documents retrieved in the similar fashion or in other words the

```
aswin@aswin-pc:~/workspace/project2$ trec_eval test-data/qrels.trec6-8.nocr test-data/default.out
runid                   all     default
num_q                   all     150
num_ret                 all     137122
num_rel                 all     13692
num_rel_ret             all     6151
map                     all     0.1730
gm_map                  all     0.0819
Rprec                   all     0.2227
bpref                   all     0.2016
recip_rank              all     0.5795
iprec_at_recall_0.00    all     0.6342
iprec_at_recall_0.10    all     0.4274
iprec_at_recall_0.20    all     0.3101
iprec_at_recall_0.30    all     0.2253
iprec_at_recall_0.40    all     0.1771
iprec_at_recall_0.50    all     0.1376
iprec_at_recall_0.60    all     0.0914
iprec_at_recall_0.70    all     0.0649
iprec_at_recall_0.80    all     0.0424
iprec_at_recall_0.90    all     0.0266
iprec_at_recall_1.00    all     0.0102
P_5                     all     0.4307
P_10                    all     0.3733
P_15                    all     0.3373
P_20                    all     0.3113
P_30                    all     0.2813
P_100                   all     0.1707
P_200                   all     0.1176
P_500                   all     0.0666
P_1000                  all     0.0410
```

Figure 1 – Vector Space Model

```
aswin@aswin-pc:~/workspace/project2$ trec_eval test-data/qrels.trec6-8.nocr test-data/bm25.out
runid                   all     bm25
num_q                   all     150
num_ret                 all     137116
num_rel                 all     13692
num_rel_ret             all     6064
map                     all     0.1722
gm_map                  all     0.0830
Rprec                   all     0.2227
bpref                   all     0.1989
recip_rank              all     0.6016
iprec_at_recall_0.00    all     0.6535
iprec_at_recall_0.10    all     0.4348
iprec_at_recall_0.20    all     0.3143
iprec_at_recall_0.30    all     0.2216
iprec_at_recall_0.40    all     0.1700
iprec_at_recall_0.50    all     0.1289
iprec_at_recall_0.60    all     0.0867
iprec_at_recall_0.70    all     0.0592
iprec_at_recall_0.80    all     0.0359
iprec_at_recall_0.90    all     0.0158
iprec_at_recall_1.00    all     0.0099
P_5                     all     0.4360
P_10                    all     0.3920
P_15                    all     0.3631
P_20                    all     0.3337
P_30                    all     0.2924
P_100                   all     0.1708
P_200                   all     0.1162
P_500                   all     0.0658
P_1000                  all     0.0404
```

Figure 2 – BM25

ratio of the number of relevant documents retrieved in the first |Relevant| number of documents to the number of relevant documents is almost similar for both the models.

One of the standards for evaluating the IR models is 11-point interpolated-precision curve.

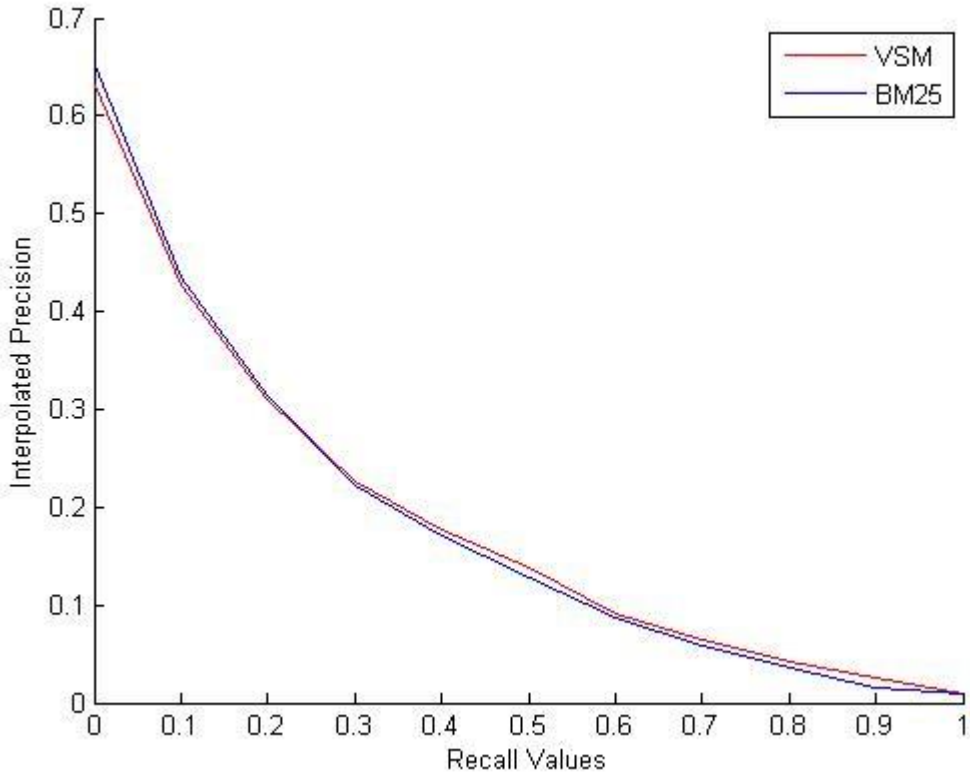

Figure 3 – Interpolated Precision

The Figure 3 – Interpolated Precision shows how the interpolated precision values vary at different recall levels. It is observed that both the models show a decreasing trend in the values of precision on increasing value of recall. But, the early interpolated values of precision in the BM25 model are slightly greater than that of VSM. This clearly supports the observation about early precision.

Effect of Query length on the models:

In case of longer queries that try to leverage the information need the values of MAP and interpolated precision values are better than the ones obtained for shorter queries in case of both the models. This can be verified by comparing the results of running a single query with just one word with those with 2 words and all the words (which is the normal title query for which the results were exhibited earlier).The results of which are represented by the following screenshots:

```
aswin@aswin-pc:~/workspace/project2$ trec_eval test-data/qrels.trec6-8.nocr result/result.out
runid                   all     default
num_q                   all     149
num_ret                 all     109264
num_rel                 all     13631
num_rel_ret             all     3563
map                     all     0.0802
gm_map                  all     0.0045
Rprec                   all     0.1087
bpref                   all     0.1098
recip_rank              all     0.2624
iprec_at_recall_0.00    all     0.2916
iprec_at_recall_0.10    all     0.1897
iprec_at_recall_0.20    all     0.1436
iprec_at_recall_0.30    all     0.1033
iprec_at_recall_0.40    all     0.0774
iprec_at_recall_0.50    all     0.0598
iprec_at_recall_0.60    all     0.0478
iprec_at_recall_0.70    all     0.0383
iprec_at_recall_0.80    all     0.0288
iprec_at_recall_0.90    all     0.0201
iprec_at_recall_1.00    all     0.0145
P_5                     all     0.1691
P_10                    all     0.1523
P_15                    all     0.1369
P_20                    all     0.1292
P_30                    all     0.1139
P_100                   all     0.0770
P_200                   all     0.0568
P_500                   all     0.0353
P_1000                  all     0.0239
```

Figure 4 – VSM 1 word queries

```
aswin@aswin-pc:~/workspace/project2$ trec_eval test-data/qrels.trec6-8.nocr result/result.out
runid                   all     default
num_q                   all     150
num_ret                 all     129210
num_rel                 all     13692
num_rel_ret             all     5287
map                     all     0.1344
gm_map                  all     0.0320
Rprec                   all     0.1795
bpref                   all     0.1634
recip_rank              all     0.4382
iprec_at_recall_0.00    all     0.4892
iprec_at_recall_0.10    all     0.3137
iprec_at_recall_0.20    all     0.2437
iprec_at_recall_0.30    all     0.1789
iprec_at_recall_0.40    all     0.1431
iprec_at_recall_0.50    all     0.1105
iprec_at_recall_0.60    all     0.0755
iprec_at_recall_0.70    all     0.0564
iprec_at_recall_0.80    all     0.0389
iprec_at_recall_0.90    all     0.0244
iprec_at_recall_1.00    all     0.0104
P_5                     all     0.2960
P_10                    all     0.2800
P_15                    all     0.2564
P_20                    all     0.2370
P_30                    all     0.2187
P_100                   all     0.1365
P_200                   all     0.0967
P_500                   all     0.0567
P_1000                  all     0.0352
```

Figure 5 – VSM 2 word queries

```
aswin@aswin-pc:~/workspace/project2$ trec_eval test-data/qrels.trec6-8.nocr result/result.out
runid                   all     bm25
num_q                   all     149
num_ret                 all     109263
num_rel                 all     13631
num_rel_ret             all     3765
map                     all     0.0866
gm_map                  all     0.0052
Rprec                   all     0.1140
bpref                   all     0.1137
recip_rank              all     0.2829
iprec_at_recall_0.00    all     0.3065
iprec_at_recall_0.10    all     0.2037
iprec_at_recall_0.20    all     0.1537
iprec_at_recall_0.30    all     0.1114
iprec_at_recall_0.40    all     0.0820
iprec_at_recall_0.50    all     0.0646
iprec_at_recall_0.60    all     0.0521
iprec_at_recall_0.70    all     0.0406
iprec_at_recall_0.80    all     0.0296
iprec_at_recall_0.90    all     0.0206
iprec_at_recall_1.00    all     0.0146
P_5                     all     0.1973
P_10                    all     0.1705
P_15                    all     0.1570
P_20                    all     0.1473
P_30                    all     0.1273
P_100                   all     0.0832
P_200                   all     0.0593
P_500                   all     0.0373
P_1000                  all     0.0253
```

Figure 6 – BM25 1 word queries

```
aswin@aswin-pc:~/workspace/project2$ trec_eval test-data/qrels.trec6-8.nocr result/result.out
runid                   all     bm25
num_q                   all     150
num_ret                 all     129209
num_rel                 all     13692
num_rel_ret             all     5300
map                     all     0.1368
gm_map                  all     0.0331
Rprec                   all     0.1833
bpref                   all     0.1638
recip_rank              all     0.4813
iprec_at_recall_0.00    all     0.5256
iprec_at_recall_0.10    all     0.3378
iprec_at_recall_0.20    all     0.2433
iprec_at_recall_0.30    all     0.1794
iprec_at_recall_0.40    all     0.1406
iprec_at_recall_0.50    all     0.1045
iprec_at_recall_0.60    all     0.0779
iprec_at_recall_0.70    all     0.0508
iprec_at_recall_0.80    all     0.0316
iprec_at_recall_0.90    all     0.0159
iprec_at_recall_1.00    all     0.0100
P_5                     all     0.3467
P_10                    all     0.3067
P_15                    all     0.2818
P_20                    all     0.2583
P_30                    all     0.2331
P_100                   all     0.1417
P_200                   all     0.0976
P_500                   all     0.0572
P_1000                  all     0.0353
```

Figure 7 – BM25 2 word queries

```
// Queries which are of different word lengths
String[] pair = line.split(" ");
Query query;
if ( pair.length >= 3)
    query = parser.parse((pair[1]+" "+pair[2]).trim());
else if (pair.length == 2)
    query = parser.parse(pair[1]);
else
    continue;

    doBatchSearch(in, searcher, pair[0], query, simstring);
```

Figure 8 – creating 2 word queries

The Figure 8 – creating 2 word queries shows how the given title queries are modified to 2 word queries. It is also experimented and found that any combination of 2 word queries formulated from the title query are found to satisfy the trend found between 1 word 2 word and normal title queries.

But, in case of expanded queries that actually don't leverage the information need it is found to reduce the performance of the system. This is found by adding random terms from the description or narrative into the query terms. Therefore it is understood from this experiment that the Out of Vocabulary terms actually reduce the performance of the system.

There is another striking observation which is, when the queries are expanded to include the contents of the description as well as the narrative; due to the length of the query VSM fails to produce higher performance whereas BM25 is capable of consuming the length to produce higher performance.

Model-Query	MAP	Relevant docs retrieved
VSM-title+description	0.1888	6342
BM25-title+description	0.1904	6523
VSM-title+description+narrative	0.1800	6223
BM25-title+description+narrative	0.1955	6697

Modification of parameters:

Parameters namely term frequency; inverse document frequency can be set for the similarity model while actually computing the scores. This will eventually improve the performance of the system. Since the number of terms can't be found out unless the documents are available, the values of Inverse Document Frequency (IDF) were modified. Earlier the mean value of IDF was 4.7538. This was estimated by making use of the Search explanations for the expanded query

which is made use of to compare the results in section 3 of the report. When this value was doubled, the scores improved slightly. This is done by making use of a custom scorer.

Once inside the Searcher, a Collector is used for the scoring and sorting of the search results. These important objects are involved in a search:

1. The Weight object of the Query. The Weight object is an internal representation of the Query that allows the Query to be reused by the Searcher.
2. The Searcher that initiated the call.
3. A Filter for limiting the result set. Note, the Filter may be null.
4. A Sort object for specifying how to sort the results if the standard score based sort method is not desired.

When the parameters of the BM25 model are varied, there is performance difference observed. Figure 9 – k modified BM25 model shows how the MAP values are varied as we vary the value of k from 0.1 to 0.9 keeping b at 0.5.

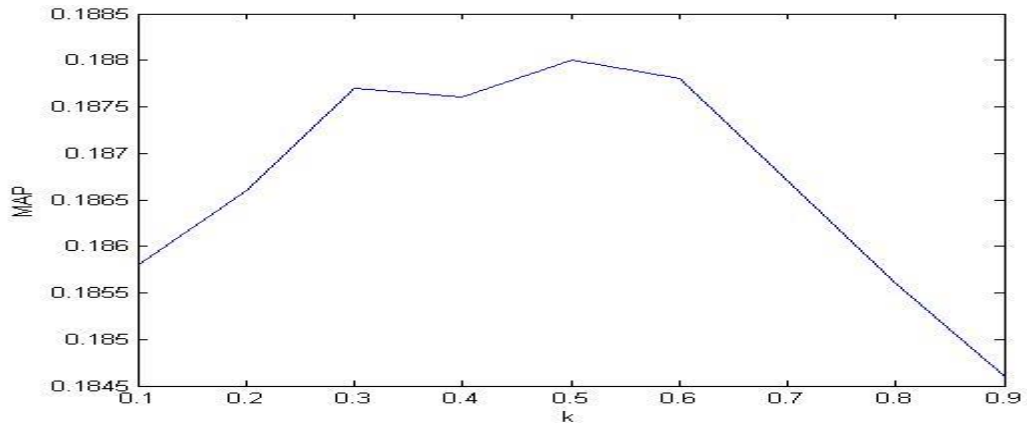

Figure 9 – k modified BM25 model

It can be inferred from the graph that the parameter values set at k = 0.5 and b = 0.5 maximum performance is obtained.

Boolean Model:

The search method which takes in the Collector object along with the query is used to process Boolean queries. BooleanQuery is the class that is made use of for formulating the queries. For example a negation and conjunction: (+X – Z) is represented by

BooleanQuery q = new BooleanQuery();.add(X, Occur.MUST); q.add(y, OCCUR_MUST_NOT);

The Collector API is responsible to incrementally add the query results for each query encountered.

Section 2:

This section presents an explanation about designed query processing module and shows how the techniques adopted in developing the module have been implemented in code.

The Query Processing module is developed by making use of two techniques:

1. Weighting different zones of the Topics file provided differently
2. Parts of Speech tagging

The three stages of query processing includes:

1. Text Parsing
2. Query Reformulation
3. Weighting/ Query Boosting

The various parts of the Query processor utilize the content present in topics.301-450 file. They try to leverage the queries that are used to evaluate the models upon all the three sections present in the information need provided by the 150 given topics. These are title, description and narrative. Though during the IR model evaluation without query processing took the title queries for the purpose of estimating the various collection statistics for each model, this method of taking into consideration the contents of the other two sections improves the performance of the system. Now, all the above parts of the designed query processor are explained in detail.

It is noted that the algorithm developed is aimed at replicating the QueryParserHelper class that is capable of handling the query processing by delegation of different activities to different components. It has a configuration module which is used to initiate the query processing. First step is query parsing. Then input query is treated as a node and is processed as a top-level query with the help of analyzers and filters. Then the query is built. The same way there is text parsing and then the query is reformulated and then built. The final step is weighting which is a heuristic that is used to improve the efficiency of the output. The complete flow is given in Figure 10 – Query Processing Design Diagram.

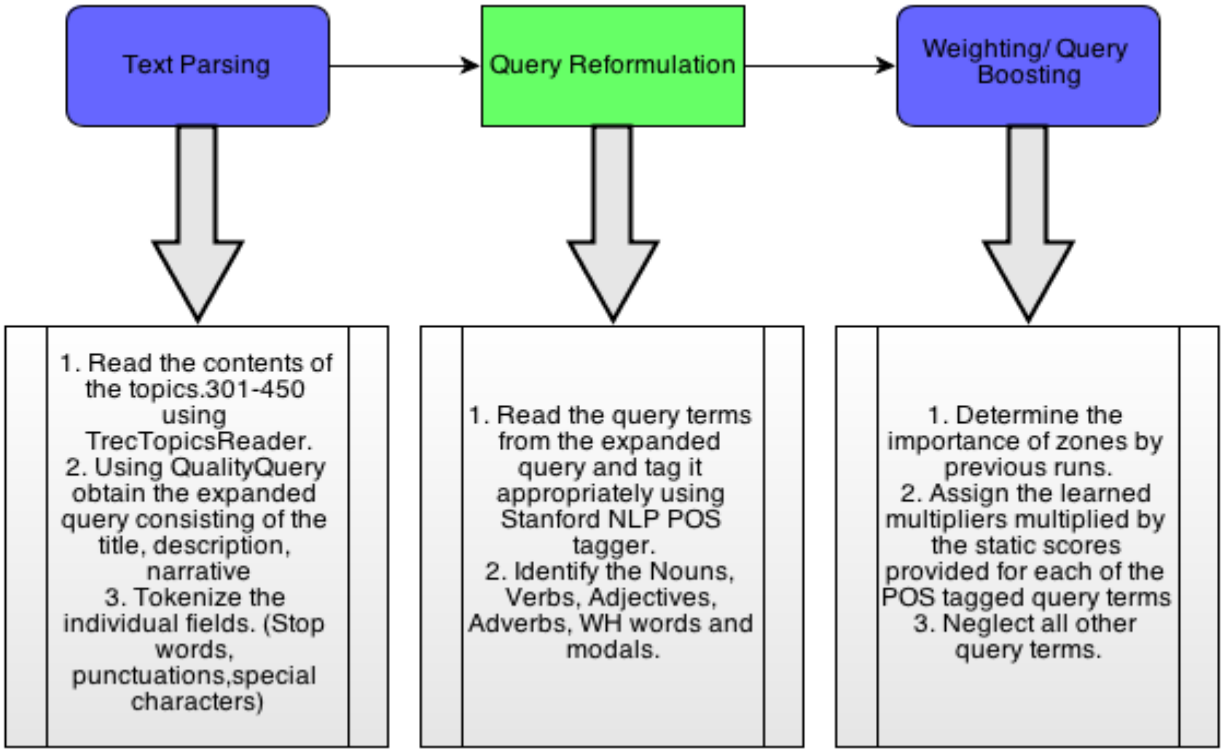

Figure 10 – Query Processing Design Diagram

Text Parsing:

In order to perform text parsing the details have to be obtained from the Topics file. This is done by making use of the TrecTopicsReader class present in the benchmarking API of Lucene. This class is capable of separating the individual topics and stores the content of its fields separately. This enables us to obtain the individual topics' decription and narrative in addition

```
public static QualityQuery[] trecTopicParse(){
    File topicsFile = new File("./test-data/topics.301-450");
    TrecTopicsReader topicReader = new TrecTopicsReader();
    QualityQuery expandedQuery[]  = null;
    try {
        expandedQuery = topicReader.readQueries(new
                BufferedReader(new FileReader(topicsFile)));
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

    return expandedQuery;
}
```

Figure 11 – Topics File Parsing

to the title. Once the Topics file has been processed (Figure 11 – Topics File Parsing) making use of Regular Expressions, all the unwanted special characters like brackets, colon, dashes, hyphens and quotation marks are removed. Once this is done, the StandaedAnalyzer is used to parse the contents of every topic and remove stopwords.

Weighted Query terms:

As the first and early approach towards Query processing: The individual sections of the topic are given a score boost as seen in Figure 12 – Zone weights over Query. This is done to estimate how the system performs with all the query terms given a weight to them depending upon the position they occur in the information need.

```
while(querycount < expandedQuery.length) {

    float score1 = 0.4F;
    float score2 = 0.2F;

    qno = expandedQuery[querycount].getQueryID();

    parsedTitle = parser.parse(expandedQuery[querycount]
            .getValue("title").replaceAll(pat.pattern(), ""));
    titleString = parsedTitle.toString().replaceAll(" ", "^"+score1+" ");
    titleString += "^"+score1;
    parsedDescription = parser.parse(expandedQuery[querycount]
            .getValue("description").replaceAll(pat.pattern(), ""));
    descString = parsedDescription.toString().replaceAll(" ", "^"+score1+" ");
    descString += "^"+score1;
    parsedNarrative = parser.parse(expandedQuery[querycount]
            .getValue("narrative").replaceAll(pat.pattern(), ""));
    narrativeString = parsedNarrative.toString().replaceAll(" ", "^"+score2+" ");
    narrativeString += "^"+score2;

    reformulatedQuery += qno+" "+titleString+" "+descString+" "+narrativeString+"\n";
    querycount++;

}
```

Figure 12 – Zone weights over Query

As it is evident from the code, the title and description hold a higher weight than the narrative section. This can be explained by the results obtained while comparing the IR models with all the terms in the topics included in the query and the corresponding performance of the IR models.

Model	MAP	Relevant docs retrieved
VSM-weighted	0.1890	6355
BM25-weighted	0.2001	6828

This result is very descriptive of the performance improvement that is possible through query processing. Given this result in hand, the query was reformulated to achieve better performance.

Query Reformulation:

This part of the design corresponds to the actual job of extracting the helpful parts present in a topic and trying to analyze them and then assign weights to them. Identification of such terms in the Topics given the parsed document is easier and the approach followed is divided into 2 steps.

1. POS tagging using the Stanford NLP POS tagger
2. Process the tagged and expanded query

The first part of POS tagging is done over the individual sections of the Topic so that we can further subject them to zone weighting later in the query processing. The Stanford NLP POS tagger is used by creating the maximum entropy tagger.

A bi-directional dependency network tagger i.e. english-bidirectional-distsim.tagger is made use of for utmost accuracy, but in this case we need only speed and minimal accuracy will be more than sufficient. So, for this purpose we make use of the model using only left sequence information and similar but less unknown words and lexical features as the previous model i.e. english-left3words-distsim.tagger (Figure 13 – POS Tagger).

```
public static String posTag(String input){
     // Initialize the tagger
    String field = "contents:";
    MaxentTagger tagger = new MaxentTagger("taggers/english-left3words-distsim.tagger");
    input = input.replaceAll(field, "");
    // The tagged string
    String tagged = tagger.tagTokenizedString(input);

    tagged = tagged.replaceAll(" ", " "+field);
    tagged = field+tagged;


    return tagged.substring(0, tagged.lastIndexOf(" "+field));
}
```

Figure 13 – POS Tagger

In addition to that, there is also some time saved in re tokenization of the terms because the parsing has already taken care of that. This can be made possible by calling the tagger over the string to be tagged using tagTokenizedString method. It is essential that the string is devoid of anything except the individual tokens to be tagged, it is therefore mandatory to remove the

field name from the parsed query before tagging and re insert them after the tagging is complete.

Once the title/description/narrative has been tagged, it is processed for the presence of Nouns, Verbs, Adjectives, Adverbs, WH-words, interjections and Modals. The rest of the tagging is conveniently neglected since they don't contribute to the information need. The tags for the above are represented using the Penn Treebank Tagset:

JJ Adjective	JJR Adjective, comparative	JJS Adjective, superlative	MD Modal
NN Noun, singular or mass	NNS Noun, plural	NNP Proper noun, singular	NNPS Proper noun, plural
RB Adverb	RBR Adverb, comparative	RBS Adverb, superlative	UH Interjection
VB Verb, base form	VBD Verb, past tense	VBG Verb, gerund or present participle	VBN Verb, past participle
VBP Verb, non 3rd person	VBZ 3rd person	WDT Whdeterminer	WP Whpronoun
WP$ Possessive Wh pronoun	WRB Wh adverb		

Regular expressions (Figure 14 – Regular expressions) are defined to identify these tags and give that to the next stage of processing which is the assignment of weights according to the zone in which the terms were found and the type of the term represented by these tags.

```
Pattern pat = Pattern.compile("[^\\s\\w]");
Pattern patN = Pattern.compile("_N[^ ]*");
Pattern patV = Pattern.compile("_V[^ ]*");
Pattern patJR = Pattern.compile("_[JR][^ ]*");
Pattern patUW = Pattern.compile("_[WU][^ ]*");
Pattern patNone = Pattern.compile("_[^ ]*");
```

Figure 14 – Regular expressions

The complete algorithm for reformulation is given by

```
while(querycount < expandedQuery.length) {
    float score1 = 0.4F,score2 = 0.2F, score3 = 0.3F, score4 = 0.1F;
    qno = expandedQuery[querycount].getQueryID();
    parsedTitle = parser.parse(expandedQuery[querycount]
            .getValue("title").replaceAll(pat.pattern(), ""));
    taggedTitle = parsedTitle.toString().replaceAll(" ", "^"+score1*6+" ");
    taggedTitle += "^"+score1;
    parsedDescription = parser.parse(expandedQuery[querycount]
            .getValue("description").replaceAll(pat.pattern(), ""));
    taggedDesc = posTag(parsedDescription.toString());
    taggedDesc = taggedDesc.replaceAll(patN.pattern(),"^"+(score1*3)+" ");
    taggedDesc = taggedDesc.replaceAll(patV.pattern(),"^"+(score3*2)+" ");
    taggedDesc = taggedDesc.replaceAll(patJR.pattern(),"^"+(score2*3)+" ");
    taggedDesc = taggedDesc.replaceAll(patUW.pattern(),"^"+(score4*3)+" ");
    taggedDesc = taggedDesc.replaceAll(patNone.pattern(),"");
    parsedNarrative = parser.parse(expandedQuery[querycount]
            .getValue("narrative").replaceAll(pat.pattern(), ""));
    taggedNarrative = posTag(parsedNarrative.toString());
    taggedNarrative = taggedNarrative.replaceAll(patN.pattern(),"^"+(score1*2)+" ");
    taggedNarrative = taggedNarrative.replaceAll(patV.pattern(),"^"+(score3)+" ");
    taggedNarrative = taggedNarrative.replaceAll(patJR.pattern(),"^"+(score2*2)+" ");
    taggedNarrative = taggedNarrative.replaceAll(patUW.pattern(),"^"+(score4*2)+" ");
    taggedNarrative = taggedNarrative.replaceAll(patNone.pattern(),"");

    reformulatedQuery += qno+" "+taggedTitle+" "+taggedDesc+" "+taggedNarrative+"\n";

    querycount++;

}
bw.write(reformulatedQuery);
```

Figure 15 – Query Reformulation and weighting code

Weighting/ Query boosting:

The process of weighting the terms is based upon the results that were obtained by running the collection with queries involving several combinations between title, description and narrative. The weight is higher for Nouns and then closely followed by verbs. The adjectives receive a higher weight than the interjections and WH- words. These values are depending upon the

frequency of these terms and also the effect of improving their scores on the performance of the system.

The title is not POS tagged for all the terms in title are highly important for the queries to return good results as in Figure 15 – Query Reformulation and weighting code. Therefore they are boosted by a constant factor. Whereas the terms in the description are much more succinct in describing the information need and therefore are given higher values for the multiples than that for narrative.

It is evident that the other terms in the query are not provided with any boosting factor and are left as it is.

Section 3:

In this section it is explained as of:

1. Which model performed well overall after the query processing?
2. Which technique of query expansion is better?
3. What are the other techniques that can be tried?

Answering the first question, here is the snapshot of the results from the query processed IR models:

```
aswin@aswin-pc:~/workspace/project2$ trec_eval test-data/qrels.trec6-8.nocr result/result.out
runid                   all     default
num_q                   all     150
num_ret                 all     149994
num_rel                 all     13692
num_rel_ret             all     6624
map                     all     0.1898
gm_map                  all     0.0989
Rprec                   all     0.2384
bpref                   all     0.2104
recip_rank              all     0.6809
iprec_at_recall_0.00    all     0.7213
iprec_at_recall_0.10    all     0.4466
iprec_at_recall_0.20    all     0.3339
iprec_at_recall_0.30    all     0.2583
iprec_at_recall_0.40    all     0.1996
iprec_at_recall_0.50    all     0.1561
iprec_at_recall_0.60    all     0.1064
iprec_at_recall_0.70    all     0.0715
iprec_at_recall_0.80    all     0.0407
iprec_at_recall_0.90    all     0.0204
iprec_at_recall_1.00    all     0.0083
P_5                     all     0.4573
P_10                    all     0.4153
P_15                    all     0.3804
P_20                    all     0.3480
P_30                    all     0.3073
P_100                   all     0.1832
P_200                   all     0.1262
P_500                   all     0.0708
P_1000                  all     0.0442
```

Figure 16 – VSM after Query processing

```
aswin@aswin-pc:~/workspace/project2$ trec_eval test-data/qrels.trec6-8.nocr result/result.out
runid                   all     bm25
num_q                   all     150
num_ret                 all     149992
num_rel                 all     13692
num_rel_ret             all     6854
map                     all     0.2035
gm_map                  all     0.1163
Rprec                   all     0.2516
bpref                   all     0.2241
recip_rank              all     0.7190
iprec_at_recall_0.00    all     0.7509
iprec_at_recall_0.10    all     0.5087
iprec_at_recall_0.20    all     0.3735
iprec_at_recall_0.30    all     0.2854
iprec_at_recall_0.40    all     0.2072
iprec_at_recall_0.50    all     0.1594
iprec_at_recall_0.60    all     0.0967
iprec_at_recall_0.70    all     0.0660
iprec_at_recall_0.80    all     0.0331
iprec_at_recall_0.90    all     0.0155
iprec_at_recall_1.00    all     0.0061
P_5                     all     0.5227
P_10                    all     0.4440
P_15                    all     0.4013
P_20                    all     0.3757
P_30                    all     0.3307
P_100                   all     0.1905
P_200                   all     0.1298
P_500                   all     0.0738
P_1000                  all     0.0457
```

Figure 17 – BM25 after Query processing

The TF-IDF based Vector Space Model is found to show better improvements after query processing and it can be further improved by measures that will be discussed later in the report. But, in spite of poor performance improvement after query processing the BM25 model is found to produce better performance over all.

When the values are compared with the baseline, all of them clearly are more than the baseline performance. A few observations are that the 11- point interpolated precision values (Figure 18 – Interpolated Precision after Query Processing) when plotted for the new values after query processing: It is seen that the VSM values of precision are very low when compared to BM25 but are greater than that of the non-query processed model. It is also seen that the VSM performs so well at higher recall than BM25 and thus explains why VSM improves after query processing but BM25 is good at overall performance.

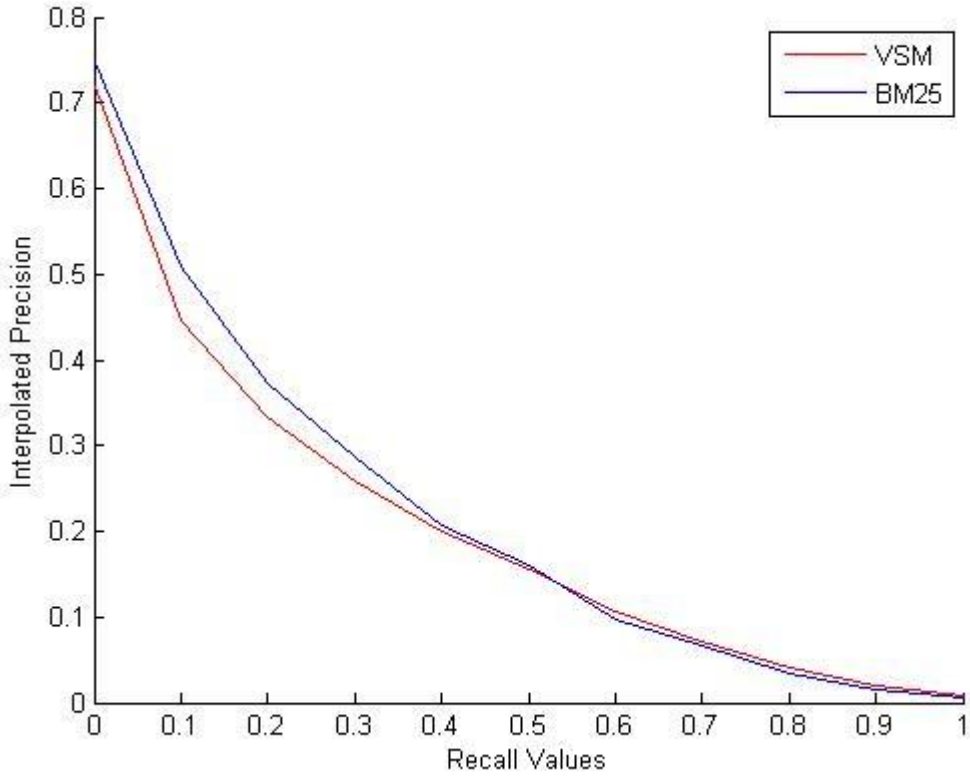

Figure 18 – Interpolated Precision after Query Processing

The second question is to explain which of the query expansion techniques worked.

Model	MAP	Relevant docs retrieved
VSM-weighted	0.1890	6355
BM25-weighted	0.2001	6828

These were the values obtained over weighting the zones of the topic file. The snapshots that explain the results of the query processing done using POS tagging shows better results if not the best results. The main reason for this betterment of results is because of the ability of the POS tagging to neutralize the increasing number of terms in the query.

$$score(q,d) = coord\text{-}factor(q,d) \cdot query\text{-}boost(q) \cdot \frac{V(q) \cdot V(d)}{|V(q)|} \cdot doc\text{-}len\text{-}norm(d) \cdot doc\text{-}boost(d)$$

Lucene Conceptual Scoring Formula

Figure 19 – Scoring Formula used for all the Similarity models

The lucene scoring (Figure 19 – Scoring Formula used for all the Similarity models) is dependent upon the coord-factor of the query and also the length of the query vector. This is drastically increased on query expansion and eventually lowers the score against a matching document. In order to improve this parsing is done that removes the unwanted terms and the reformulation process takes into consideration the type of terms and boosts its scores only if they actually contribute to the score. These two operations are part of the Query processing and thus at the end of it there is improvement in the MAP and number of relevant documents retrieved.

The third question to be answered leads to the extension of the query processing module which will lead to improved results. There are certain things that can be done to tokenize the terms much more efficiently like:

1. Using an analyzer that is prompt for the system being implemented
2. Removal of terms like “relevant” which occur in almost all the topic narratives
3. Introducing Capitalization and Hyphenation to improve search results

In addition to modification of the existing queries a thesaurus can be constructed and the values can be added to the queries.

The document can be indexed zone wise and the values be matched with them.

The POS tagging done is not completely exploited in the developed module. It can be futher made use of by developing phrase queries from them and improving the search results. The various types of tags can be segregated and then boosted.

The tags can reveal very essential information about the important term in a given sentence. Therefore the narratives can be subjected to sentence wise POS tagging and the important terms in the sentence, namely the noun and the verbs can be taken for query formulation.

Conclusion:

The results of IR models namely Vector Space Model and BM25 were initially compared. Then a query processing module was developed to improve the retrieval performance. At last the improved models were compared with the previously developed baseline and found to have exceeded the performance of baseline.