University of New Hampshire

CS953

Team1 Prototype2

Author:
Amith Ramanagar Chandrashekar

Author:
Pooja Himanshu Oza

Author:
Rachel E Cates

Author:
Ahmed M Alnazer

Supervisor:
Dr. Laura Dietz

March 21, 2019

Contents

1 Problem Definition

There is an ever increasing demand of providing more efficient and effective ways of Knowledge Discovery to the users. With the explosion of the data in last decade and evolution in the information needs of the user, retrieving relevant information to meet the user demands is becoming a greater challenge. Our system is trying to find the solution to meet this challenge of retrieving complex information needs effectively. The motivation of the system is to retrieve relevant information from the collection, combine the retrieved information and present it to the user.

2 Approaches

2.1 Markov Random Field

A Markov Random Field (MRF) is a probability distribution p over variables $x_1, x_2....x_n$defined by an undirected graph G. Where p is defined as

$$p(x_1, x_2..x_n) = \frac{1}{Z} \prod_{c \in C} \phi_c(x_c)$$

and C denotes the set of cliques (potential functions).

Markov random field is performed by collecting multiple features (Aka Evidences) from a query-document pair in a learning to rank setting to generate a scalar score for a document using the feature vector. The Evidences collected to create the feature vector is described in the next subsections.

2.1.1 Evidence 1

The bm25 retrieval score is used as the feature one for the learning to rank for each query-document pair.

2.1.2 Evidence 2

The query and the document vectors is computed using the GloVe word embedding of 100 dimension. The process to create the vector is as follows

- Get the tokens using the Lucene English Analyzer for query and document terms except the STOP WORDS.
- Use the word embedding to retrieve the vector for each tokens and compute the centroid vector by averaging them. This vector is computed only once for the query tokens and for each document.
- Cosine similarity is computed between the query vector and the document vector and this is considered as the feature two

2.1.3 Evidence 3

For each document, the K nearest neighbor documents were found. The centroid is constructed using the K nearest neighbor as described in the section 2.1.2 and cosine similarity between the query vector and a centroid is computed and this is considered as feature three.

2.1.4 Joint Probability and General Notes

Instead of calculating the Joint probability, the learning to rank is used to learn a weight factor by using all the features generated for a query-document pair on the Benchmark-Y1 train set, this weight vector from the RankLib is used to score the Benchmark-Y1 Test result. (i.e,.. Weight vector from the train result is used to compute the result for the Benchmark-Y1 using its feature vector. All the feature vectors are Z score normalized.

2.1.5 Command line

```
java -jar -Xmx20g target/cs953-team1-1.0-SNAPSHOT-jar-with-dependencies.jar
search section  -i /home/team1/indexed_file/
-q /home/team1/query_data/benchmarkY1-train/train.pages.cbor-outlines.cbor
-we /home/team1/glove_word_embeddings/glove.6B.100d.txt  -dim=100 -k=100
--qrel-path
/home/team1/query_data/benchmarkY1-train/train.pages.cbor-hierarchical.qrels
--mrf --parallel
```

2.2 Word Similarity

The retrieved documents is re-ranked using the String Similarity algorithm as described below.

2.2.1 Cosine Similarity

A cosine score is computed between all the query tokens with the document tokens and summed up to get a result for each query-document pair.

2.2.2 Jaccard Similarity

A Jaccard Similarity is computed between all the query tokens with the document tokens and summed up to get a result for each query-document pair.

2.2.3 Jaro Winkler Simialrity

A Jaro Winkler Simialrity score is computed between all the query tokens with the document tokens and summed up to get a result for each query-document pair.

2.2.4 Sorensen Dice Coefficient

A Sorensen Dice Coefficient score is computed between all the query tokens with the document tokens and summed up to get a result for each query-document pair.

2.2.5 Normalized Levenshtein

A Normalized Levenshtein score is computed between all the query tokens with the document tokens and summed up to get a result for each query-document pair.

2.2.6 Command Line

The below command line runs for all the above methods.

```
java -jar -Xmx20g target/cs953-team1-1.0-SNAPSHOT-jar-with-dependencies.jar
search section  -i /home/team1/indexed_file/
-q /home/team1/query_data/benchmarkY1-test/test.pages.cbor-outlines.cbor
--cosine-sim --jaccard-sim --jaro-sim --dice-sim --leven-sim --parallel
```

2.3 Query Expansion-Document Frequency reranking

Query expansion methods implemented in the prototype 1 is combined with Document-Similarity ranking variant (Document-Similarity based on the highest DFs) to rerank the initial retrieved document using query expansion.

The process is as follows

- Use query expansion method to expand the query using Entities based query expansion.
- Use the Document-Similarity variant to rerank the document considering the top 20 documents to build a centroid and perform the re ranking.

2.3.1 Command Line

Example to run one of the Query-Expansion type

```
java -jar -Xmx20g target/cs953-team1-1.0-SNAPSHOT-jar-with-dependencies.jar
search section -i /home/team1/indexed_file/
-q /home/team1/query_data/benchmarkY1-test/test.pages.cbor-outlines.cbor
-we /home/team1/glove_word_embeddings/glove.6B.50d.txt -dim=50 -k=100 --bias-fact=20
-qe-type entityText --qe-reranking
```

2.3.2 Methods Employed

For the query expansion we use QE using EntityID, EntityText and EntityTextID and this is re-ranked using the Document Similarity re-ranking variant based on choosing the highest DF terms to represent the document vector and by choosing the Top 20 document to build a centroid vector and each document score is calculated using Cosine Similarity between centroid vector and every other document.

2.4 Query expansion based on DBpedia

Expand the query by:-
Adding the split of entity pulled from bm25 to the main query.
Then query the new query into Dbpedia to get the count of rows returned.
Select the top n of these values to use to create new query.
Then reranking the new query using bm25
Still in the development since I start by using the entire entity with out split. result in not record found. now I try to split each entity then get the total row count for all the split term in the entity with the original Query.which take time.

2.5 Integration of the Spam filter to our base bm25

Add ability to remove any spam paragraph that is return from bm25

2.5.1 Method used:

We use Bigrams classifier witch has the best result than the other methods. Then query and apply the spam filter and return the clean result after removing the spam

2.5.2 Command:

We apply the filter on out base BM25 method by apply this command line argument
–spam-filter command to run with spam filter
–spam-loc location of the spam training file
–ham-loc location of the spam training file

2.6 Entity Relation

In entity relation, the passages are re-ranked based on the entity-entity relation of 1 hop and entity-entity context. Using these 2 features, 2 different methods have been applied to generate both entity retrieval and passage retrieval.

2.6.1 Features

- Entity-Entity Relation
  - 1 hop relation - whether the entity exists in the outlinks/inlinks of the entity
- Entity-Entity Context
  - How many times the entities are co-mentioned in the context. The context being the passage

2.6.2 Database Generation

Mongodb collection entityIndex is generated using entity index. This database collection is used for generating the feature vectors. Each document in the collection contains entity details such as id, lead text, inlinks, outlinks, anchor names, disambiguation names.

2.6.3 Feature Vector Generation

Feature Vectors are generated using the above mentioned 2 features. The following process is followed for generating the feature vectors:

- For every query, retrieve BM25 results, by default 100 passages are retrieved
- For every query, get the entities present in the passages of the BM25 results
- For every entity, compare it with the outlinks and inlinks of the other entities present
- For every entity, count the number of times it is present in the same passage as the other entity
- Take average of both the above mentioned features, as total/entity length. Hence it will generate only 1 value for every feature for each entity in each query
- Normalize the above results, by the formula as: observation value - feature max value/ feature max value - feature min value

Once the feature vectors are generated the following approaches are applied on the feature vectors:

2.6.4 Approaches

- RankLib Approach
  - Trained Ranklib on the generated feature vectors and generated a model file
  - Dot product of the generated feature vectors with the Ranklib model values to get a scalar value. So now for each query for each entity there is a score.
  - Sort the entities based on the score
  - For each query, each passage, for every entity present in the passage add the entity score calculated
  - Rerank the passages based on the new score
- Centroid Approach
  - For each query, calculate the entity centroid vector as total/vectorlength
  - For each query, for each entity take the dot prodcut between the feature vector and the centroid vector. So now each query, each entity there is only 1 score.
  - Sort the entities based on the score
  - For each query, each passage, for every entity present in the passage add the entity score calculated
  - Rerank the passages based on the new score

2.6.5 Retrieval

There are two retrievals done in both the approaches, Entity Retrieval and Passage Retrieval

2.6.6 Commands

- RankLib Approach Command

```
java -jar target/cs953-team1-1.0-SNAPSHOT-jar-with-dependencies.jar search
--entity-ranklib section
-f /home/team1/prototype2/pooja_data/output_ranking_feature_vectors_section_test.txt
-model /home/team1/prototype2/pooja_data/model_section_train.txt
-i /home/team1/indexed_file/
-q /home/team1/query_data/benchmarkY1-test/test.pages.cbor-outlines.cbor
```

Results will be stored in the results folder in the current directory
output_ranking_entity_ranklib_section_test.txt for the entity results and
output_ranking_paragraph_ranklib_section_test.txt for the passage results

- Centroid Approach Command

```
java -jar target/cs953-team1-1.0-SNAPSHOT-jar-with-dependencies.jar search
--entity-centroid section
-f /home/team1/prototype2/pooja_data/output_ranking_feature_vectors_section_test.txt
-i /home/team1/indexed_file/
-q /home/team1/query_data/benchmarkY1-test/test.pages.cbor-outlines.cbor
```

  Results will be stored in the results folder in the current directory *output_ranking_entity_avg_centroid_section_test.txt* for the entity results and *output_ranking_paragraph_avg_centroid_section_test.txt* for the passage results

- Feature Vector Generation

```
java -jar target/cs953-team1-1.0-SNAPSHOT-jar-with-dependencies.jar search
--entity-relation section
--entity-index /home/team1/entity.lucene/
-q /home/team1/query_data/benchmarkY1-test/test.pages.cbor-outlines.cbor
-i /home/team1/indexed_file/
-qrel /home/team1/query_data/benchmarkY1-test/test.pages.cbor-hierarchical.entity.qrels
```

  Results will be stored in the results folder in the current directory *output_ranking_feature_vectors_section_test.txt* for the entity results and *output_ranking_rank_lib_section_test.txt* for the passage results

3 Document-Similarity based on Entity similarity

As defined in the prototype 1, the idea is to perform the document similarity based on the Entity mentions and its abstract.The document representation is built using the Entities abstract only. The main motivation is,if there is any low ranked document entity mentions refers to the top k documents entity mentions, bring the low ranked document up the order in the retrieval.

Process:

- Get the initial ranking list for the given Query Q, call it D_n
- Make an assumption that the document $d_{1..b}$ in D_n for a given query Q is relevant where b is a bias factor
- Compute the document representation for $d_{1..b}$ by adding all the terms in entity abstract in d_b.word embedding of certain dimension is retrieved using GloVe/Word2Vec for each word and their average is calculated. ie for each word get its vector , sum them up and take a mean of it.

- Compute the cosine similarity between d_b (biased vector) obtained in the previous step and d_1,d_2,d_3 ..... d_n. Document representation is calculated similar way for other documents.
- Use the Spam classification to decide whether a document should be considered for scoring or to ignore if it is a spam.
- Re-rank all the document by taking linear combination of d_n.Score * Cosine.Score + d_b.Score where n=1,2,3,.....n

The method evaluated after we generated the result graph and this methods output is missing.

MAP=0.1177

4 Evaluation

4.1 Data set

We use the TREC-CAR v2.0 for data set.

4.2 Corpus

ParagraphCorpus.v2.0 is used as corpus for Paragraph Index. UnprocessedAllbutBenchmark.v2.1 is used as corpus for indexing Entities and generating Mongodb collection. Mongodb collection is used as corpus for generating feature vectors.

4.3 Training

benchmarkY1-train.v2.0 is used as the training data set.

4.4 Test

benchmarkY1-test is used as test data set

4.5 Queries

We use benchmarkY1-test outlines to generate section level queries.

4.6 Ground Truth

We use benchmark-y1-test hierarchical qrels as the ground truth for section level retrieval and benchmark-y1-test.hierarchical.entity.qrels as the ground truth for section entity retrieval

4.7 Evaluation Measure

We use the following evaluation metrics:

- Mean Average Precision (MAP)

5 Results

5.1 Spam Filter on basic training and test sets

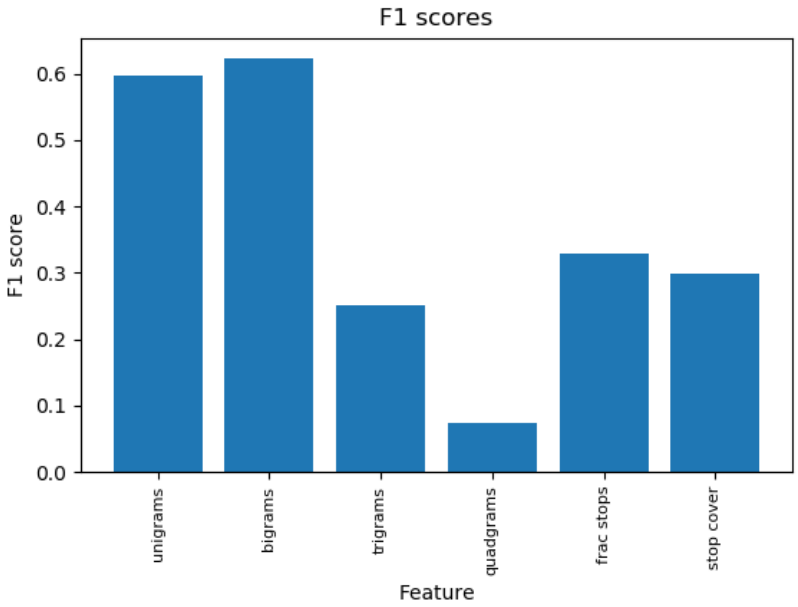

Figure 1: F1 scores for the spam classifier, trained and tested on the basic train and test data

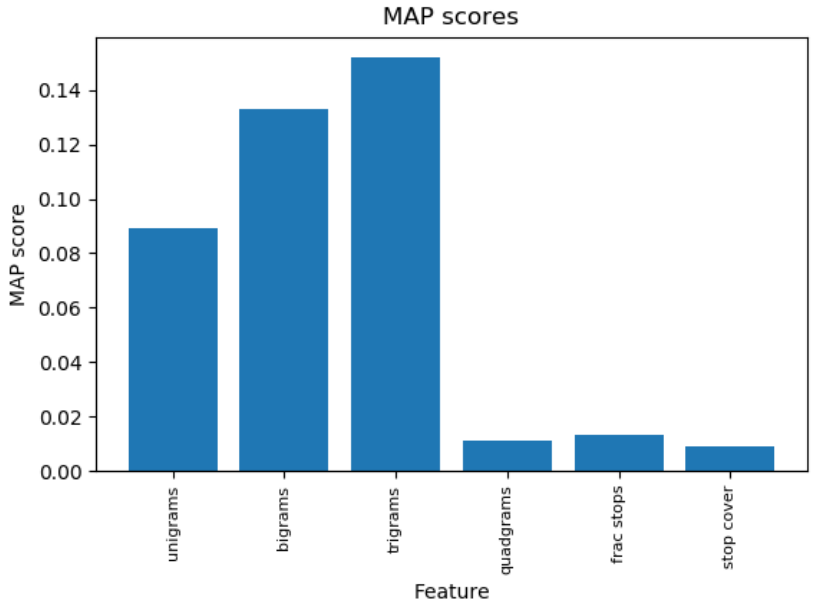

Figure 2: MAP scores for the spam classifier, trained and tested on the basic train and test data

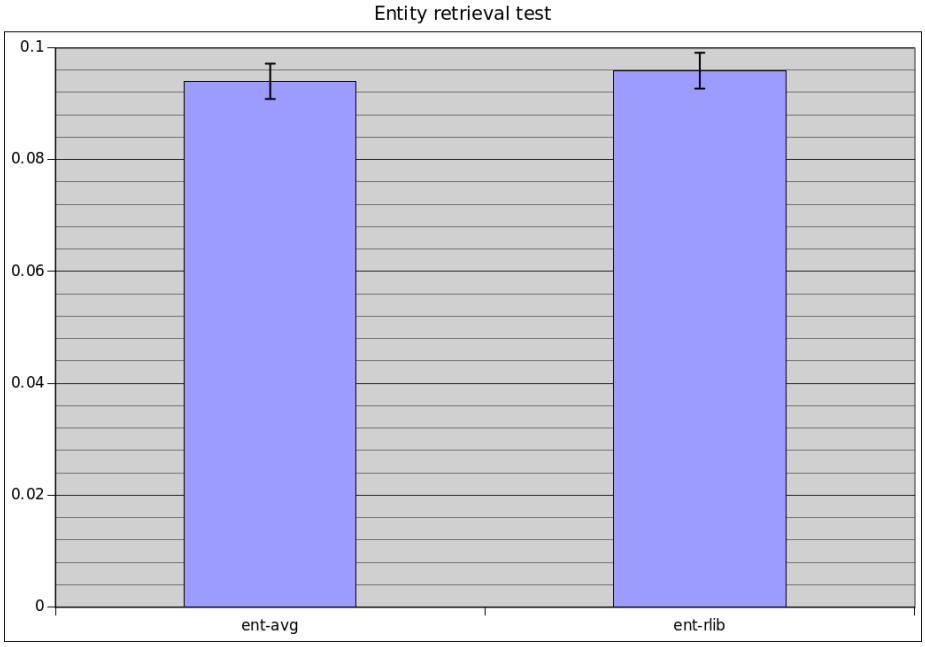

Figure 3: Test MAP Result for Entity Retrieval Approaches -
ent-avg = Centroid (section 2.6.4, Centroid Approach),
ent-rlib = RankLib (Section 2.6.4, RankLib Approach)

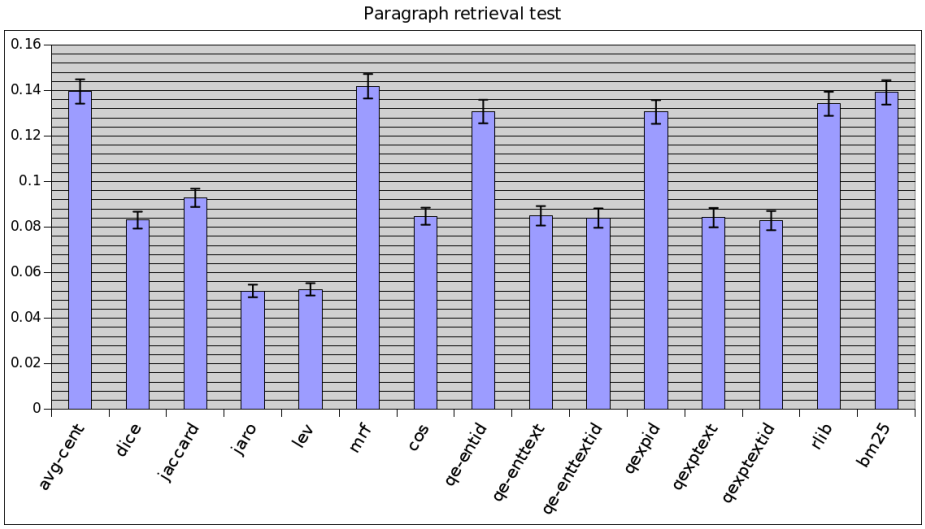

Figure 4: Test MAP Result without Spam Feature Enabled for Passage Retrieval Approaches

avg-cent = Centroid (section 2.6.4, Centroid Approach),
dice = Dice Coefficient (section 2.2.4, Dice Coefficient),
jaccard = Jaccard Similarity (section 2.2.2, Jaccard Similarity),
jaro = Jaro Winkler Similarity (section 2.2.3, Jaro Winkler Similarity),
lev = Normalized Levenshtein (section 2.2.5, Normalized Levenshtein),
mrf = Markov Random Field (section 2.1, Markov Random Field),
cos = Cosine Similarity (section 2.2.1, Cosine Similarity),
qe-entid = Query Expansion - Document Frequency Reranking (section 2.3),
qe-enttext = Query Expansion - Document Frequency Reranking (section 2.3),
qe-enttextid = Query Expansion - Document Frequency Reranking (section 2.3),
qexpid = Query Expansion - Document Frequency Reranking (section 2.3),
qexptext = Query Expansion - Document Frequency Reranking (section 2.3),
qexptextid = Query Expansion - Document Frequency Reranking (section 2.3),
rlib = RankLib (Section 2.6.4, RankLib Approach),
bm25 = BM25 Baseline

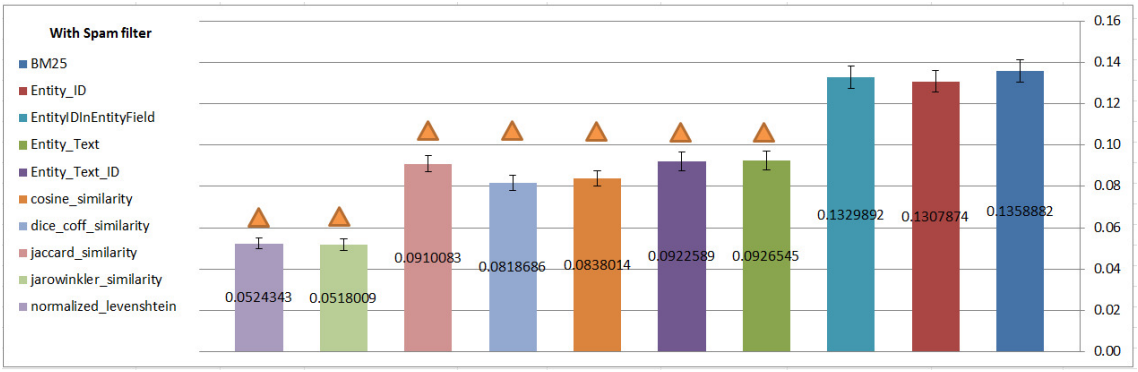

Figure 5: Test MAP Result with Spam Feature Enabled for Passage Retrieval Approaches

dice_coff_similarity = Dice Coefficient (section 2.2.4, Dice Coefficient),
jaccard_similarity = Jaccard Similarity (section 2.2.2, Jaccard Similarity),
jarowinkler_similarity = Jaro Winkler Similarity (section 2.2.3, Jaro Winkler Similarity),
normalized_levenshtein = Normalized Levenshtein (section 2.2.5, Normalized Levenshtein),
mrf = Markov Random Field (section 2.1, Markov Random Field),
cosine_similarity = Cosine Similarity (section 2.2.1, Cosine Similarity),
Entity_ID = Query Expansion - Document Frequency Reranking (section 2.3),
EntityIDInEntityField = Query Expansion - Document Frequency Reranking (section 2.3),
Entity_Text = Query Expansion - Document Frequency Reranking (section 2.3),
Entity_Text_ID = Query Expansion - Document Frequency Reranking (section

2.3),
BM25 = BM25 Baseline

6 Conclusion

The main motivation of the prototype 2 is to integrate the Spam filter that tries to remove the documents that are considered to be spam from the initial retrieved list, although not all methods employed spam filter but based on Figure 5 we can see that it did not help us to increase the MAP score, the reason being, the training data for the spam considering the big corpus is too less. The query-expansion method on top of the document similarity re-ranking method performed well but did not outperform the BM25 score. Some of the String similarity methods were also computed as part of this prototype but it does not help us to improve any score, no further exploration will be made on this in the future. The MRF and Centroid approach method outperformed the BM25 score by achieving the MAP of 0.1419 and 0.1396 respectively. Both the methods looks promising and can be further explored with more features in order to achieve a good MAP.

7 Contributions

- **Rachel:** Implemented different Spam Filters for everyone to use and Implemented a class to demonstrate it.
- **Amith:** Implemented Markov Random Field method,Wrapper to Prototype 1 methods, it is used for all the Ahmed's query expansion method. (i.e,..Perform Document Frequency reranking after performing the query expansion), Word Similarity algorithms such as cosine, Jaccard, jaro Winkler, Normalized Levenshtein and Sorensen Dice Coefficient Similarity, Entity-Document Similarity ranker, Added more Util functions,Installation script and build script/install.pdf
- **Pooja:** Implemented Feature Vectors generation, Entity Retrieval through Ranklib and Centroid approaches and Passage Retrieval through Ranklib and Centroid approaches. Mongodb installation, collection generation and construction.
- **Ahmed:** Implemented Query expansion using DBpedia not fully functional, Integrate the spam filter to the BaseBM25, Stacked up Query expansion method from prototype1 on top of the Document Similarity re ranker, helped to create the Standard bar error graphs for all the methods.