

Github Link - <https://github.com/yuanph4ever/TREC-Complex-answer-retrieval-Track>

Prototype 2

Team 5

Nithin, Peihao Yuan, Sangeeta

### 1. Task

- To perform and evaluate query feature expansion using entities in the paragraph, k-means clustering, with bm25 as baseline
- To re-rank the documents retrieved by document category type and the weight using DBpedia spotlight.
- To classify the documents retrieved from bm25 using certain machine learning models and evaluate with the ground truth.

### 2. Data

- For our experiment, we use “dedup.articles-paragraphs.cbor” as corpus, “benchmarkY1-train” data set as training data to train our models.
- For the classification we use base-para-cbor of train v2.0 for training the classifier models.

### 3. Evaluations and Baseline

We use Precision@R, MAP and MRR as evaluation methods. And we use BM25 to search for corpus by using queries from “train.pages.cbor-outlines.cbor” as baseline. Here’s the evaluation result for baseline.

	<a href="#">Precision@R</a>	MAP	MRR
BM25_pages	0.1393	0.0822	0.4054
<a href="#">BM25_sections</a>	0.0769	0.1039	0.1481

### 4. Methods and Results

#### 4.1. Pseudo Relevance Feedback with Entities

##### 4.1.1 Introduction

Pseudo relevance feedback, also known as blind relevance feedback, provides a method for automatic local analysis. It automates the manual part of relevance feedback, so that the user gets improved retrieval performance without an extended interaction. The method is to do

normal retrieval to find an initial set of most relevant documents, to then assume that the top k ranked documents are relevant, and finally to do relevance feedback as before under this assumption. [ <https://nlp.stanford.edu/IR-book/html/htmledition/pseudo-relevance-feedback-1.html>]

This task expands query by entities of first-round-answers from top1, top3, and top5. By using entities, we can remove the useless words with high term frequency and extract the words which should be given bias. We assume the result will be improved by doing this.

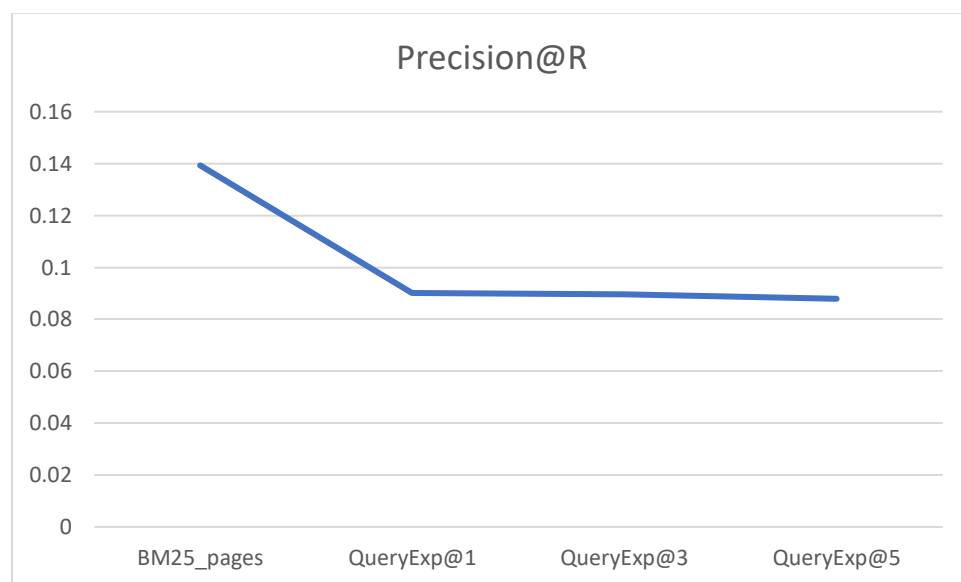
#### 4.1.2 Implementation

We use DBpedia Spotlight, which is a tool for automatically annotating mentions of DBpedia resources in text, to annotate entities from first-round-answers. For searching, we use BM25 method.

#### 4.1.3 Evaluations

For evaluation, we use MAP, Precision@R, and MRR. And BM25 on query of “page name” without query expansion is the baseline here. Following is the result of them.

	<a href="#">Precision@R</a>	MAP	MRR
BM25	0.1393	0.0822	0.4054
<a href="#">QueryExp@1</a>	0.0902	0.0517	0.367
<a href="#">QueryExp@3</a>	0.0895	0.0496	0.3727
<a href="#">QueryExp@5</a>	0.0879	0.0475	0.3543



As we can see from the plots, Precision@R decreases by increasing the number of k (top k first-round-answers). For MRR, k = 3 is better than k = 1 and k = 5. But it's still worse than disusing query expansion.

#### 4.1.4 Conclusion

Pseudo Relevance Feedback with Entities does not improve the result. Reversely, it decreases the measurement. I think the reason is, there is a tiny number of relevant document in the top retrieved answers because the score of evaluation for top 100 is so tiny as we can see. So actually, there are more noises than relevant feedback in the top retrieved documents.

### 4.2. Re-Rank by K-means Clustering

#### 4.2.1 Introduction

K-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. K-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as prototype as the cluster.

#### 4.2.2 Implementation

This task uses K-means clusters to re-rank runfile. The basic idea is, firstly retrieve paragraphs for queries as we did before. Second, assign a cluster to query by comparing the similarity between them and generate the rank of clusters for each retrieved paragraph and see what the position of cluster for query is and use the reversed number of it as the additional score. Finally, use original score plus additional score to re-rank paragraphs. The algorithm is,

$$\text{Score}(\text{query}, \text{paragraph}) = \text{BM25\_score}(q, p) + 10 / \text{para\_clu\_rank}(\text{query's cluster})$$

, where para\_clu\_rank(x) gives you the position of x in the rank of paragraph's clusters. For example, if the cluster for query Q is C1 and the rank of clusters for paragraph P is C3, C1, C2 (from top to bottom). Then the final score for P based on Q is the original score (given by BM25) plus 5 (10/2).

To achieve this, I took 10,000 paragraphs from "dedup.articles-paragraphs.cbor" to make my corpus for clustering. I developed paraCluster.py to convert texts from clustering corpus to vectors and run K-means from "sklearn.cluster" in python to get clusters. Here I used k = 3 and "tfidf" of terms as the values in vectors. After clustering, I used the labels of each vector to assign each paragraph to its cluster. Then I have three clusters where store the text content of paragraphs. Next, I used lucene to index clusters and finally I have the index file for clusters. Since the index file for clusters is generated offline, the above process will not decrease the speed of searching.

For searching, I used “BM25” of “lucene” in java to compute the similarity between query/clusters and paragraph/clusters to assign cluster to query and generate clusters’ rank for each paragraph. And I used the above algorithm to re-rank.

#### 4.2.3 Evaluation

The evaluation result of this method is almost the same as baseline. As record from program, 65% of the paragraphs are given the new score by the algorithm. But it seems like the new score does not affect the rank. Later we’ll increase the weight of additional score and see if there’s any change.

#### 4.2.4 Conclusion

For k-means cluster, I’m using a 10,000- paragraph-corpus for clustering. I believe the result will has a better improvement by using a larger cluster corpus.

### 4.3 Re-rank by Category Clustering

#### 4.3.1 Introduction

For each entity in Wikipedia, it has a category. We choose the categories which have more than 100 entities to make clusters. And we use the same methodology as K-means cluster to do re-rank for run files.

#### 4.3.2 Implementation

For each paragraph Ids in the “dedup.articles-paragraphs.cbor”, entities were extracted using DBpedia variation. The extracted entities were then parsed into a java program that executes the curl command, which gives the type for each entity. The entities were then clustered according to their respective type.

#### 4.3.3 Evaluation

The evaluation result is almost the same as K-means cluster.

### 4.4 Re-rank by DBpedia Type

#### 4.4.1 Introduction

DBpedia spotlight, is a system that automatically annotate text documents with DBpedia URIs. It gives a wide range of entity and type relationship. This relationship is used in this method to check the relevance of a paragraph.

#### 4.4.2 Implementation

This task uses DBpedia variation and the frequency of its type in each paragraph to re-rank the run file.

- 1) A search method was implemented, “BM25” similarity of “lucene” was used to compute similarity between query and paragraph to generate a baseline run file.
- 2) The paragraph IDs of the baseline run file were compared with the “dedup.articles-paragraphs.cbor” to get the text content.
- 3) For each paragraph ID, entities and its respective type was gathered using DBpedia spotlight.
- 4) The frequency of DBpedia type was clustered for each paragraph ID in the run file.
- 5) The run file was then re-ranked as per the descending order of the frequency of type in a paragraph given a query.

This method was implemented for top 20 paragraph IDs.

#### 4.4.3 Evaluation

For evaluation, we use MAP, Precision@R, and MRR. And BM25 without query expansion is the baseline here. Following is the result of them.

	<a href="#">Precision@R</a>	MAP	MRR
Re-rank by DBpedia Type	0.1016	0.0548	0.3991

#### 4.4.4 Conclusion:

The first method was implemented to showcase the importance of the knowledge base categorization. It was assumed that higher the frequency of the entities in a paragraph, more relevant. The results give almost similar score as baseline method.

### 4.5 Re-Rank by DBpedia Type and BM25 similarity with weight

#### 4.5.1 Implementation

This task uses DBpedia variation for the frequency of its type and and BM25 similarity score in each paragraph to re-rank the run file.

- 1) A search method was implemented, ” BM25” similarity of “lucene” was used to compute similarity between query and paragraph to generate a baseline run file.
- 2) The paragraph IDs of the baseline run file were compared with the “dedup.articles-paragraphs.cbor” to get the text content.
- 3) For each paragraph ID, entities and its respective type was gathered using DBpedia spotlight.
- 4) The frequency of DBpedia type was clustered for each paragraph ID in the run file.
- 5) The run file was then re-ranked as per the descending order of the frequency of type in a paragraph given a query.
- 6) The run file generated was further processed to get a better score were both the rank and BM25 score could be used to get an appropriate ranking.

New score = BM25 similarity score +  $1/(\text{Rank of the para id as per frequency of type})$

- 7) The New score was then used to re-rank the run file.

This method was implemented for top 20 paragraph IDs.

#### 4.5.2 Evaluation

For evaluation, we use MAP, Precision@R, and MRR. And BM25 without query expansion is the baseline here. Following is the result of them.

	<a href="#">Precision@R</a>	MAP	MRR
Re-rank by DBpedia type with BM25 similarity	0.1016	0.0548	0.3991

#### 4.5.3 Conclusion:

The second method was implemented to showcase the importance of the knowledge base categorization considering BM25 score with help of weights. It was assumed that higher the frequency of the entities in a paragraph, more relevant. But at the same time the BM25 score cannot be ignored. So, weights were provided it was re-ranked according to the formula mentioned. This also provides score similar to baseline method. There is no significant improvement but there is no decrease in the measures.

### 4.6 Classifier

For this Prototype, we have considered three different types of Machine Learning classifiers. They are J48, Random Forest, Naïve Bayes. We have used a weka libraries to build our classifiers. Weka supports several standard data mining tasks, more specifically, data preprocessing, clustering, classification, regression, visualization, and feature selection.

#### Naïve Bayes

In machine learning, **naïve Bayes classifiers** are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features. Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set.

#### J48 Classifier

**J48** is an algorithm used to generate a decision tree. J48 builds decision trees from a set of training data, using the concept of information entropy. The training data is a set of already classified samples. Each sample consists of a p-dimensional vector, where the represent attribute values or features of the sample, as well as the class in which falls.

At each node of the tree, J48 chooses the attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. The splitting criterion is the normalized information gain (difference in entropy). The attribute with the highest normalized information gain is chosen to make the decision.

#### Random Forest Classifier

Random Forest are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is

the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

#### 4.6.2 Implementation

Steps includes

1. We have used train v2.0 of the TREC -car dataset to train the classifier.
2. To train the classifier we considered base.train.cbor-paragraphs.cbor of the train data.
3. Here the paragraph is the text and the heading corresponding to the paragraph are the class values.
4. The training set is created.
5. Now feed these training set to the above-mentioned classifiers.
6. It takes lot of time to build the classifier for the training set is generated because of the huge file size. So we had taken 10000 paragraphs and built the classifiers.
7. Now the classifier is trained with the training set of 10000 paragraphs,
8. The base line method of bm25 is run and the results are evaluated.
9. The baseline method is again run and the paragraph retrieved from the method are made into the test set.
10. The test set is supplied to the classifier and classified
11. The results are written to the run file.
12. The performance is not significant because of the small number of paragraphs.
13. The original model with all the paragraph is still running on the server. I have put that in screen mode.
14. I run the classifier on the cluster result generated by peihao.










#### 4.6.2 Results

The evaluation results for the classifier are not as promising as the baseline. While all the classifier takes time to classify each text. I have included the made the J48 classifier run for pages in the main class as it runs faster than Naïve Bayes and Random Forest.

#### 4.6.4 Conclusion and Future work:

I believe the results of the classifier are not as promising. Its because very small set of data I have used. Once the model is fully built I believe the results would be better. For the prototype, I plan to build the model for 100,000 paragraphs on the train v2.0 and run the classify algorithm along with more methods from my team.

#### 5. Contribution

	Nithin	Peihao	Sangeeta
Pseudo Relevance Feedback with Entities			
Re-Rank by K-means Clustering			
Re-rank by Category Clustering			
Re-rank by DBpedia Type			
Re-Rank by DBpedia Type and BM25			
Classifier_J48			
Classifier_Naïve Bayes			
Classifier_Random Forest			
Classifier_J48_Clustering	