MONGOOSE PROTOTYPE 2

Team 3

Prototype submission for partial fulfillment of the requirements of CS980.01

Spring 2018

SUMANTA KASHYAPI
SHUBHAM CHATTERJEE
TARUN PRASAD GANESA PANDIAN
AJESH VIJAYARAGAVAN

CONTENTS

1. INTRODUCTION

1.1. TASK DEFINITION

Given an article stub Q, retrieve for each of its sections H_i, a ranking of relevant passages. The passage is taken from a provided passage corpus.

1.2. METHODS USED

a) Candidate set generation using BM25 and Pseudo-Relevance Feedback with Query expansion using RM3.
b) Clustering: Hierarchical Agglomerative Clustering, K-Means Clustering
c) Mapping paragraphs using *word2vec*
d) PageRank Algorithm
e) Entity Linking
f) Text summarization

2. APPROACH

2.1. CANDIDATE SET GENERATION

For candidate set generation, we explored several variations of the basic methods. W2.e used BM25 and Language Models such as LM-DS and LM-JM to retrieve top 100 paragraphs for each page title and then applied query expansion using two methods: RM3 and KNN. We experimented with variations of each of these and found that none of these could beat the basic BM25 model in performance (measured in terms of MAP). The results for each variation experimented with is shown in table-1. The BM25, LM-DS, LM-JM and RM3 models are standard information retrieval models and hence we refrain from further discussing these models. However, a description of how KNN was applied for the task of query expansion follows.

2.1.1. Query Expansion using KNN

Let the given query Q be$\{q_1, \dots, q_m\}$. In this approach, we define the set C of candidate expansion terms as

$$C = \bigcup_{q \in Q} \boldsymbol{NN}(q)$$

where $\boldsymbol{NN}(q)$ is the set of K terms that are nearest to q in the embedding space. We use *GloVe* to obtain the word vectors for each word. For each candidate expansion term t in C, we compute the mean cosine similarity between t and all the terms in Q using the equation:

$$Sim(t, Q) = \frac{1}{|Q|} \sum_{q_i \in Q} \boldsymbol{t}.\boldsymbol{q_i}$$

The terms in C are sorted by this mean score and the top K candidates are chosen as the actual expansion terms. We use a set of pseudo-relevant documents (PRD) that are retrieved at top ranks in response to

the initial query to restrict the search domain for the candidate expansion terms. Instead of searching for nearest neighbor within the entire vocabulary of the document collection, we consider only those terms that occur within PRD. The size of PRD may be varied as a parameter.

2.1.2. Query Expansion using ConceptNet

Each query is expanded using additional terms from ConceptNet for those terms in the query. ConceptNet is a knowledge graph of terms with a reasoning of how one term is connected to additional terms in RDF format which is retrieved as JSON object and value of labels from JSON array. Ranking of paragraphs with score for each query is retrieved that is evaluable with top level section ground truth qrel file.

Pseudocode:

for each page:

for each top level section:

retrieve top level section term from query string

execute api url with concatenated term and retrieve response to JSON object conversion and retrieve related terms from JSON Array

Add all retrieved terms and expand query

Pass expanded query to query builder and retrieve top documents.

2.1.3. Combining PageRank scores with other methods

We obtained a candidate set using BM25 using page titles as queries and applied the PageRank algorithm to the graph of paragraphs obtained. The graph is defined as a set of nodes (which consist of paragraphs) and a set of edges (entities shared by the paragraphs). An edge is added between two nodes, if the paragraphs representing the nodes have at least one entity in common. This score for each paragraph is combined with the score obtained by other methods as a weighted score to obtain the final score as follows:

$$final\ score = (\ 0.8 * score1\ ) + (\ 0.2 * score2\ )$$

where $score1$ is the score obtained from some other method and $score2$ is the PageRank score. See tables 1 and 2 in section 3.6 for results.

2.2. CLUSTERING

The different modules in the clustering pipeline are described below.

2.2.1. Candidate set generation

This module uses the techniques describes in Section 2.1 above to generate a candidate set on which clustering methods can be applied. By default, it uses BM25 and query expansion with RM3 and PRF.

2.2.2. Calculation of similarity scores of paragraph pairs

For each page i, get the set of retrieved paragraphs ret_i and calculate similarity scores between all paragraph pairs. Formally, calculate $score(p_x, p_y)$ where $(p_x, p_y) \in ret_i;\ p_x \neq p_y$. Two methods have been used to calculate this similarity score:

a) Similarity based on WordNet: We calculate similarity of a pair of paragraphs as the mean similarity score of every pair of word in the paragraphs. Four similarity scoring methods have been used:

- **i)** Jiang-Conrath Similarity
- **ii)** Wu-Palmer Similarity
- **iii)** Path Similarity
- **iv)** Lin Similarity

The mean similarity between two paragraphs obtained using each of the above similarity measures is used as a feature for a feature vector. These vectors are used with *RankLib* to get an optimized weight vector (optimized for MAP using k-fold cross-validation). This weight vector is finally used to calculate the similarity as the dot product of the feature vector and the weight vector. These similarity vectors are used with clustering.

b) Similarity based on Word2Vec: We obtain vector representations of each paragraph as follows:

- **i)** We sort the terms in each paragraph based on the TF-IDF values.
- **ii)** The top 10 terms are used as representative terms for each paragraph.
- **iii)** The vector representation of a paragraph is the average vector of the word vectors of each of the terms obtained in step (ii) above.

We use *GloVe* pre-trained vectors in the above computation.

2.2.3. Clustering

Two famous clustering techniques have been experimented with:

a) Hierarchical Agglomerative Clustering: Using the distance formula between two paragraph pairs (p_x, p_y) , $d(p_x, p_y) = score(p_x, p_y)$,where $score(p_x, p_y)$ is obtained using methods described in 2.2.2 above, we apply HAC to get clusters of paragraphs.

b) K-Means: We use word vectors (obtained from GloVe) to represent documents and apply K-Means on them.

2.2.4. Mapping clusters to sections

We map the clusters to the corresponding sections as follows:

i) We define a 'cluster vector', that is, a vector representing the paragraphs in the cluster, as the average vector of each of the paragraphs in the cluster.

ii) We define a 'query vector', that is, a vector representing a query (top-level section names), as the average vector of each word in the query.

iii) We rank the clusters for each query based on the dot product of the cluster vector and the query vector.

2.2.5. Ranking paragraphs in each cluster using PageRank

The method takes in clusters of paragraphs, and ranks the paragraphs inside the clusters using the PageRank algorithm. Using the candidate input set, the clustering algorithms produce a set of clusters within each page, as described in Section 2.2 above. Based on the clustering algorithm used, we get different number of clusters inside each page. This is the input to the PageRank algorithm. The input is structured as follows: we have multiple pages identifiable by their page ID. Each of these pages have single/multiple clusters. Each of the clusters have single/multiple paragraphs, identifiable by their paragraph ID.

The paragraphs within the clusters are not ranked by any order. We create a graph of these paragraphs, using the paragraphs as nodes. The edges between them are derived using the two variants described below. Each variant produces a different graph structure, leading to different ranking of the paragraphs inside the cluster.

a) DBPedia Spotlight

This process is done for every cluster inside the page. Using the paragraph ID, the paragraph text is extracted from the Lucene Index, which is passed as the input to DBPedia Spotlight. The output from DBPedia Spotlight is a list of entities found in the paragraph. This is stored, and the process is repeated for all the paragraphs inside the cluster. The list of paragraphs and their entities are analyzed. Paragraphs having same entities are linked with each other.

b) Common Words

This process is also done for every cluster of paragraphs inside the page. The words from each of the paragraphs are extracted, and stored along with their paragraph IDs. The number of common words between each of the paragraphs is computed, and stored in a matrix. The average number of common words between paragraphs in the cluster is calculated. The paragraph pairs which have a higher number of common words than this average value, are linked with each other.

Using this graph structure, we apply the PageRank algorithm. The score for each of the paragraph is taken from the PageRank vector, which is also used in ranking these paragraphs. A pseudocode describing the method is given below:

```
for each page :
      for each cluster :
            for each paragraph :
                  extract entities
                  extract words
            build graph with common entities
            build graph with common words
            apply page rank on both graphs
```

```
return pages-of-ranked-clusters
```

2.3. RANKING USING WORD EMBEDDING

We use word embeddings to rank paragraphs for each query as follows:

2.3.1. Candidate set generation

This module uses the techniques describes in Section 2.1 above to generate a candidate set on which clustering methods can be applied.

2.3.2. Ranking

The vector representing each paragraph p_i is calculated as described in 2.2.2(b) above. Then ranking is created using the cosine similarity between top-level section vector and paragraph vectors as the rank score.

2.4. TEXT SUMMARIZATION

2.4.1. Motivation

The motivation behind this comes from the insight that section headings and paragraphs are not generated from the same model. Section headings are constructed in such a way that it best captures the prevalent ideas expressed in the following paragraphs in a concise manner whereas paragraphs are more elaborated on the section topic. Hence, for the passage retrieval task, there is little chance of success if we try to directly map section headings to paragraph texts. In this pipeline, we try to summarize the paragraph and use that summary to map to section headings.

2.4.2. Candidate set generation

This module uses the techniques describes in Section 2.1 above to generate a candidate set on which clustering methods can be applied. By default, it uses BM25 and query expansion with RM3 and PRF.

2.4.3. Generating paragraph summary

There are various summarization techniques. However, due to time constraints, we used a simple technique to generate summary of each paragraph. Typically, the words that best summarize the whole paragraph have high TF-IDF value. Hence, we sorted tokens in each paragraph by TF-IDF scores and took the top 5 terms as the summary of the paragraph. This gives us a strong baseline that we intend to beat using some more in-depth summarization techniques in our next submission.

2.4.4. Mapping paragraphs to sections

We ranked paragraphs for each top-level sections based on similarity between the section heading and paragraph summary using *word2vec* and *WordNet* similarity.

2.5. SHORTEST PATH ON CAR GRAPH

All the unique paragraphs ids are stored in list from the candidate set generated using LM-JM method. For each unique paragraph id, text is retrieved using entity linking and anchor texts are validated by passing text and retrieving annotation from DBpedia Spotlight. A graph is formed using primary paragraphs as nodes and connected paragraphs as edges. Applying breadth first search on the above unweighted graph, shortest path from one node to reach to every other edge is found using FIFO queue.

Pseudo code:

```
Initialize all vertices as undiscovered;

Mark 1st node as discovered and add in FIFO queue (Q.enqueue (Start
node));

While (Q is not empty):

    Q.dequeue();
        For each outgoing edges;
            If (the edge is not discovered):
                Discovered = TRUE;
                Enqueue it(Q.enqueue(edge));
```

From the above algorithm, parent node at each level is taken as 1st level and immediate edges it discovered are at 2nd level. Paragraph ids at each level are clustered to allocate score and the score a paragraph received is added and reranked for top level sections evaluable with top level qrels ground truth.

3. EVALUATION

3.1. DATA SET

We use the ground truth files from the benchmarkY1-train for evaluation and the page-level and top-level section level training files for k-fold cross validation and training our methods.

3.2. CORPUS

The *paragraphCorpus* is used as the corpus.

3.3. QUERIES

The top-level section names are used as queries for top-level sections whereas the page names are used as queries for articles.

3.4. GROUND TRUTH

The files *train.pages.cbor-toplevel.qrels* is used as the ground truth for top-level section name queries and *train.pages.cbor-article.qrels* is used as the ground truth for page level queries.

3.5. EVALUATION MEASURES USED

We use the following three evaluation measures:

- Mean Average Precision (MAP)
- Precision at R (RPrec)
- Reciprocal rank (recip_rank)

3.6. RESULTS

3.6.1. Evaluation results for candidate set generation

Variation	MAP	Rprec	recip_rank
BM25+RM3+PRF	0.0262	0.0186	0.0457
BM25+KNN+PRF	0.0290	0.0203	0.0512
LM-DS+RM3+PRF	0.0306	0.0239	0.0527
LM-DS+KNN+PRF	0.0228	0.0178	0.0408
LM-JM+RM3+PRF	0.0237	0.0172	0.0415
LM-JM+KNN+PRF	0.0272	0.0220	0.0470
LM-JM+ConceptNet	0.0039	0.0035	0.0042
BM25+RM3+PRF+PR	0.0004	0.0001	0.0008
BM25+KNN+PRF+PR	0.0007	0.0008	0.0018
LM-DS+RM3+PRF+PR	0.0005	0.0001	0.0013
LM-DS+KNN+PRF+PR	0.0004	0.0000	0.0011
LM-JM+RM3+PRF+PR	0.0019	0.0018	0.0036
LM-JM+KNN+PRF+PR	0.0006	0.0000	0.0015

Table 1: Top-level section names as queries

Variation	MAP	Rprec	recip_rank
BM25+RM3+PRF	0.0651	0.1221	0.2711
BM25+KNN+PRF	0.0727	0.1222	0.3904
LM-DS+RM3+PRF	0.0662	0.1168	0.3891
LM-DS+KNN+PRF	0.0569	0.0988	0.3092
LM-JM+RM3+PRF	0.0463	0.0900	0.1982
LM-JM+KNN+PRF	0.0566	0.0942	0.3161
BM25+RM3+PRF+PR	0.0016	0.0032	0.0077
BM25+KNN+PRF+PR	0.0022	0.0073	0.0334
LM-DS+RM3+PRF+PR	0.0017	0.0036	0.0136
LM-DS+KNN+PRF+PR	0.0018	0.0040	0.0229
LM-JM+RM3+PRF+PR	0.0031	0.0063	0.0202
LM-JM+KNN+PRF+PR	0.0013	0.0049	0.0160

Table 2: Page names as queries

3.6.2. Evaluation results for clustering

Method	Using candidate set (BM25 + RM3)		Using ground truth (article.qrels)	
	Adjusted RAND	F Measure	Adjusted RAND	F Measure
HAC + WordNet Similarity	0.0106	0.1779	0.0215	0.3252
HAC + Word2Vec Similarity	0.0017	0.0324	0.0088	0.2223
K-Means + Word2Vec Similarity	0.0384	0.1469	0.1320	0.2524

Table 3: Clustering results

Method	Using candidate set (BM25 + RM3)	Using ground truth (article.qrels)
HAC + WordNet Similarity	0.0100	0.0893
HAC + Word2Vec Similarity	0.0089	0.1554
K-Means + Word2Vec Similarity	0.0062	0.1472

Table 4: Clustering results for mapping clusters to sections (using MAP)

3.6.3. Evaluation results for text summarization

4. Variation	MAP	Rprec	recip_rank
TF-IDF Summary with word2vec	0.0181	0.0171	0.0546
TF-IDF summary with WordNet	0.0176	0.0168	0.0470
TFIDF Summary with WordNet (QE candidate set using BM25+RM3)	0.0201	0.0199	0.0490

Table 5: Text summarization

3.6.4. Evaluation results for ranking within clusters

Method	Using DBpedia			Using common words		
	MAP	Rprec	recip_rank	MAP	Rprec	recip_rank
HAC + WordNet Similarity	0.0091	0.0072	0.0272	0.0096	0.0058	0.0306
HAC + Word2Vec Similarity	0.0131	0.0093	0.0352	0.0086	0.0058	0.0212
K-Means + Word2Vec Similarity	0.0123	0.0076	0.0345	0.0072	0.0059	0.0211

Table 6: Ranking within clusters

3.6.5. Evaluation results for running prototype on *benchmarkY1-train* after combining with RankLib

- MAP = 0.0270
- Rprec = 0.0233
- Recip_rank = 0.0583

4. CONCLUSION AND FUTURE SCOPE

We found that our candidate set generation is not satisfactory. If the candidate set itself is not of high quality, the other methods cannot hope to achieve better results. The best candidate set generated for page titles had a MAP of 0.0727 (for BM25+KNN+PRF) whereas the best candidate set generated for top-level section names had a MAP of 0.0306 (for LM-DS+RM3+PRF). Hence, we need to improve out candidate sets. In the third prototype, this would be our primary goal.

We also found that combining static query-independent PageRank scores with the other methods actually leads to a substantial decrease in performance. As described in section 2.1.2, the combined score is obtained as a weighted sum of scores. The choice of the weights is arbitrary and intuitive. In some ways, we think that since the PageRank score of each paragraph is query-independent, it does not really reflect the relevance of the paragraph for the query. Rather, it is a static quality measure. Hence, the contribution of PageRank to the final score should be less. Of course, this does not have any base in mathematics and the entire problem of choosing correct weights so as to optimize the results falls into a class of mathematical optimization problems called Multi-Objective Optimization. Since, our implementation of PageRank really does not serve as a measure of relevance of a document for a query, we did not spend much time on solving this Multi-Objective Optimization problem. We also experimented with using a harmonic mean of the scores as the final scores but this too did not yield any better results since the scores themselves are static quality measures. We do not include those results here. In the next prototype, we have plan to improve on this in the following ways:

i) Implement a query-dependent PageRank algorithm
ii) Choose the weights for the combination accurately

We have carried out a separate experiment where we found how many true paragraph pairs are present in the candidate set (recall). For candidate set generated using BM25, there are 259 top-level sections out of 859 for which at least 1 paragraph pair (2 paragraphs that should go together) is retrieved. There are 27 pages out of 117, for which not a single pair of paragraphs is retrieved. This clearly shows that unless we have a significantly better candidate set, our clustering techniques are not able to improve the results.

K-Means algorithm works with feature vectors which corresponds to each paragraph. But WordNet similarity measure gives us a score that corresponds to a paragraph pair. We need to find a metric that will convert pairwise measure to individual one so that we can have a variation with K-Means and WordNet similarity.

APPENDIX A

Installation Instructions

Follow the instructions given below to install and run the prototype:

Step-1: Clone the project from GitHub

```
git clone https://github.com/nihilistsumo/Mongoose.git
```

Step-2: Change to Mongoose directory

```
cd Mongoose
```

Step-3: Change permission of the script run.sh

```
chmod 755 run.sh
```

Step-4: Run the installation script

```
./run.sh
```

Step-5: Run the prototype

```
java -jar target/Mongoose-0.0.1-SNAPSHOT-jar-with-dependencies.jar
```

You can specify the inputs such as the location of the index directory, location of output directory, name of output file, number of threads, etc. in the file ***project.properties***. However, some default values have already been added and you should be able to run the prototype without worrying about specifying the settings yourself. We already have the index and other required files on the server and running the script provided would do the work for you. However, you can change the values in the file if you so wish.

APPENDIX B

Individual Contribution

- **Sumanta Kashyapi**: Clustering, Text summarization, merging code, RankLib
- **Shubham Chatterjee**: Candidate set generation, Pseudo-Relevance Feedback, Query Expansion using RM3 and KNN, PageRank for paragraphs, document preparation
- **Tarun Prasad Ganesa Pandian**: PageRank for clusters using DBpedia spotlight and common words
- **Ajesh Vijayaragavan**: Candidate set generation using LM-JM, Query expansion using concept net, shortest path with clustering