Prototype Report : 2

Team No:1
Github Repository: https://github.com/ba1044/DataScience_FinalProject
Prototype2 Release: https://github.com/ba1044/DataScience_FinalProject/releases/tag/Prototype2

Benchmark Test Runfiles:

While their locations are mentioned in the GitHub repository, just to be clear about what we want evaluated:

results_prototype2/public_test/combined_method.run.gz
result_prototype2/public_test/sdm_section.run.gz

If possible, we would like our third runfile to be the one we generated from Prototype 1, located at:
results/benchmark_kY1_public_query.run.gz

If this is not allowed, then we would like the third runfile to be this:
results_prototype2/public_test/alternative_to_prototype_1_run/sdm_run.run.gz

Method Description:

The following are brief descriptions of the ranklib_query methods. For a more detailed description, please see the repository's readme (located at the bottom of the readme).

- **average_abstract:** Query is searched against a Lucene index containing entities and their abstract (first three paragraphs of page) using BM25. A paragraph's score is expressed as the average score of the linked entities (Spotlight), where the score is derived from the query against the abstract Lucene index.

- **hyperlink:** The allButBenchmark paragraph corpus was parsed to create an entity mention model from the anchor text and linked entities (similar to the method in the entity linking homework). This is used to score the likelihood of an entity given the query terms (they are treated as entity mentions). A paragraph's score is equal to the average likelihood of its annotated entities (using Spotlight) given the query.

- **sdm:** This is my attempt at a sequential dependence model. The weights for each feature and the alpha parameter were trained using RankLib (see KotlinRanklibTrainer).

- **abstract_sdm:** As sdm method, except it is language model is based on entity abstracts, and paragraph are scored according to the likelihood of their annotated entities given this model.

- **sdm_expansion:** This uses Kevin's entity query expansion method to expand each query term with the top k entities (from the abstract Lucene index) such that the abstract's entities (linked via Spotlight) are added as new terms to the query. These expanded queries are then used according to the sdm method.

- **sdm_section:** The sdm method was used to score each section independently, and weights on these sections were learned using RankLib.

- **nat_sdm:** Using Kevin's natural language processor methods, nouns and verbs are extract from the query string and the paragraph text, and a separate SDM model is created for the nouns and verbs, with the final score being averaged.

- **string_similarity_section:** A paragraph's score is expressed as the average similarity from the query terms to the paragraph's annotated entities (using Spotlight). A weighted combination of the Jaccard, Jaro Winkler, Normalized Levenshtein, and Sorensen Dice Coefficient string similarities was learned for this method. The feature was then used to score each section separately, and new weights were learned using RankLib.

- **tfidf_section:** Using Bindu's Tfidf method, a document's score is expressed as the average tfidf score of each of its terms to that of the query's terms. This feature was then used to score each section and a new feature was created based on learning the best weighted combination of these scores.

- **combined:** This is a weighted combination of my ***sdm*** method, a simple ***weighted section BM25*** feature I didn't count as a method, my ***string_similarity_section*** method, and Lucene's ***LMDirichletSimilarity*** score.

- **super_awesome_teamwork**: This if a weighted combination of the methods I created using other team member's methods (**tfidf_section, sdm_expansion, nat_sdm**) and **without adding bm25** as an additional feature (to see if these methods can hold up on their own when combined).

****

- **Query Ranking by Entity linking using DBPedia Spotlight:** In this method I am trying to rank the Query based on annotated Entity of paragraph content and then paragraph's score is being calculated as the average score of the annotated entities using spotlight where the score is derived from the query against the abstract index. Here we can use both type of Query that page query and section path query.

- **Paragraph text with TF-IDF similarity:** In order to rank the Query based on Paragraph score, using TF_IDF lnc.ltc variant. adding log is to dampen the importance of term that has a high frequency. I add 1 to the log(tf) because when tf is equal to 1, the log 1 is

zero. by adding one, I can distinguish between tf=0 and tf=1 also for normalization using Cosine normalization and Ranked the query based on paragraph score.

- **Paragraph text with Wordnet:** To rank the paragraph text using Wordnet API. Which gives similarity score. For Wordnet implementation I am using extjwnl (Extended Java WordNet Library) is a Java API for creating, reading and updating dictionaries in Wordnet format.

- **Paragraph with just entity:** In this method I am trying to rank the paragraph based on entity present in paragraph only. Then counting the total number of entities present in the paragraph and scoring accordingly. The difference between first one and this is that It does not check the entity content.

****

- **Query Entities Relevance Model + Query Expansion**
    - **Page Queries**: Use Lucene to retrieve abstract text of entities in query (Created by Jordan), and use Spotlight API to annotate the abstract to get entities relevant to query. Then rank the entities to expand the query, finally run expanded query against index using BM25.
    - **Section Path Queries**: Divide the section path into two/ three part, the entities of top level and bottom level of the section path will rank higher than the middle. For example, Section Path Query:“Micheal Jordan / Professional career / Second three-peat (1995–1998)”, the entities in “Micheal Jordan” and “Second three-peat(1995-1998)” will rank higher than ones in the middle. And based on entities ranking, create a new expanded query to run against BM25.

- **Document Entities Relevance Model + Query Expansion**: Predict relevant entities through entity linking paragraphs of the feedback run using BM25, we have a entities field in index called “spotlight”. Then expand query with top 5 entities, run against paragraph index using BM25. Create run file for:
    - **Page Queries**
    - **Section Path Queries**

Contributions:

Jordan: Other than methods: making sure merges go smoothly, fixing master branch when it breaks, organizing files required by team and making sure it’s accessible on the server, testing project on the server to make sure everything works, creating run.sh to

automatically run teammate's methods and store results for convenience, lots and lots of documentation and organization.

Kevin: Except ranking method, implement DBpedia Spotlight API to annotate a paragraph of text and return a list of entities and information associated with these entities. Implement Natural Language Processing method (Stanford NLP), this will be extended more features for Prototype 3. Implement Log4j for logging system. And debugging on project and index.

Bindu: Implemented four method (Query Ranking by Entity linking using DBPedia Spotlight), Paragraph text with TF-IDF similarity, Paragraph text with Wordnet and Paragraph with just entity). Created graph for easy visualization for all methods which is being implemented by my teammates.

Teamwork:

Jordan: Created lucene index of entities and their associated abstracts (also used by Kevin in some of his methods). Made use of Kevin's entity query expansion method and natural language processing methods in two of my methods. Used Bindu's Tfidf similarity with Paragraph method in one of my methods.

Bindu : Using lucene index of entities and their associated method which was created by Jordan and used Kevin variation package method which was useful for my Entity Similarity with Dbpedia.

Kevin: Using lucene index of entities and their associated abstracts (Abstract index and "spotlight" field in paragraph index), create query expansion method, Spotlight API method and natural language processing method for Jordan and Bindu.

Evaluation Results on BechmarkY1train:

Method	MAP	MRR	Rprec	Author
average_abstract	0.1041	0.1487	0.0773	Jordan
hyperlink	0.1052	0.1499	0.0788	Jordan
sdm	0.1199	0.1661	0.0947	Jordan
abstract_sdm	0.1309	0.1488	0.0774	Jordan

sdm_expansion	0.1037	0.1480	0.0773	Jordan
sdm_section	0.1323	0.1805	0.1093	Jordan
nat_sdm	0.1097	0.1540	0.0818	Jordan
string_similarity_section	0.1047	0.1491	0.0775	Jordan
tfidf_section	0.1041	0.1482	0.0778	Jordan
combined	0.1303	0.1789	0.1033	Jordan
super_awesome_team work	0.0838	0.1190	0.0607	Jordan
Query Ranking by Entity linking using DBPedia Spotlight	0.05	0.1059	0.3036	Bindu
Paragraph text with TF-IDF similarity	0.0044	0.0377	0.0105	Bindu
Paragraph text with Wordnet	0.0822	0.1393	0.4054	Bindu
Paragraph with entity	0.0043	0.0035	0.0351	Bindu
Query RM + Query Expansion on pages	0.0674	0.3310	0.1167	Kevin
Query RM + Query Expansion on section path	0.0635	0.0918	0.0449	Kevin
Document RM + Query Expansion on pages	0.0547	0.3036	0.1059	Kevin
Document RM + Query Expansion on section path	0.0638	0.0911	0.0446	Kevin

The Standard error graph can be viewed here.
https://github.com/ba1044/DataScience_FinalProject/blob/master/results_prototype_2/Allgraph.xlsx

Required Files:

Files required for prototype_2 are hosted in the /trec_data folder on the server (you may also generate the files by following index, hyperlink_indexer, gram_indexer, and abstract_indexer methods). The following are descriptions and the locations of these files:

Precompiled prototype 2 jar
/trec_data/team_1/program.jar

Location to RankLib jar (used by feature_selection method)
/trec_data/RankLib-2.1-patched.jar

Location to a compiled copy of trec_eval (used by run.sh)
/trec_data/trec_eval

Location to benchmark cbor and qrel files (used by many methods)
/trec_data/benchmarkY1-train

Location to paragraph corpus (used by index command to create Lucene index)
/trec_data/paragraphCorpus/dedup.articles-paragraphs.cbor

Location to unprocessedAllButBenchmark (used by hyperlink_indexer and abstract_indexer)
/trec_data/unprocessedAllButBenchmark/unprocessedAllButBenchmark.cbor

Premade abstract Lucene index (made with abstract_indexer command)
/trec_data/team_1/abstract

Premade gram Lucene index (made with gram_indexer command)
/trec_data/team_1/gram

Premade Lucene index for Paragraph Corpus (made with index command)
/trec_data/team_1/myindex

Spotlight jar and required models
/trec_data/team_1/spotlight_server

Premade entity mentions database (made with hyperlink_indexer command)
/trec_data/team_1/entity_mentions.db