**Name: Abuzar Khan**

**Andrew ID: abuzark**

**Machine Learning for Text Mining**

**Homework 1 - Template**

1. **Statement of Assurance**
2. **Experiments**
3. Describe the custom weighing scheme that you have implemented. Explain your motivation for creating this weighting scheme.
4. Report of the performance of the 9 approaches.

I. Metric: MAP

|  |  |  |  |
| --- | --- | --- | --- |
| Method \ Weighting Scheme | NS | WS | CM |
| GPR |  |  |  |
| QTSPR |  |  |  |
| PTSPR |  |  |  |

II. Metric: Precision at 11 standard recall levels  
(Use one table for each recall level, so totally there would be 11 tables.)

|  |  |  |  |
| --- | --- | --- | --- |
| Method \ Weighting Scheme | NS | WS | CM |
| GPR |  |  |  |
| QTSPR |  |  |  |
| PTSPR |  |  |  |

III. Metric: Wall-clock running time in seconds

|  |  |  |  |
| --- | --- | --- | --- |
| Method \ Weighting Scheme | NS | WS | CM |
| GPR |  |  |  |
| QTSPR |  |  |  |
| PTSPR |  |  |  |

IV. Parameters

1. Compare these 9 approaches based on the various metrics described above.
2. Analyze these various algorithms, parameters, and discuss your general observations about using PageRank algorithms.
3. 1. What could be some novel ways for search engines to estimate whether a query can benefit from personalization?
   * + 1. If they observe that the order of visited pages by the user differs from one user to another for the same query, then this might indicate that there is some benefit from personalizing for each user.
       2. If they observe that order of visited pages differs from user to user but largely based on the previous searches (search context), then there might be some benefit from personalization based on recent searches.
       3. Similarly, regional personalization might be helpful if such differences are observed when aggregating region-level user behavior. For example, searching for CMU in Pittsburgh would refer to Carnegie Mellon, but perhaps in Michigan would refer to Central Michigan University.
       4. In general, observing a low correlation between documents’ relevance to query and the documents that the users end up going for would mean that personalization can be helpful.

2. What could be some novel ways of identifying the user’s interests (e.g. the user’s topical interest distribution Pr(t|u)) in general?

* + - 1. One possible way is to randomly inject pages with a certain topic distribution, and check for which random pages the users visit regardless of the relevance to the query they write. The ones that are often visited would indicate that the user is interested in that topic.
      2. Another way is to use the preview text to show passages that have topic distributions skewed towards a randomly chosen topic. We can then measure the difference in the frequencies of the user exploring some topics over others.

1. **Details of the software implementation**
2. Describe your design decisions and high-level software architecture;

The directory structure of the **src/** folder is given below.

src/

├── config.py

├── data

│   ├── doc\_topics.txt

│   ├── indri-lists/

│   ├── query-topic-distro.txt

│   ├── transition.txt

│   └── user-topic-distro.txt

├── hw1.sh

├── main.py

├── models.py

├── output/

├── requirements.txt

├── retrieval.py

└── utils.py

The goal of the system was to make as much of it reusable as possible and increasing modularity so that functionality is not repeated in code. The overall file structure is as follows:

* + - 1. **config.py** This file contains some useful constants for the system. Some variables here may need changing based if the corpus is changed, such as total number of documents and total number of topics.
      2. **models.py** This file contains the definitions of pageRank algorithm classes.
         1. It first outlines the Abstract class *Ranker* that contains common initializations, functions that step through and run power iteration till convergence, and definitions of abstract functions.
         2. Next, we have pageRanks (GPR) and pageRanksPersonalized (TSPR) classes that inherit from Ranker and define class functions like initialize algorithm and getRanks that are different for the two algorithms.
         3. Lastly, we have the *getRanker()* function that returns the appropriate ranker object based on arguments passed into the system.
      3. **retrieval.py** contains the three retrieval functions (NS, WS, and CS) along with a function that will return the right function as per arguments of the system.
      4. **utils.py** contains utility functions for file I/O and parsing command line arguments.
      5. **main.py** runs the specified algorithm as per arguments.

The arguments of the system are as follows with relevant help statements. These were maintained using the **argparse** library.

usage: main.py [-h] [--transition TRANSITION] [--load\_saved\_matrix] [--queryTopics QUERYTOPICS] [--docTopics DOCTOPICS] [--userTopics USERTOPICS] [--debug] [--algo ALGO] [--seed SEED] [--alpha ALPHA] [--beta BETA] [--gamma GAMMA] [--scorer SCORER] [--output]

optional arguments:

--transition - path to transition matrix

--load\_saved\_matrix - If set, load the saved normalized transition matrix

--queryTopics - path to query topic distribution matrix

--docTopics - path to document topic matrix

--userTopics - path to user-topic distribution matrix

--debug - Set debug mode

--algo ALGO - One of [GPR, QTSPR, PTSPR, all]

--seed SEED - Set random seed

--alpha ALPHA - Alpha parameter (dampening factor for transition Matrix

--beta BETA - Beta parameter (dampening factor for topic-based probability vector, p\_t)

--gamma GAMMA - Gamma parameter (dampening factor for initial p\_0 vector

--scorer SCORER - default “all”. To change, use NS or WS or CS

--no\_op - Set to BLOCK creation of output files for trec\_eval.

1. Describe major data structures and any other data structures you used for speeding up the computation of PageRank;

The code uses the following data structures for ease of programming as well as efficiency:

* + - 1. **Hash-Tables / Dictionaries:** 
         1. To speed up lookups for probability distribution vectors
         2. To standardize fetching topic distributions and relevant document relevance scores for particular user-query pair
         3. To store all information required to produce
      2. **Scipy.sparse.csr\_matrix** to speed up sparse matrix calculations during power iteration
      3. **Scipy.sparse.lil\_matrix** to vastly speed up normalization of transition matrix to handle rows or all-zeros.

1. Describe any programming tools or libraries and programming environment used;

Libraries:

* + - 1. scipy.sparse
      2. numpy
      3. time

Environment:

* + - 1. Python 3.9.16 (main, Jan 11 2023, 10:02:19)
      2. [Clang 14.0.6 ] :: Anaconda, Inc. on darwin

1. Describe strengths and weaknesses of your design, and any problems that your system encountered