Name: Yan Luo
Andrew ID: yanluo

Machine Learning for Text and Graph-based Mining

Homework 1 - Template

1. Statement of Assurance

I hereby certify that all of the work I am submitting is original, created solely by myself, and has not been copied from or written in collaboration with any other source unless otherwise explicitly cited and credited.

2. Experiments

a) Describe the custom weighing scheme that you have implemented. Explain your motivation for creating this weighting scheme.

The formula used is:

$$NewScore = \alpha \cdot \log(IPV[Index, 0]) + \beta \cdot Score$$

Where:

- $IPV[Index, 0]$ is the PageRank score of the document,
- Score is the search-relevance score for the document,
- α and β are the weights assigned to the log of the PageRank score and the search-relevance score, respectively.

The motivation behind this weighting scheme is to combine the benefits of both the PageRank algorithm and the search-relevance score. Since the search-relevance score is negative, while the PageRank score that I calculated is positive that are very large. When calculating with custom weighing schema, I give search-relevance score a weight of 0.9 and the log function on PageRank score with a weight of 0.1. The use of the logarithm of the PageRank score helps to mitigate the effect of very high PageRank scores and put more weight on search-relevance scores.

b) Report of the performance of the 9 approaches.

1. Metric: MAP

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0457	0.2636	0.2569
QTSPR	0.0495	0.2635	0.2489
PTSPR	0.0482	0.2636	0.2545

2. Metric: Precision at 11 standard recall levels

(Use one table for each recall level, so totally there would be 11 tables.)

ircl_prn.0.00

Method \ Weighting Scheme	NS	WS	CM
GPR	0.1446	0.8405	0.8215
QTSPR	0.2004	0.8405	0.8148
PTSPR	0.1793	0.8405	0.8386

ircl_prn.0.10

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0875	0.5926	0.5790
QTSPR	0.0903	0.5926	0.5769
PTSPR	0.0854	0.5926	0.5732

ircl_prn.0.20

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0786	0.4732	0.4610
QTSPR	0.0763	0.4732	0.4663
PTSPR	0.0784	0.4732	0.4577

ircl_prn.0.30

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0737	0.3781	0.3709
QTSPR	0.0707	0.3781	0.3573
PTSPR	0.0756	0.3781	0.3733

ircl_prn.0.40

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0699	0.3145	0.3015
QTSPR	0.0661	0.3145	0.2896
PTSPR	0.0673	0.3145	0.3051

ircl_prn.0.50

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0653	0.2430	0.2347
QTSPR	0.0617	0.2426	0.2193
PTSPR	0.0625	0.2426	0.2258

ircl_prn.0.60

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0534	0.1677	0.1646
QTSPR	0.0503	0.1673	0.1515
PTSPR	0.0501	0.1673	0.1520

ircl_prn.0.70

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0300	0.0914	0.0915
QTSPR	0.0274	0.0915	0.0831
PTSPR	0.0271	0.0915	0.0896

ircl_prn.0.80

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0115	0.0550	0.0618
QTSPR	0.0112	0.0550	0.0513
PTSPR	0.0113	0.0550	0.0595

ircl_prn.0.90

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0074	0.0388	0.0382
QTSPR	0.0072	0.0388	0.0351
PTSPR	0.0075	0.0388	0.0361

ircl_prn.1.00

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0041	0.0101	0.0117
QTSPR	0.0040	0.0101	0.0080
PTSPR	0.0043	0.0101	0.0113

3. Metric: Wall-clock running time in seconds

Retrieval Time

Method \ Weighting Scheme	NS	WS	CM
GPR	0.8169	1.0083	1.1780
QTSPR	0.7661	0.7126	0.7570
PTSPR	0.7316	0.7520	0.7228

PageRank Time

Method	
GPR	0.2236
QTSPR	2.6368
PTSPR	2.6377

4. Parameters

- GPR Algorithm: The GPR function computes the Generalized PageRank vector using the power method. The PageRank algorithm is modified to include a damping vector d which allows a random surfer to jump to any node, not just the dangling nodes.
- Damping Factor: I am using alpha=0.2 as the damping factor for the GPR algorithm.
- Consolidate Indri Lists: The consolidate_indri_lists function consolidates multiple Indri list files into a single file.
- TSPR Algorithm: I am using 0.8 as the damping factor for the TSPR algorithm. The get_tspr function is used to compute the Topic-Sensitive PageRank for a specific user and query.
- Weighing Factor for Combining Scores: In the create_tree_eval_file_for_gpr and create_tree_eval_file_for_tspr functions, the new score is computed by combining the original score and the PageRank score. The weights for combining the scores are specified by the alpha and beta parameters. For example, if alpha=0.5 and beta=0.5, the new score will be the average of the log of the PageRank score and the original score.
- Other Parameters: The max_iter and tol parameters in the GPR function are used to control the convergence of the algorithm. The max_iter parameter specifies the maximum number of iterations, and the tol parameter specifies the tolerance for

convergence. If the change in the PageRank vector is less than tol, the algorithm will stop. For example, it will stop the power iteration when $||r^{(k)} - r^{(k-1)}||_1 \leq 10^{-8}$.

c) Compare these 9 approaches based on the various metrics described above.

1. MAP:
Across all weighting schemes, the WS and CM methods perform than NS, indicating the PageRank scores are not helpful in this particular dataset. PTSPR has the highest MAP for the WS weighting schemes, while QTSPR has a slightly higher MAP for the NS weighting schem.

2. Precision at 11 standard recall levels:
At recall level 0.00, PTSPR has the highest precision in CM weighting schemes. As recall increases, there is no consistent best-performing method across all weighting schemes. For all recall levels and all weighting schemes, the differences in precision between the three methods are very marginal. However, a pattern can be observed where GPR tends to have slightly higher precision at lower recall levels, while PTSPR slightly outperforms at higher recall levels.

3. Wall-clock running time in seconds:
For retrieval time, QTSPR is the fastest method for WS and CM weighting schemes, while PTSPR is the fastest for NS. For PageRank time, GPR is significantly faster than both QTSPR and PTSPR, which have almost identical times.

In conclusion, for all methods, we could observe that PageRank score is not so useful.

d) Analyze these various algorithms, parameters, and discuss your general observations about using PageRank algorithms.

- Using PageRank algorithms for document retrieval can be effective in terms of ranking quality as indicated by the MAP and precision at various recall levels. However, there is a trade-off between ranking quality and computational efficiency. For example, PTSPR tends to have higher MAP and precision but is slower in terms of wall-clock running time compared to QTSPR.
- The performance of the algorithms can vary with the weighting scheme used. This suggests that the choice of weighting scheme is crucial for the performance of the PageRank algorithms.
- There is no one-size-fits-all solution. The best algorithm and parameters may vary depending on the specific use case, the importance of ranking quality versus computational efficiency, and the characteristics of the data being used.

e) 1. What could be some novel ways for search engines to estimate whether a query can benefit from personalization?.

We could analyze query ambiguity, which involves determining the level of ambiguity in a query. Ambiguity in a query arises when the query can have multiple meanings or interpretations. For example, apple could refer to both a fruit and a company. To estimate whether such a query can benefit from personalization, the search engine can analyze the user's past searches or preferences. If the user frequently searches for topics related to technology, it's likely that they are referring to Apple Inc., the technology company. Hence, personalizing the search results to show information about Apple Inc. would enhance the user's search experience. Analyzing the level of ambiguity in a query and the user's past searches or preferences can help the search engine estimate whether the query can benefit from personalization.

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?

We could continuously update a user's profile based on their recent online activity and interactions with content. This approach recognizes that a user's interests may change over time and allows for a more dynamic and accurate representation of their interests. It involves collecting real-time data on a user's online activity and interactions, analyzing the data using machine learning and data analytics to identify and update the user's interests, and then recommending content based on the updated profile. A feedback loop is created by monitoring the user's interaction with the recommended content and using this feedback to further update their profile. This approach is adaptive, real-time, and personalized, but faces challenges related to data privacy and computational resources.

3. Details of the software implementation

a) Describe your design decisions and high-level software architecture;

The entire code is contained within a single main.py file to keep the application simple and straightforward.

- Generalized PageRank (GPR): This part of the code is reading a transition matrix from a file, normalizing it, handling dangling nodes, and then computing the PageRank vector using the power method.

- Indri-list Consolidation: This part of the code is consolidating all indri-list files into a single text file.
- GPR TreeEval File Creation: This part of the code is reading the consolidated indri-list file and computing a new score for each document based on the GPR vector. It supports three methods: NS (New Score), WS (Weighted Score), and CM (Combination Method). The new scores are then used to create a TreeEval file.
- Topic-Sensitive PageRank (TSPR): This part of the code is computing a PageRank vector for each topic using the transition matrix and the topic distribution. It then computes a query-topic distribution and a user-topic distribution.
- Query Topic-Sensitive PageRank (QTSPR): This part of the code is computing a PageRank vector for a specific query by combining the topic-specific PageRank vectors using the query-topic distribution.
- Personalized Topic-Sensitive PageRank (PTSPR): This part of the code is computing a PageRank vector for a specific user and query by combining the topic-specific PageRank vectors using the user-topic distribution.
- TSPR TreeEval File Creation: This part of the code is creating a TreeEval file for the QTSPR and PTSPR vectors using the same methods as the GPR TreeEval file creation.

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

- The code is using a sparse matrix representation for the transition matrix, which is efficient for large matrices with many zeros.
- The code is handling dangling nodes by redistributing their rank uniformly across all nodes.
- The code is using a dampening factor in the TSPR calculation to ensure convergence.
- The code is using a specific user and query for the QTSPR and PTSPR calculations.

c) Describe any programming tools or libraries and programming environment used;

- Python 3.9 (Visual Studio Code)
- Libraries: numpy, pandas, scipy, os, csv, time
- Environment: Windows 10, 64 bit

d) Describe strengths and weaknesses of your design, and any problems that your system encountered

Strengths:

- Modularity: My code is organized into functions, each with a single responsibility, which makes it easier to understand, test, and modify.
- Efficiency: I use sparse matrices and vectorized operations, which can lead to significant performance gains for large datasets.

Weaknesses:

- Memory Usage: For very large datasets, even sparse matrices and vectorized operations might use a lot of memory. Depending on the size of the data you are working with, this could be a limiting factor.
- Code Complexity: While the modularity of your code is a strength, it also means that there are many different parts that interact, which can make it harder to understand the whole system for someone unfamiliar with my code.

Problems Encountered:

- Convergence: Depending on the data, the GPR algorithm might take a long time to converge, or might not converge at all within the maximum number of iterations.
- Data Size: As mentioned above, for very large datasets, memory usage might be a problem. This could potentially be addressed by using a more efficient data structure or by implementing the algorithm in a more memory-efficient way.