Name: Yan Zhao
Andrew ID: yanzhao2

Machine Learning for Text Mining

Homework 3

1. Statement of Assurance

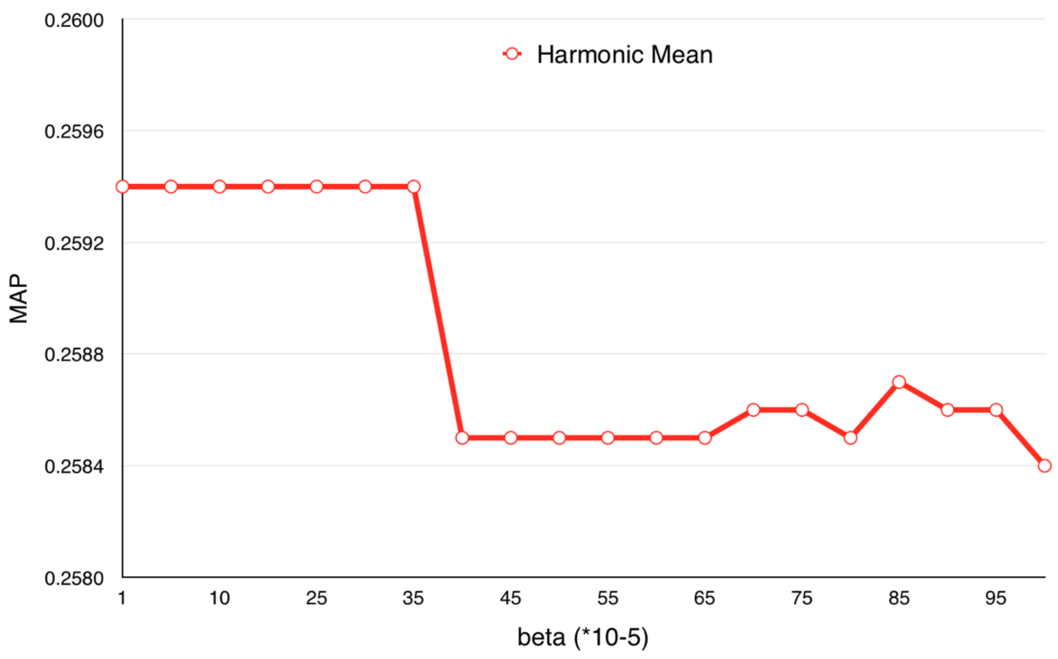
a) I did not receive help of any kind from anyone in developing your software for this assignment.
b) I did not give help of any kind to anyone in developing their software for this assignment.
c) I am the author of every line of source code submitted for this assignment.
d) I am the author of every word of your report.

2. Experiments

a) Describe the custom weighing scheme that you have implemented. Explain your motivation for creating this weighting scheme.
I come up with three ideas for my custom weighting scheme. To help make decisions, I experiment on using Indri score alone and get MAP equals to 0.2594.
i) I try to average Indri score and PageRank score with Harmonic Mean. My intuition is instead of arithmetic mean, I can try with different average method to see if there are some improvements. And due to PageRank scores are quite small, I scale them by multiply a scale factor equals to 1000. Then I tuned factor *beta*. The formula is described below:

$$\text{score} = \frac{1}{\frac{1}{1+\beta^2} * \frac{1}{Indri} + \frac{\beta^2}{1+\beta^2} * \frac{1}{PageRank}}$$

When beta is less than 1, this mean will favor Indri score. Since Indri score is more accurate in this experiment, so I tuned beta with values less than 1.
However, based on the experiments, this method is not quite useful. As we can see from the results below, I record experiments on tuning *beta* with GPR. Here *beta* = 0 means we totally use Indri score, which should have a MAP equals to 0.2594. But when I tuned beta from a value close to 0 (1e-5) and grow, the MAP drop significantly and stay at a very low level, even I set each step to be extremely small (by increasing 1e-5 one time).
I think the reason for the failure in this weighting scheme is that harmonic mean is best used in situations where extreme outliers exist. And PageRank scores are distributed actually non-uniformly, with some significantly large values. Thus, they will influence the final average significantly and cause the retrieval results drop significantly.

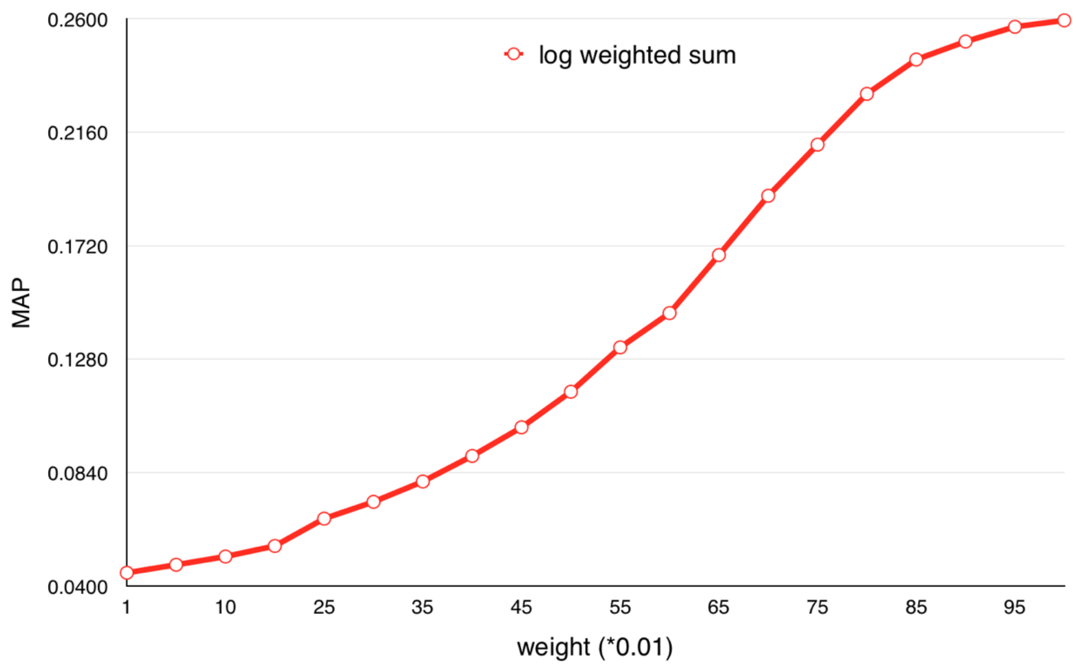


ii) The second idea is to use weighting sum, while for PageRank score, I calculate with a *log* function. The intuition is also simple, since Indri score is actually an Inference Nets + Statistical language model, which measures probability of a document given a query. And for this assignment, all Indri scores are calculated by a *log* function to avoid small values. Also, according to definition, PageRank is calculated by Markov Matrix which means it is also represented as probability in this assignment. Thus, if I want to perform with a weighted sum, I need to compare Indri and PageRank scores at the same level, so I give PageRank a log function and calculate final scores as combination of two probabilities. The formula is as below.

$$\text{score} = \text{weight} * \text{Indri} + (1 - \text{weight}) * \log(\text{PageRank})$$

Based on the experiment, the method is also not quite useful. As is shown below, I run experiments tuning weight with GPR. With the increase of Indri weight, the MAP get improved continuously. And as Indri weight gets closer to 1, MAP gets closer to 0.2594, which is the baseline of using Indri score alone. Through this result, I find PageRank factor in this weighting scheme has no help at all on improving retrieval results.

I think the reason of failure for this weighting scheme is that compared with Indri score, PageRank score is not very useful in this assignment. By performing log function on PageRank, I actually magnify the influence of PageRank on retrievals. Thus, it may be better to put more heart on Indri score and use PageRank score for tiny modifications, as my third try of custom weighting scheme.

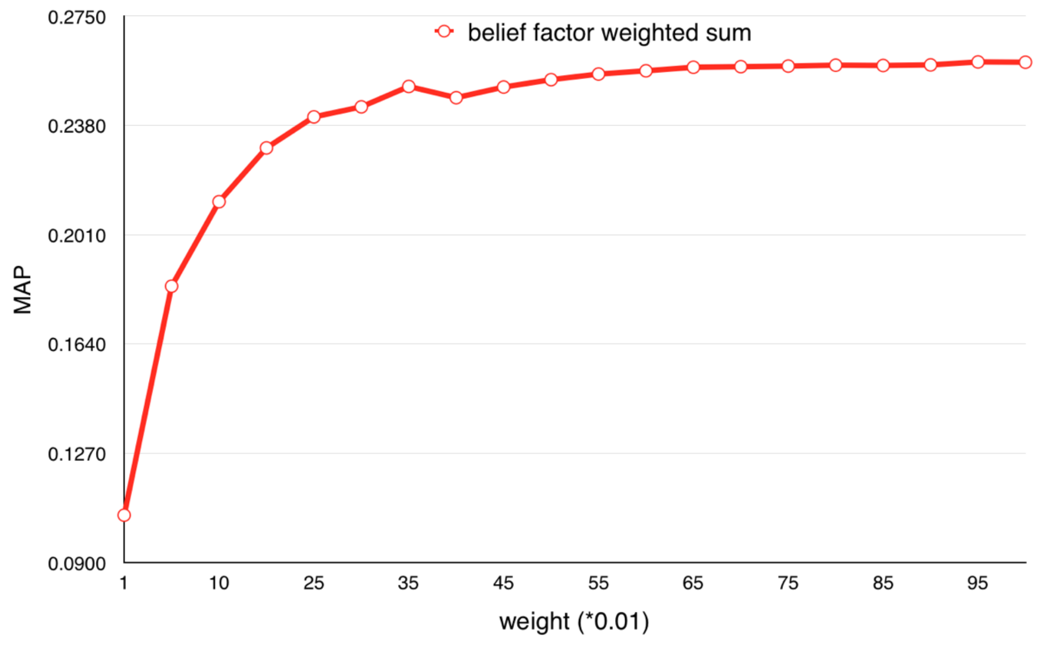


iii) Based on the experience on second weighting scheme, I try to magnify influence of Indri score. **This is the custom method I use in the following sections.**

I first normalize PageRank scores using min-max normalization, which is PR = (PR-min)/(max-min), to normalize PageRank scores to [0,1]. My intuition is to represent PageRank as a trust factor, where 1 means totally trust and 0 means not trust at all. Then I multiply this trust factor with Indri score, which means how much I trust this score, and finally use this as an award to the original Indri score. The formula is shown below. Since Indri scores are all less than 0, so I use a minus to make the latter factor larger than 0. For example, if one PageRank score is large, closer to 1, then *–(1-weight)*PR*Indri* will be a relatively large positive number, meaning high award for the original Indri score.

$$\text{score} = \text{weight} * \text{Indri} - (1 - \text{weight}) * \text{Norm\_PR} * \text{Indri}$$

Based on the experiment, the method is much better than the first two. As is shown below, I run experiments tuning weight with PTSPR. With the increase of Indri weight, the MAP get improved, but stay when weight equals to about 0.60, which is quite different from first two methods. Also, when Indri weight equals to 0.95, MAP gets to 0.2595, which is a tiny better than the baseline using Indri score alone. Through this result, I find using PageRank score as trust factor of Indri score can be helpful on retrieving documents. And I tune parameters and analyze results using this method in the following sections.



b) Report of the performance of the 9 approaches.

Since PageRank score are very small, when performing weighted sum, I scale PageRank by multiplying a scale factor equals to 1000, for better weight tuning.

I. Metric: MAP (Baseline with Indri Alone: 0.2594)

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0456	0.2593	0.2593
QTSPR	0.0431	0.2598	0.2592
PTSPR	0.0471	0.2598	0.2595

II. Metric: Precision at 11 standard recall levels

Precision @ 0% recall

Method \ Weighting Scheme	NS	WS	CM
GPR	0.1805	0.8314	0.8314
QTSPR	0.1618	0.8313	0.8313
PTSPR	0.1588	0.8306	**0.8358**

Precision @ 10% recall

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0794	0.5841	0.5841
QTSPR	0.0752	0.5840	0.5841
PTSPR	0.0877	**0.5843**	0.5840

Precision @ 20% recall

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0750	0.4658	0.4658
QTSPR	0.0698	**0.4712**	0.4654
PTSPR	0.0839	0.4668	0.4661

Precision @ 30% recall

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0700	0.3704	0.3702
QTSPR	0.0669	0.3699	0.3706
PTSPR	0.0715	**0.3713**	0.3692

Precision @ 40% recall

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0664	0.3084	0.3084
QTSPR	0.0629	0.3085	0.3084
PTSPR	0.0660	**0.3133**	0.3085

Precision @ 50% recall

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0609	0.2360	**0.2360**
QTSPR	0.0582	0.2348	0.2355
PTSPR	0.0594	0.2355	0.2358

Precision @ 60% recall

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0484	0.1597	0.1597
QTSPR	0.0459	0.1596	0.1593
PTSPR	0.0463	**0.1604**	0.1601

Precision @ 70% recall

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0302	0.0938	0.0938
QTSPR	0.0284	0.0939	0.0938
PTSPR	0.0282	**0.0942**	0.0938

Precision @ 80% recall

Method \ Weighting Scheme	NS	WS	CM

GPR	0.0119	0.0578	0.0575
QTSPR	0.0116	**0.0582**	0.0572
PTSPR	0.0120	0.0575	0.0572

Precision @ 90% recall

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0074	0.0399	0.0399
QTSPR	0.0075	0.0397	0.0400
PTSPR	0.0073	**0.0404**	0.0399

Precision @ 100% recall

Method \ Weighting Scheme	NS	WS	CM
GPR	0.0042	**0.0104**	0.0104
QTSPR	0.0042	0.0103	0.0104
PTSPR	0.0041	0.0103	0.0104

III. Metric: Wall-clock running time in seconds

For running time I only measure calculation time, without including file I/O time.

Method \ Weighting Scheme	NS	WS	CM
GPR	0.6021	0.6236	0.6451
QTSPR	6.3432	6.4642	6.3583
PTSPR	1.0143	1.7525	1.9258

IV. Parameters

For this assignment, I mainly tuned damping factor and indri score weight. For damping factor, I set factor of transitional matrix **M** as *alpha*, factor of topic sensitive vector $\mathbf{p_t}$ as *beta*, and factor of $\mathbf{p_0}$ as *(1-alpha-beta)*. For the stop criteria, I fix number of iterations to be equal to 200, and difference of iterations to be equal to 1e-13.

Method \ Weighting Scheme	NS	WS	CM

GPR	alpha = 0.8	weight = 0.88
		alpha = 0.8
			weight = 0.89
			alpha = 0.8

QTSPR	alpha = 0.8
	beta = 0.1
		weight = 0.67
		alpha = 0.8
		beta = 0.1
			weight = 0.84
			alpha = 0.8
			beta = 0.1

PTSPR	alpha = 0.2
	beta = 0.8
		weight = 0.54
		alpha = 0.4
		beta = 0.4
			weight = 0.96
			alpha = 0.4
			beta = 0.4

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

1) For MAP value, we can see with only PageRank scores, all three algorithms perform very bad, which means given no query, PageRank cannot reflect user's information needs efficiently. However, if we combine Indri scores with PageRank, we can get improvements. Since using Indri alone we can get MAP of 0.2594, using custom method I can only improve MAP to 0.2595 by PTSPR, which is nearly the same as baseline, and I cannot get improvements by two other algorithms. When I use weighted sum on PTSPR and QTSPR, I can get a tiny improvement to 0.2598. In conclusion, from MAP point of view, PTSPR and QTSPR using weighted sum perform the same, better than the others.
2) For Wall-clock running time, we can see three weighting schemes differ a little bit while three algorithms differ a lot. GPR takes least time because it does not need user or query prior to calculate PageRank, while QTSPR and PTSPR have to calculate PageRank for all 12 topics. However, QTSPR runs 6 times as PTSPR. I think the reason is in QTSPR, I give transition matrix M a large damping factor: alpha = 0.8, while in PTSPR, I give transition matrix M and topic vector p_t equal weight: alpha = 0.4, beta = 0.4. As a result, QTSPR needs more iterations to converge while PTSPR can converge fast. In conclusion, from running time point of view, PTSPR outperforms QTSPR, even they have same MAP higher than baseline.
3) For Precision at 11 standard recall levels, we can see PTSPR with weighted sum gets highest precision at 10%, 30%, 40%, 50%, 60%, 70% and 90%; QTSPR with weighted sum gets highest precision at 20% and 80%; GPR with weighted sum gets highest precision at end point 100%; while PTSPR with CM only gets highest precision at 0%. Thus, PTSPR with WS gets highest score in more precision-recall levels, QTSPR gets less and GPR only majors in end point. From this point of view, PTSPR with weighted sum outperforms the other approaches

In conclusion, PTSPR with weighted sum is the best approach we can get.

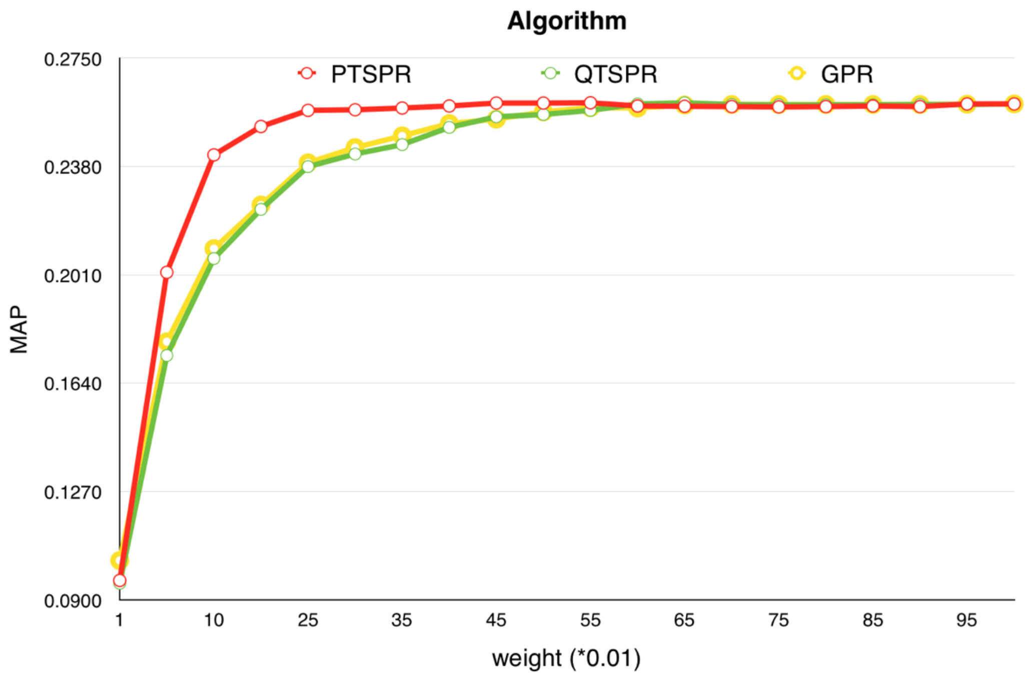
d) Analyze these various algorithms, parameters, and discuss your general observations about using PageRank algorithms.

For this section, I did a lot of experiments trying to find trend from different aspects.

1) First I compare three algorithms (GPR, QTSPR, PTSPR) using each of best tuned damping factors, which is (0.8,0.1) for GPR and QTSPR; (0.4,0.4) for PTSPR, and observe the MAP value with restrict to weighted sum method. Results are shown below.

   As is shown, PTSPR gets highest MAP value at about weight=0.5, QTSPR gets highest MAP value at about weight=0.7 and GPR gets highest MAP value at about weight=0.9. Also, PTSPR gets stayed at about weight=0.25, with a tiny fluctuate while weight increases; QTSPR and GPR have similar trend, which get stayed at about weight=0.55 with tiny fluctuation after that. Another thing needs to mention is only PTSPR and QTSPR beat baseline with weighted sum.

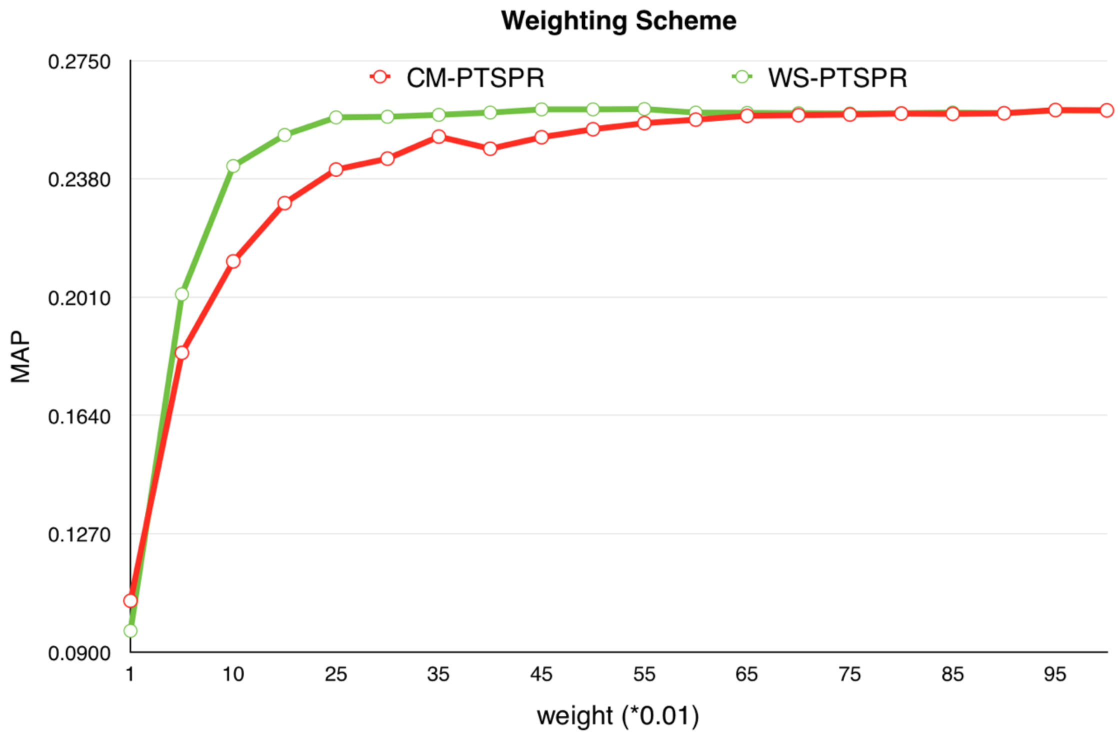
Based on these information, we can see PTSPR outperforms the other two, which is same as I analyzed in section c. With increasing of weights, MAP of PTSPR get stayed earlier than the other two algorithms. This observation shows PTSPR can have a higher MAP value even if we give a relatively high weight to PageRank score, and thus show Personalized PageRank is more meaningful to combine with Indri.



2) Since PTSPR has best performance, so in second step I compare two weighting schemes (WS and CM) using PTSPR with damping factors (alpha=0.4, beta=0.4), and compare MAP value with restrict to Indri weight. Results are shown below.

   As is shown, WS gets highest MAP value at about weight=0.5, and CM gets highest MAP value at about weight=0.95. Also, WS gets stayed at about weight=0.25, with a tiny fluctuate while weight increases; while CM gets stayed at about weight=0.7 with tiny fluctuation after that. Also, PTSPR with weighted sum gets higher MAP (0.2598) than with custom weighting scheme (0.2595).

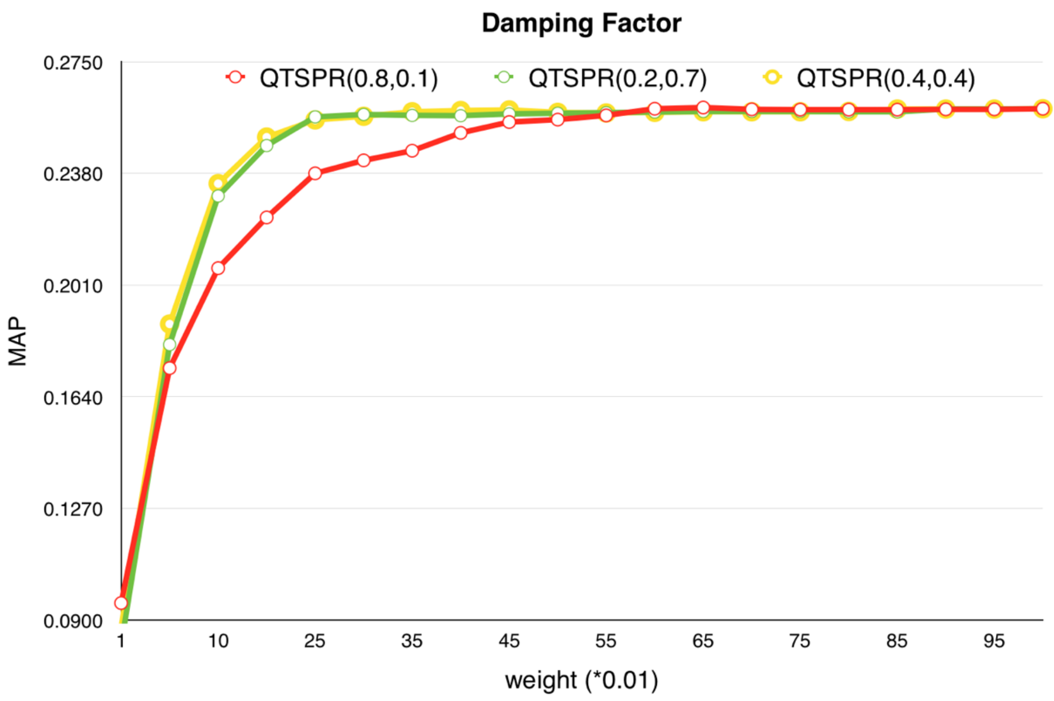
   Based on these information, we can see WS outperforms CM, which is same as I analyzed in section c. With increasing of weights, MAP of WS get stayed earlier than CM. As analyzed before, this observation shows WS can have a higher MAP value even if we give a relatively low weight to Indri score, and thus show weighted sum is more meaningful than treating PageRank as a trust factor in CM.



3) Thirdly I compare different damping factors for three algorithms. As is shown in section 2, GPR gets best performance when alpha = 0.8, QTSPR gets best performance when alpha = 0.8 and beta = 0.1, PTSPR gets best performance when alpha = 0.4 and beta = 0.4. To analyze damping factor in detail, I compare QTSPR using three damping factor pairs: alpha,beta=0.8,0.1; alpha,beta = 0.4,0.4; and alpha,beta = 0.2,0.7 with restrict to Indri weights. Results are shown below.

   As is shown, damping factor (0.4,0.4) and (0.2,0.7) get stayed at about weight=0.25, with a tiny fluctuate while weight increases; while damping factor (0.8,0.1) gets stayed at about weight=0.6 with tiny fluctuation after that.

   Based on these information, we can see damping factor (0.8,0.1) is the worst among three, which is a little different from results in section c. In section c, I treat (0.8,0.1) as best damping factor for QTSPR because it can get highest MAP value (0.2598). But based on analyze here, with increasing of weights, MAP of (0.8,0.1) get stayed later than the other two. One explanation is compared with transition matrix, personalized or topic vector p_t has more or equal importance. Giving much damping factor to transition matrix will cause final PageRank score to be less robust. There can be cases when high damping factor on transition matrix gets better performance, but it will lose stability at the same time. In this way, this result has more or less same meaning as section c, which shows PTSPR with damping factor (0.4,0.4) is better.



In general, for PageRank, we can conclude that PageRank is less useful than Indri in retrieval. This is because Indri estimates the probability of a document given a query, which can better reflect the information needs of user. However, PageRank estimates the connection properties of a document with no relationship to query. In some cases, users want to find documents with many connections while in other cases, users try to retrieve specific documents even with no connections. Also, some documents with many connections are likely to be spams or meaningless ones, which will has negative influence on retrieval as a result.

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

For a search engine, we can retrieval documents of a query with and without personalization, and interleave the retrieval results just like a diversity retrieval. Then we can record the click action of users to analyze which one can be better represent user's information needs.

Another way to estimate users' favor properties is by estimating search log. We can set a threshold of time like 30 minutes, search actions within 30 minutes can be treated as with same information needs and should be in the same search log. Then we can analyze the behavior or users in search log, if it has many click and search actions, we can say the first retrieval in the search log meets users' information needs, and thus should be judged as good retrieval.

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?

I come up with two ways in identifying user's interests.

First is group by users. We can get search log of a user and get his frequently searched documents, reform them by a feature vector to represent a user. Thus we can build a matrix where each row is user and each column is feature, then we can perform clustering algorithms like K-means to group users. As a result, for users in the same group, we can infer a same topic preference within whole user group.

Second is each user, we can represent them by a feature vector including their gender, education level, ages, documents frequently read, etc. And we can get topic preference training data from web. After that, we just need to perform a logistic regression or Naïve Bayes model to predict the probability distribution of different topics.

3. Details of the software implementation

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

I want to make code more clear. All the code for calculating PageRank is in pagerank.py.

I define some reader function to read given files to transition matrix (sparse matrix), p_t (vector), user and query prior (vector), and indri scores (dictionary). Then I read parameters from files to decide algorithms, weighting schemes etc.

After this, I do the calculation by calling different algorithm functions, while all of them will call the same iteration helper function at each step.

Finish calculating PageRank, I will combine it with Indri scores by different weighting schemes, and re-rank to outputs.

Besides of pagerank.py, I also write a fetch.py, which can upload my result to website and fetch metric output automatically. Inside the python file there are some functions like tuningWeight(), tuningDamping(), with a bit modification, I can tune different parameters and get retrieval results to help analyze.

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

I use scipy to build a csr sparse matrix as transition matrix M, and for each iteration, I multiply this sparse matrix by r to speed up computation.

Also, when calculating PageRank, instead of using $r^{(k+1)} = ((1-\alpha)M + \beta E_T + \gamma E)^T r^{(k)}$, I use derived formula $r^{(k+1)} = (1-\alpha)M^T r^{(k)} + \beta p_t + \gamma p_0$ in every iteration. The reason is that M is a sparse matrix while E is dense one. If I use the first formula, then I will multiply a whole dense matrix by r, but with the second formula, I only need to calculate a sparse matrix multiply by r and then add two dense vectors. In this way, I can reduce the run time significantly.

Besides, for online computing to calculate user/query prior with PageRank, I store prior of topics in a row vector, I transpose PageRank to a row vector and concatenate all topics' PageRank to a matrix, then I make dot product of these two to directly finish calculation on TSPR.

c) Describe any programming tools or libraries that you used;

I use scipy to store and calculate sparse transition matrix M. And I use sklearn to help me do L1 normalization of M by rows. All the other codes are written by myself including PageRank calculation

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

1) The strengths of my program lies in three aspects.

First I don't have any redundancy in my program. I have iteration helper functions which can perform each iteration on calculating PageRank, so that every algorithm can call. Also, once I finish calculating PageRank, I will combine it with Indri using different weighting schemes without modifying or recalculating it.

Second I have a parameter file which stores all the parameters including algorithms, weighting schemes, weight, damping factor, stop criteria and so on. All I need to do is to change parameters in the file and call run() function in pagerank.py to get final retrieval documents.

Third I write a python file which can upload my retrieval result to website and fetch MAP and other metrics automatically. In this way, I can let the program to try a plenty of parameter settings and perform experiments to help me analyze results. All I need to do is to run fetch.py.

2) The weaknesses of my design lies on transition matrix.

Since there are many documents which have no connection to the others. If I just skip these documents in initialize transition matrix, it will not be a markov matrix any more, and r will not sum to one after iteration. To solve this, I set diagonal of transition matrix to be 1, assuming each document has connection to itself. My intuition is if I assume any document connect to itself, then I should assume every document connect to itself to make it fair.

However, I think this assumption may be wrong because it change the truth of connection. A better way may be set every zero-row to be uniformly distributed with probability 1/n in each entry, or normalize r to make it sum to one after each iteration. I will keep trying to improve performance in this way after this assignment.

4. Describe how to run your code (programming environment, command line, etc.)

There is a "parameter.txt" in src folder, where you can choose algorithm, change weighting scheme and modify other parameters.

To run program, just cd into src folder and type "$ python pagerank.py".