Your Name: Xueting Hu

Your Andrew ID:xuetingh

Homework 2

Collaboration and Originality

Your report must include answers to the following questions:

1. Did you receive help of any kind from anyone in developing your software for this assignment (Yes or No)? It is not necessary to describe discussions with the instructor or TAs.

   No. I used my own code from last semester, I dropped after midterm exam.

2. Did you give help of any kind to anyone in developing their software for this assignment (Yes or No)?

   No

3. Are you the author of every line of source code submitted for this assignment (Yes or No)? It is not necessary to mention software provided by the instructor.

   Yes

4. Are you the author of every word of your report (Yes or No)?

   Yes

Your Name: Xueting Hu Your Andrew ID: xuetingh

Homework 2

1. Experiment 1: Baselines

	Ranked Boolean	**BM25 BOW**	**Indri BOW**
P@10	0.0900	0.5100	0.6600
P@20	0.1300	0.5000	0.5700
P@30	0.1267	0.4500	0.5333
MAP	0.0104	0.1303	0.1526

2. Experiment 2: BM25 Parameter Adjustment

2.1. k_1

	k_1							
	1.2	0	1	2	4	1.4	1.6	1.8
P@10	0.5100	0.1800	0.5100	0.5100	0.4600	0.5100	0.5100	0.5100
P@20	0.5000	0.1800	0.4950	0.4900	0.4300	0.5150	0.5050	0.5150
P@30	0.4500	0.2000	0.4400	0.4533	0.4300	0.4600	0.4567	0.4600
MAP	0.1303	0.0404	0.1328	0.1230	0.1074	0.1295	0.1294	0.1282

2.2. b

	b							
	0.75	0	0.25	0.50	1	0.20	0.30	0.4
P@10	0.5100	0.3600	0.6700	0.5900	0.3500	0.7100	0.6200	0.6200
P@20	0.5000	0.3700	0.6050	0.5750	0.3550	0.5700	0.6200	0.6000
P@30	0.4500	0.3567	0.5300	0.5267	0.3367	0.5167	0.5400	0.5467
MAP	0.1303	0.0908	0.1646	0.1586	0.0996	0.1630	0.1651	0.1622

2.3. Parameters

According to textbook, without optimization (like grid search), experiments have shown reasonable values are to set k_1 between 1.2 and 2 and b = 0.75.

For k_1 I tried a sequence in ascending order, most values was twice larger as the previous one, and inserted three values between 1.2 and 2 to take a closer look at the suggested range. This value range went beyond the textbook recommended range, so I could observe the how MAP and P@N changed as k_1 changed.

For parameter *b*, it should be in [0,1]. I narrow down the range to find the b value with best performance. I first tried some values uniformly within the range [0, 1]. I observed that the performance is better when b is 0.25, then I focused on this, tested b=0.2, b=0.3 and finally b=0.4. I observed the change of MAP and P@N as *b* changed, knowing the general pattern in this process, and I think the best performance with this set of query strings can be achieved by setting b to somewhere near 0.3.

2.4. Discussion

k_1 is a parameter in tf weight, $\frac{tf}{tf + k_1(\dots)}$. We introduce k because we don't want a document whose tf =10 gets too much more score than tf=5, and the smaller the k_1 is, the tf weight increases slower as tf increases. When we set k_1=0, tf weight is a constant 1. In the experiment fixing b=0.75, MAP goes up and then goes down, and within [1.2, 2] the MAP is higher like textbook indicated (but not too much higher) than out of this range, and within this range, the performance appears better when b is between 1 and 1.2. I can translate this observation of k_1 like this: for this set of query strings, we don't want to value term frequency in a document very much, so I think of a scenario like this: doc A has only one of the term in the query string and it appears 10 times, doc B has 3 of the query terms and each appears 1 time, so a smaller k_1 favors doc A, that maybe indicates that the query strings are hard to match in this collection of documents (our index). The reason that MAPs not vary much is that most tf is not large, so the impact of tf weight to the score is not large, therefore, the factor k_1 doesn't affect the performance very much. When k_1 =4, the performance drops to 0.1074, so too much adjustment to tf weight is not good for performance.

b is the parameter in tf weight, $\frac{tf}{tf + k_1(1 - b + b \times \frac{docLength}{avgLength})}$. As b increases, the penalization for long document increases, while as *b* goes to 0, the tf weight does not care about document length, that means longer documents (usually with larger tf) are favored, as can be seen from the result, MAP of b=0 is significantly low compared to others. In the experiments part, b=0.3 (far away from the popular choice b=0.75) had the best performance. It may indicate the best value of *b* is relevant to document collection

we are retrieving in and the set of query strings, our query needs smaller b than a common situation, maybe it is because the retrieved documents are not very long so they don't need much penalization.

In this experiment, MAP changes more obviously when b changes. So that might indicate that adjusting k_1 (means adjusting how much we value term frequency) is not as important as adjusting b (means adjusting tf weight based on document length).

3. Experiment 3: Indri Parameter Adjustment

3.1. μ

	μ							
	2500	1000	4000	2000	3000	1800	1600	1900
P@10	0.6600	0.6100	0.6500	0.6300	0.6500	0.6300	0.6200	0.6300
P@20	0.5700	0.5850	0.5550	0.5800	0.5700	0.5800	0.5850	0.5800
P@30	0.5333	0.5233	0.5300	0.5300	0.5333	0.5300	0.5267	0.5333
MAP	0.1526	0.1474	0.1482	0.1538	0.1519	0.1542	0.1528	0.1535

3.2. λ

	λ							
	0.4	0	0.2	0.6	0.1	0.3	0.05	0,35
P@10	0.6600	0.6300	0.6500	0.6300	0.6400	0.6500	0.6300	0.6500
P@20	0.5700	0.5800	0.5850	0.5650	0.5850	0.5750	0.5850	0.5700
P@30	0.5333	0.5467	0.5433	0.5033	0.5500	0.5333	0.5533	0.5367
MAP	0.1526	0.1539	0.1515	0.1509	0.1520	0.1525	0,1529	0.1524

3.3. Parameters

In this experiment, I first pick up 1000 and 4000 which are on different side of 2500, and both have smaller MAP than 2500, so I then try 2000 and 3000, μ=2000 has higher MAP, and then pick some value near it, finally get the highest MAP among those 8 value, when μ=1900.

I test different value trying to find a best λ, MAP is the highest when λ=0, however, those values seem not to have obvious influence on MAP and P@N.

3.4. Discussion

The parameter μ a smoothing parameter in $p(q_i|d) = \frac{tf + \mu p_{MLE}(q_i|C)}{length(d) + \mu}$, without μ it would be MLE only; when μ is large, tf and document length has less effects. μ also has different impact on different document lengths, as for long documents probabilities are more smooth and large μ has not too much effects; for short documents, probabilities are less smooth and large μ has more effects. The parameter μ has less effect over long documents than over short ones, large documents tend to have higher score than short ones. In the experiment, the MAP goes up and then goes down as μ increases, reaching maximum at $\mu = 1900$. I think maybe the documents that satisfy the query in this index are not very long.

The parameter λ is the smoothing parameter that has the "idf like" effect.
$p(q_i|d) = (1-\lambda)p_{MLE}(q_i|d) + \lambda p(q_i|C)$, it should be within [0, 1], otherwise the term frequency or the smoothing term from maximum likelihood estimator might be negative. The parameter λ gets larger, the smoothing effect becomes more obvious. In the experiment, as λ increased, MAP decreased, i.e. without smoothing we got the best performance. The reasons are that our queries are short, thus all terms must match and "idf weighting" is less important; also, maybe because the scores are good enough so we don't need prior so much, so the distribution of terms in this document collection is not far away from average.

In this case, μ has a larger range than λ, and MAP changes more obviously when μ changes. So that might indicate that adjusting idf weight (punishing common term) is not as important as adjusting MLE (tf weight) based on document length. This conclusion is similar with Experiment 2.

4. Experiment 4: Different representations

4.1. Example Query

Following is an example with weights: 0.1 url, 0.2 keywords, 0.3 title, 0.4 body for query "indiana child support".

#AND (#WSUM(0.1 indiana.url 0.2 indiana.keywords 0.3 indiana.title 0.4 indiana.body) #WSUM(0.1 child.url 0.2 child.keywords 0.3 child.title 0.4 child.body) #WSUM(0.1 support.url 0.2 support.keywords 0.3 support.title 0.4 support.body))

4.2. Results

	Indri BOW (body)	**0.1 url 0.1 keywords 0.7 title 0.1 body**	**0.1 url 0.7 keywords 0.1 title 0.1 body**	**0.7 url 0.1 keywords 0.1 title 0.1 body**	**0.3 url 0.2 keywords 0.1 title 0.4 body**	**0.2 url 0.1 keywords 0.1 title 0.6 body**
P@10	0.6600	0.5300	0.4400	0.5400	0.6000	0.6400
P@20	0.5700	0.4400	0.3550	0.4350	0.5350	0.5450
P@30	0.5333	0.4100	0.3633	0.4400	0.4900	0.5067
MAP	0.1526	0.1171	0.1100	0.1267	0.1462	0.1503

4.3. Weights

Since we already have the result for only body field, so I set 3 series of weights separately focusing on url, keywords and title; and according to these 3 results I set weight of url field higher than those of keywords and title to see if it gets better results, column 4 and 5 are different in a way that 5 has larger weight of body and equal weights of title and keywords.

4.4. Discussion

In terms of MAP, the body field deserves the largest weight and url second largest. However, combining other fields (in column 4 and 5) seems only draw back the performance, comparing to when there is only body field.

The reason that body field has more effect over MAP might be that if there is term that is not in the documents, but Indri operator still provides the default score for those terms. When the default score value is large enough, even the documents with missing query terms matching might have higher score than other documents with every query terms matching (but may have lower term frequency). As for url field with dominant weight, the P@N is far better than keywords and title with dominant weight, but it is worth noticing that as N increases, P@30 is higher than P@20 for both queries where keywords or url have higher weight in column 2,3. I think the reason is that keywords and url are hard to match, but as we look at more documents, those who match are more likely to be relevant. In this experiment, results show that only using body field leads to the best performance, but combining body field with other fields can give a more stable result, that is P@N doesn't decrease much when N gets larger.

5. Experiment 5: Sequential dependency models

5.1. Example Query

The example is of following weights: 0.40 AND 0.20 NEAR 0.40 WINDOW:

#wand(0.4 #and(indiana child support) 0.2 #and(#near/1(child support) #near/1(indiana child)) 0.4 #and(#window/8(child support) #window/8(indiana child)))

5.2. Results

	Indri BOW (body)	**0.10 AND 0.80 NEAR 0.10 WINDOW**	**0.10 AND 0.10 NEAR 0.80 WINDOW**	**0.20 AND 0.40 NEAR 0.40 WINDOW**	**0.40 AND 0.20 NEAR 0.40 WINDOW**	**0.40 AND 0.40 NEAR 0.20 WINDOW**
P@10	0.6600	0.3200	0.4900	0.4300	0.4800	0.4300
P@20	0.5700	0.3200	0.4250	0.3750	0.4200	0.3850
P@30	0.5333	0.2567	0.3700	0.3300	0.3800	0.3233
MAP	0.1526	0.0718	0.1157	0.1051	0.1161	0.1059

5.3. Weights

For baseline, the weight is 1.0, 0, 0 for only using #AND operator. I set 2 series of weights separately focusing on #NEAR and #WINDOW; the rest 3 series are same weights distributed differently, to see how operators work together.

5.4. Discussion

#AND operator is the most effective way for our retrieval system in terms of Precision. In addition, it's the fastest way that #AND operator doesn't go through loops. Second, #WINDOW/8 performs better than #NEAR/1. The reason is that #Near/1 is a very strict operator that has high precision and low recall, so it map is low because of missing documents; while #WINDOW/8 is less strict than #NEAR/1 both in order and distance, thus has higher recall and thus #WINDOW/8 outperformed. As for the last 3 columns, column 4 that gave #AND and #WINDOW higher weights outperformed other two, I thin maybe the #AND and #WINDOW got a balance here.

For computation efficiency, #AND > #NEAR/1 > #WINDOW/8, because #AND has only one loop to find document while #NEAR and #WINDOW have two loops for document and location, and the loop of #WINDOW is more complicated since it won't judge match until go over all terms and only update the smallest location when not match (#NEAR easily break the inner loop since the match is judged all the time).