Chen Sun

chens1@andrew.cmu.edu

Homework 2

1 Statement of Assurance

You must certify that all of the material that you submit is original work that was done only by you. If your report does not have this statement, it will not be graded.

All of the material I submit is original work that was done only by me.

(All of unit of time used in this report is counted by eclipse and calculated unit of ms)

2 Experiment 1: Baselines

	Ranked Boolean	BM25 BOW	Indri BOW ctf	Indri BOW df
P@10	**0.2333**	**0.4967**	**0.5700**	**0.3067**
P@20	**0.2233**	**0.4683**	**0.4950**	**0.3017**
P@30	**0.2089**	**0.4178**	**0.4444**	**0.2722**
MAP	**0.1904**	**0.4782**	**0.5146**	**0.2958**
num_q	**30**	**30**	**30**	**30**
Time	**32242**	**45002**	**19165**	**18684**

Ranked Boolean performs as well as it was in homework 1.

It is obvious that Indri and BM25 performs much better than Ranked Boolean Algorithm. It means the statistical methods of scoring really improve the performance when both term frequency and document frequency are considered from the perspective of the whole documents frequency. From the result, both tf and idf are important measures of the relevance.

In term of time consuming, BM25 has the most time consuming while Indri relatively low time. It can be inferred that Indri with tf might be the best algorithm since it provides the highest MAP with relatively low time in use. The reason BM25 is that high might be the complexity of its calculation. In combining period, it actually faster than Indri in my implementation.

Also, the df smooth method is far more worse than ctf method, which means the distribution of terms are not evenly among the documents. Otherwise they should provide similar smoothing effect.

3 Experiment 2: Parameter adjustment

Describe what parameters you chose to adjust and the range of parameter values that you explored. Give a brief justification for your choices.

BM25	Setting 1	Setting 2	Setting 3	Setting 4	Setting 5	Setting 6	Setting 7
	b=0.7	b = 0.8	k1=1,b=0.5	k1=1	k1=1.5	k1=1.8	k1=2
P@10	0.5100	0.4800	0.5667	0.4889	0.5000	0.5067	0.5133
P@20	0.4850	0.4417	0.5150	0.4667	0.4700	0.4683	0.4700
P@30	0.4344	0.4056	0.4622	0.4178	0.4211	0.4233	0.4233
MAP	0.4953	0.4641	0.5375	0.4748	0.4811	0.4857	0.4872
num_q	30	30	30	30	30	30	30
Time	44456	44604	45394	44752	45414	46444	46257

For BM25, both b and k1 can be adjusted. According to Wikipedia page of BM25 (http://en.wikipedia.org/wiki/Okapi_BM25), k1 usually is usually in the range of [1.2, 2] and no clear value of b. I changed b to 0.8 and 0.7 the performance changes quite much comparing with its precision. As we can see, the running time is similar, the MAP drops as the value of b increases. It means the weight of doclen/ag_doclen is not good a good sign to calculate the precision.

Then I moved to k1. I changed k1 as 1, 1.2, 1.5, 1.8 and 2 as shown from setting 4 to setting 7. From the table, it is clear that the MAP increases with the growing of k1. The bigger k1 is, the less weight tf takes. That's why the results increases. Setting 3 was by accident. And it turns amazingly good. Thus I tried k1=2, and b =0.5 and MAP was **0.5485,** which is extremely good for the method. The bigger k1 and smaller b are, the higher rate of MAP is got. In this way, tf weight less and doclen affect the result less.

Indri	Setting 1	Setting 2	Setting 3	Setting 4	Setting 5	Setting 6	Setting 7
	smooth= df	u=2000 λ=0.4	u=2000 λ=0.5	u=2000 λ=0.7	u=3000 λ=0.4	u=3000 λ=0.5	u=3000 λ=0.7
P@10	0.3067	0.5633	0.5667	0.5700	0.5467	0.5433	0.5500
P@20	0.3017	0.5050	0.5117	0.5167	0.4833	0.4867	0.4933
P@30	0.2722	0.4444	0.4478	0.4556	0.4411	0.4444	0.4478
MAP	0.2958	0.5193	0.5254	0.5315	0.5058	0.5113	0.5155
num_q	30	30	30	30	30	30	30
Time	18684	21027	19285	18319	19718	19489	19222

For Indri, firstly change its smooth method to df. The performance is shown there. The tf smoothing method might not serve as a good smoothing method comparing with ctf. Setting 2 to Setting are all done in ctf. It is clear that the larger lambda is, the higher MAP it gets. This can be inferred from Setting 2,3,4 and Setting 5,6,7. This means that the precision mainly be decided by term frequency part rather than the smoothing part. Also, compare 2,5, and original query, the MAP drops as the u gets bigger. Again, term/length fits here better.

There is no much difference in running time.

4 Experiment 3: Different representations

	Indri BOW (body)	url	keywords	title	inlink
P@10	**0.5700**	0.3667	0.2433	0.4333	0.2433
P@20	**0.4950**	0.3067	0.2500	0.3967	0.1900
P@30	**0.4444**	0.2778	0.2411	0.3400	0.1478
MAP	**0.5146**	0.2263	0.2241	0.3287	0.1070
num_q	**30**	30	30	30	30
Time	**19165**	5522	13525	3315	4892

All the four sets run terribly on their results in precision and mapping. The order is that body > title > url>keywords > inlink.

This performance matches my previous expectations. Title is usually present in a document and generalized the main topic in the documents. Thus, if there is a match in title, it has more chance to be a relevant document. Also, title field is usually very short, making it easy to retrieve from the documents set.

Keywords are in the metadata fields and are annotated by human. From the processing time, the keywords fields are usually very long and very informative. I wonder if it is not the wikipedia data set, will the result still be this high? Because many real world web pages do not have keywords metadata fields to indicate the content in the document. However, the relevance did not show the greatness as my expectation. It might be keywords don't always represent the real situation of the information of documents.

For URL and inlink, wikipedia url usually contains the title info for the documents, so the relevance seems similar to keywords and title. However, for inlink, the performance is extremely bad. It might be because the inlink does not provide accurate informations of inlink. Also it might be forgotten by most people in their search. The relevance here proves that it might not be proper in use.

5 Experiment 4: Sequential dependency models

Describe how you set the weights for the different components of the sequential dependency model.

Query #1: Provide your structured query for query "obama family tree"

1:#WEIGHT(0.8 #AND(obama family tree) 0.1 #AND(#NEAR/1(family tree) #NEAR/1(obama family)) 0.1 #AND(#UW/8(family tree) #UW/8(obama family)))

Query #2: Provide your structured query for query "uss yorktown charleston sc"

56:#WEIGHT(0.8 #AND(uss yorktown charleston sc) 0.1 #AND(#NEAR/1(charleston sc) #NEAR/1(yorktown charleston) #NEAR/1(uss yorktown)) 0.1 #AND(#UW/8(charleston sc)

#UW/8(yorktown charleston) #UW/8(uss yorktown)))

	Indri BOW	BM25 BOW	Indri SDM 0.8+0.1+ 0.1	Indri SDM 0.8+0.15 +0.05	Indri SDM 0.75+0.2 +0.05	Indri SDM 0.7+0.2+ 0.1	Indri SDM 0.7+0.2 5+0.05
P@10	**0.5700**	**0.4967**	0.5700	0.5867	0.5867	0.5900	0.5867
P@20	**0.4950**	**0.4683**	0.5200	0.5150	0.5167	0.5167	0.5133
P@30	**0.4444**	**0.4178**	0.4789	0.4722	0.4744	0.4744	0.4778
MAP	**0.5146**	**0.4782**	0.5394	0.5408	0.5531	0.5533	0.5481
num_q	**30**	**30**	30	30	30	30	30
Time	**19165**	**45002**	53173	46865	51692	50154	49559

Each query for the Indri SDM has three parts: the original, NEAR/1 for 2-grams and UW/8 for operations. The unodered size are set manually by me. It might be proper from the perl script to do that.

The parameter set are shown in the table. For the query, the original takes the most weight because they have the whole query and from previous tests they can return enough number of result to ensure the tests.

Comparing the first and the second, it shows that, larger near operator and smaller UW weight gives it more MAP.

Comparing the second and the third, it shows that if Near operator takes more weight than the original, the result will be more precise.

Comparing the third and the fourth, it shows that #UW takes more MAP than the original query given a set near weight.

Thus, it is clear that #NEAR > #UW > #Original query. Thus, I guess the last one should the best because it meets all the requirement condition. However, the result doesn't go as expected as shown above.

Therefore, the relationship between the three can be really complex and is not decided linearly.

I can infer the performance relies much on the first part. If a document is highly relevant, the second and third parts may truly help it to stand out, comparing them with original query.

I never try to assign the 3rd part a larger number than the second, since I think the third part may produce some redundancy, making some relevant documents (who are not the most relevant) increase dramatically, causing the mapping rate to drop. Documents that fit NEAR are in a way more accurate than unordered window.

6 Experiment 5: Multiple representations + SDMs

State which representation you used from Experiment 3, and why you made that choice.

Query #1: Provide your structured query for query “obama family tree”

1:#WEIGHT(0.9 #WEIGHT(0.7 #AND(obama family tree) 0.2 #AND(#NEAR/1(family tree) #NEAR/1(obama family)) 0.1 #AND(#UW/8(family tree) #UW/8(obama family))) 0.1 #AND(obama.title family.title tree.title))

Query #2: Provide your structured query for query “uss yorktown charleston sc”

56:#WEIGHT(0.9 #WEIGHT(0.7 #AND(uss yorktown charleston sc) 0.2 #AND(#NEAR/1(charleston sc) #NEAR/1(yorktown charleston) #NEAR/1(uss yorktown)) 0.1 #AND(#UW/8(charleston sc) #UW/8(yorktown charleston) #UW/8(uss yorktown))) 0.1 #AND(uss.title yorktown.title charleston.title sc.title))

The following results used 0.7 + 0.2 +0.1 from the Experiment 4 since it turned to be the best parameter pari in use.

	Indri BOW	**Results From Experiment 3 (Title Field)**	**SDM**	**Weight: SDM= 0.60 Exp3= 0.34**	**Weight: SDM= 0.70 Exp3= 0.30**	**Weight: SDM= 0.80 Exp3= 0.20**	**Weight: SDM= 0.90 Exp3= 0.10**
P@10	**0.5700**	0.4333	0.5900	0.5700	0.5600	0.5767	0.5967
P@20	**0.4950**	0.3967	0.5167	0.4900	0.4933	0.5133	0.5183
P@30	**0.4444**	0.3400	0.4744	0.4644	0.4744	0.4789	0.4767
MAP	**0.5146**	0.3287	0.5533	0.5436	0.5440	0.5498	0.5611
num_q	**30**	30	30	30	30	30	30
Time	**19165**	3315	50154	51915	54127	51198	50322

	Weight: SDM= 0.95 Exp3= 0.05
P@10	0.5933
P@20	0.5200
P@30	0.4767
MAP	0.5557
num_q	30
Time	53082

I used 0.7 + 0.2 +0.1 from the Experiment 4. to construct the experiment.

And also I chosed the title field which had the highest MAP rate from Experiment 3. I changed their

weight as shown in the table. Only two last sets outperform the SDM in Experiment 4. The more weight a SDM takes, it is closer to the original weight side. Title fields, as discussed before, may promote the documents' relevance if there is a match between them.The parameter choosing shows that 0.90/0.10 has a best peak result. 0.8/0.2 has the very similar MAP as it was in the original query. However, the P@10 and P@30 have more even distribution. A few documents decrease in their rank because their bad performance in title fields.

This is true for all the queries with different weight. The most relative ones (P@10) drop and P@20 and P@30 increased. From this, I guess more documents, which are out of the 100 in the rank can be taken from the list.

7 Analysis of results

Different Queries Forming

Bag of Words: Directly use the words in the query to form the query. Each term in the query has the same weight, and their position of each other does not affect the result. This way the query is fast to run and to compare to the final result. With proper parameter, BM25 and Indri could have promising results. Setting BM25 k1=1 b=0.5, although it was not normal, it was still

Multiple Representations: The documents' structure is considered in the query. Need to guess how well different part is informed. Originally each of different fields is much worse than **"**body" since body is the source of information. However, the running time is much shorter than usual ones since the fields usually have quite a few terms and not all the documents contain these fields. Again, web pages and documents are not so well organized so that the method in real environment is unpredictable. The most possible usage is to combine them with body field with a #AND or #WEIGHT to adjust the original bag of words method and to have a higher P@10 and P@20 rate.

Sequential dependency models: This method relies on the basis of bag of words and create a sequential dependency on neighbouring words. This takes into account the location neighbouring words and also the unordered location is also taken into consideration. My tests show that the determining factor is still the bag of words method. However, NEAR and UW behind can truly improve some performance and increase the relevance of documents have exact matches but fewer appearances. This method relies on the choice of parameter to have the best results. This, however, cannot be predefined without the knowledge of what queries like. This is a drawback of the method.

BM25 and Indri

Both the BM25 and Indri methods are more effective than Ranked Boolean. BM25 runs faster than Indri since BM25 sets 0 in default score and Indri has to calculate a default score for future use. When combining score lists, SUM could directly use #OR form since default score is zero and added found score

positions into list. However, the #AND in Indri is quite different although they both combine the score lists. I'll discuss it following.

In running time, the above difference is more obvious when query is longer and more complex. #AND operator could be viewed as a special term of #WEIGHT operator to perform. The weight is normalized to ensure three terms and four terms of different queries does not show a difference because of the length of queries.

In BM25, idf shows how rare the term is to give the term a proper weight. Also, a single document's weight is also adjusted by b. In Indri, with two stage smoothing, term frequency accounts for the terms in the documents, with variable mu changing the avg_len weight. And MLE show the sparseness of the term in the whole data set. As can be shown before, ctf is a more proper smoothing method since it truly reflects the term distribution in the whole index, reducing the bias by df.

Query Operators:

#SUM by BM25 Operator. It basically works on the score lists.. SUM works like #OR in ranked boolean, however, the score calculation is quite different as mentioned above. The combination of score list simply added up the term's existing scores together. #SUM could not embed other queries who return inverted list. Also, #SUM takes each term in the query equally. But #SUM, (basically it is the same as discussed in BM25 since BM25 only has one operator) can embed #WEIGHT to do that.

#AND and #WEIGHT are actually same operator (at lease I implemented this way), just set each term in #AND with equal weight, and it can run in #WEIGHT and return the designed results. #WEIGHT takes weight and made it quite flexible to adjust to achieve the best results on training data set. The time consuming is its weakness because it would run quite long.

For parameters setting:

I have no clear idea how to set the k1, mu, lambda, etc but only to keep trying. Sometimes it shows a trend but many more other times the result and the parameter have vague relationship. Only by researching into specific queries can this be explained and addressed properly.

Generally, the combination of SDM and Fields perform the best in all my trying. The parameters still can be further optimized with the information about these documents.

Combining Experiment 2,3,4, to generate Exp 5. However, the best setting of parameters in each part do not guarantee a best final evaluation result although they are quite near in number.

8 The software implementation

I expand the implementation in homework 1 to do homework 2.

Firstly, add the required feature in QryEval to read the parameters in set and make the parameter global. Also, change the "isRanked" boolean to 4 different flags to decide the proper retrieval methods. Also it is global. Add default score to inverted list so that they can be accessed if the document is absent from the

list.

Secondly, change the Praser to enable to parse #weight operators. Add scoring methods in QryopScore to calculate scores based on global flags.

Thirdly, modify NEAR and UW to support fields. Also, implement the UW operator. Use similar results comparison methods. But this time all the lists are compared together at one time and a final result returns that.

Implement #SUM based on #OR. Since BM25 has a default score = 0, it only needs to change score when matched and do the summing up or added into the list without any change. #SUM thus creates the operator and directly pass them to #OR. However, the flag is a parameter of function call.

For UW, you have to keep them all of the results list of each term to do the comparison. This is quite low efficient and consumes many memory space until it finishes scan all the lists. Yet the time is O(n1+n2+n3….). #SUM runs in O(n1+max (union(n1+n2), n3) .. ...) theoretically same scale but little shorter since some items are combined. UW uses an array to store where each point points to, A list of Results containing all the results by each query term. Also, two value of maxPos and minPos to match whether they are in the window.

QryopIndriAnd performs #AND in indri. Since all the possible documents are included, it works like #OR despite its name. However, it was very difficult to modify #OR to achieve this.

QryopWeight is the operator that QryopIndri uses. It embeds an qryopWeight and assigns each term with equal non-zero weight.

Weakness: Too many global settings to avoid the passing of parameters. I decrease the use of global flag as much as possible. However, still there some unavoidable to use. However, on the other side, it simplifies the program. Also, to decrease the workload, I just implement #SUM using #OR. If futural #OR is needed, it would be trouble.

One more thing I would like to mention is that, the printing format will affect the test result, which was quite strange. I used format fprint (“%.1”) at first, and my baseline results are always a little bit different from those of others. Then I changed it to fprint (“%.5”) **after sorting**, and it increases and the result matches others’. It was quite strange since all the results are ranked with their original scores and even the results don’t change in order, the MAP may change. I do believe there might be some issue with the trec_eval service.