Project 2 - Search Engine Two-Stage Retrieval

CS 4300 Information Retrieval

Group members: Jonya Chen (jc957), Yuxiao Tan (yt358), Valerie Hu (vh88)

due December 4, 2014

Part 1: Pseudo-relevance Feedback

(a and b).

In Part 1, we wanted to retrieve 100 new documents for each query. We first had to expand and reweigh each query before running on the collection to retrieve documents. In order to do this, we used Rocchio Feedback.

First, we used our atc.atc weighted inner product similarity from Project 1 in order to retrieve our top 7 documents per query. We took these 7 documents and assumed that they were all relevant, so that we could use them later on. For each query, we then wanted to expand each one by K terms (depending on the situation specified). We did this by using pseudo-relevance feedback search and chose the terms with the highest Rocchio weights, which would then be added onto our new expanded query. Using the new expanded query, we then ranked all the documents again in order to retrieve the top 100 documents. We did this by using inner product similarity between the new Rocchio weighted query vector and each document vector.

(c). The MAP scores for each run are as follows:

With Rocchio parameters A=4, B=8, C=0 (Part 1a):
MAP measure for CACM docs - Part 1a implementation: 0.2763414253728912
MAP measure for MED docs - Part 1a implementation: 0.5646240360290784

With Rocchio parameters A=4, B=16, C=0 (Part 1b):
MAP measure for CACM docs - Part 1b implementation: 0.24362132350209537
MAP measure for MED docs - Part 1b implementation: 0.5705236073595864

(d). We count the number of queries that improved and got worse after the Part 1a and Part 1b implementations. We present these findings below:

	# Queries Improved (CACM	# Queries Improved (MED	# Queries Worse (CACM	# Queries Worse (MED	# Queries No Change (CACM	# Queries No Change (MED

	Docs)	Docs)	Docs)	Docs)	Docs)	Docs)
With A=4, B=8, C=0 (Part 1a)	16	20	33	10	3	0
With A=4, B=16, C=0 (Part 1b)	11	22	39	8	2	0

(e).

CACM, With A=4, B=8, C=0 (Part 1a)

After expanding each query in the CACM collection by 5 terms, we found that the MAP value of the collection decreased from 0.30481006937910665 originally to 0.2763414253728912. Of the 52 queries in the CACM collection, 16 queries had improved MAP values after expanding by 5 terms, while 33 queries had worse MAP values. 3 queries kept the same MAP values after expanding by 5 terms. This overall decrease in MAP values could be due to a variety of factors, such as the volume of documents in the CACM collection. It could be the case that the large volume of 3,204 documents in the CACM collection resulted in a poor selection of the top 7 relevant documents to begin with. Due to the multitude of documents in the CACM collection, the top 7 documents selected may not have been the most relevant documents to the query, which impacted the results of the Rocchio weightings and resulted in lower MAP values for the new queries.

CACM, With A=4, B=16, C=0 (Part 1b)

After expanding each query in the CACM collection by 10 terms, we found that the MAP value of the collection decreased from 0.30481006937910665 originally to 0.24362132350209537. Out of the 52 queries in this collection, only 11 queries had improved MAP values after expanding by 10 terms, whereas 39 queries had lower MAP values, and 2 queries experienced no changes in their MAP values. The overall decrease in the MAP values for the collection could be caused by the large size of the CACM collection. The original top 7 documents that are used to calculate the final Rocchio weightings might not have been the most relevant documents to start off with, resulting in lower tf*idf calculations that contribute to inaccurate Rocchio weightings. This could lead to terms being selected that are not the most relevant to the query, which would cause the queries to be expanded by terms that are not the best choice. Then, the MAP values would be lower than the original MAP value for the collection.

MED, With A=4, B=8, C=0 (Part 1a)

After expanding each query in the MED collection by 5 terms, we found that the MAP value of the collection increased from 0.4978616753154456 originally to 0.5646240360290784. Out of the 30 queries in the collection, 20 queries experienced improved MAP values after expanding by 5 terms, while 10 queries had worse MAP values after expanding. Overall, the collection had an increase in the MAP value, likely because adding the 5 terms to the query made the query more specific, and was able to better retrieve relevant documents in accordance to the terms in the query. Also, because the MED collection has a smaller volume of 1,033 documents, the original top 7 relevant documents used could provide higher tf*idf values that result in higher or more accurate Rocchio weightings that allow the appropriate

terms to be selected to be added to each query, resulting in higher MAP values for the queries and collection as a whole.

MED, With A=4, B=16, C=0 (Part 1b)

After expanding each query in the MED collection by 10 terms, we found that the MAP value of the collection increased from 0.4978616753154456 originally to 0.5705236073595864. Out of the 30 queries in the MED collection, 22 queries had improved MAP values after expanding by 10 terms, while 8 queries had decreased MAP values after expanding by 10 terms. Most queries, as well as the overall MED collection, likely experienced increases in MAP values because the expansion to 10 queries allowed the queries to become more specific, which increases the chances of retrieving documents that satisfy the query requirements. Furthermore, because the MED collection has a smaller amount of documents, the original top 7 documents selected could have had higher chances of actually being relevant to the query, meaning the tf*idf values for those documents with respect to the terms could have contributed to more accurate Rocchio weightings that allowed the appropriate 10 terms to be selected to add to each query. These well-selected terms would then contribute to the queries receiving higher MAP values.

(f).
We chose to analyze query 3 from part a.

Original MAP CACM Score: 0.0375
Expanded MAP CACM Score: 0.0105

We see that the relevant files are:
CACM-1134, CACM-1613, CACM-1807, CACM-1947, CACM-2290, CACM-2923

We observe the old top 7 docs are:
CACM-1768,CACM-1304,CACM-1154,CACM-2858,CACM-1134,CACM-3189,CACM-1496,

We observe the new top 10 retrieved files as follows:
CACM-1768,CACM-2858,CACM-2921,CACM-2179,CACM-1825,CACM-1737,CACM-1154,

Original Query Vector (dropping zero'd values)
languag:0.8960980981869799
target:3.0285712526925375
intermedi:2.301572524756275
construct:1.3689719402557934
multi:2.090719159441382
compil:1.233850900875701

Expanded Query Vector
compil : 1.6968314643722855
construct : 1.2359002771640992
intermedi : 2.8919245337176194
languag : 1.324974319889586

target : 3.547376405131093
multi : 1.1654614256085971
tcoll : 0.0
machin : 1.2812678517173521
length : 1.8467743412603765
address : 2.205788814066656
parser : 0.5102003524888697
grammar : 1.0210833140434088

As we see above, with the newly expanded query vector, we have a new set of weights that represents their atc values in the terms. For a normal Rocchio algorithm, we maximize the difference between the average vector representing the relevant documents and the average vector representing the non-relevant documents. With the C value set to 0, we are actually just doing the former, which is maximizing the difference between the average vector of relevant documents.

To further analyze why the expanded query vector result in such a detriment, we must look at the list of files that were returned. We argue that there are a set of terms, the newly introduced onces, which detract from the meaning of the files that originally are deemed as the relevant terms.

In the original (non-expanded) query, we actually retrieved one relevant file (CACM-1134) in the top seven. This did not happen in the expanded version of the query. When we expanded, we added language like "address", "parser", and "grammar", which do not belong to the document CACM-1134 at all. As such, we actually detracted from the original intention of the query, and moved towards another direction completely. This is the fallacy of using Roccio weights and blind/imperfect feedback mechanisms. We are prone to files, which originally were picked up and intended to be viewed, to be moved away from the intention by the expansion of the query to include more non-relevant terms.

Lastly, at the same time, the weighted values of the original queries actually all drop as well. Which means that the overall "meaning" of a successful query is detracted from its original contents of the intention. As such, we conclude that it is reasonable that the final resulting MAP value from the 3rd query to be 3 times worse (0.03 -> 0.012), as the terms which were expanded upon were all pretty bad.

(g). We ran it multiples times and came out with these numbers being the best:
A = 10, B = 3, C = 0. We got a MAP value of 0.587, which is about 0.2 better than the original.
A = 7, B = 2, C = 0, We got a MAP value of 0.576, which is about 0.1 better than the original.

We hypothesize that the difference is due to dropping the value of A while increasing the value of B (but not incredibly significantly to completely out shadow A).

The A parameter in the Rocchio algorithm places emphasis on the original query, while the B parameter of the Rocchio algorithm places emphasis on the retrieved documents. In part f we showed that simply because one can expand the query does not indicate that one can utilize that query correctly. It is relevant to say that expansion of a query may lead the results of a retrieval in the incorrect direction altogether.

As such, when more emphasis is placed on the original query (A parameter higher), we tend to see a slightly better output than having a larger perspective on the B parameter. This is can also be attributed to the fact that sometimes the retrieved documents are a lot more skewed, as in they are much less informative than necessary, and they would lead us down the wrong path of calculations.

Part 2: Complete Link Clustering

(a and b).

In sections A and B of Part 2, we perform complete link clustering on the top 30 documents retrieved for each query and terminate clustering after we have received the specified number of clusters, as defined by the variable K. Our goal in Part 2 is to use clustering to re-rank the top 100 documents for each query to hopefully improve precision in each run.

To cluster documents, we use the complete link clustering method, which calculates distances between clusters based on the maximum distances between documents in the clusters being compared. These distances are calculated based on the 1/dot_product formula given, where the dot_product is the dot product of two document vectors. Clusters are then formed in an order based on which clusters have the next lowest cost to form. Once we have reached K clusters, we re-rank the top 100 documents, or essentially, the top 30 documents, for each query. Documents in the same cluster are assigned the same similarity scores; this similarity score is either the highest similarity score of all the documents in each cluster, or the average of the similarity scores of each document in each cluster, as specified by Parts 2a and 2b. Documents are re-ranked based on these new similarity scores, and then the MAP scores are calculated on these re-ranked documents to determine whether performance has improved.

(c). The MAP scores for each reranked run in the collection are as follows:

With K = 20 clusters (Part 2a):

MAP measure for CACM docs - Reclustering with 20: 0.3053443310849098
MAP measure for MED docs - Reclustering with 20: 0.5128289728472969

With K = 10 clusters (Part 2b):

MAP measure for CACM docs - Reclustering with 10: 0.29084993465247655
MAP measure for MED docs - Reclustering with 10: 0.4997149069235257

(d). We count the number of queries that improved and got worse after clustering to 20 clusters, and after clustering to 10 clusters in each of the libraries. We present these findings below:

	# Queries Improved (CACM Docs)	# Queries Improved (MED Docs)	# Queries Worse (CACM Docs)	# Queries Worse (MED Docs)	# Queries No Change (CACM Docs)	# Queries No Change (MED Docs)
From no clusters to 20 clusters	21	20	24	10	7	0
From 20 clusters to	17	14	33	16	2	0

10 clusters						

(e). **CACM, K = 20 Clusters**

After reducing the top 30 documents to 20 clusters in the CACM collection, the MAP value of the CACM collection increased from 0.30481006937910665 originally to 0.3053443310849098, after the clustering. Of the 52 queries in the CACM collection, 21 queries had higher MAP values after the clustering, while 24 had lower MAP values. Seven queries experienced no changes in the MAP values pre- and post- clustering. Reducing 30 documents into 20 clusters is probably a good size to improve the rankings of the documents to more accurately reflect the relevance of each document to the query. Prior to forming 20 clusters, some documents may have been ranked in ways that do not represent the document's actual relevance to the query. As a result of clustering to 20 clusters, these inaccurately low-ranked documents are now grouped with documents of similar relevance, and all these documents within the same clusters are re-ranked with a more accurate ranking, that is more representative of the documents' relevance to the query. Also, at a size of 20 clusters, these 30 documents are not yet being forced into clusters that do not make sense, simply to meet the 20-cluster size maximum. Thus, the documents within each cluster are probably actually relevant to each other, and so, the re-ranking of the documents is likely to result in a higher MAP value for the CACM collection. Even if there were more queries that had lower MAP values after the clustering than queries with higher values, the amount by which the improved queries increased was likely greater than the amount by which the non-improved queries decreased; therefore, the MAP value for the overall collection still increased from 0.30481006937910665 to 0.3053443310849098. Also, when we stopped clustering at 20 clusters, we assigned similarity scores to documents based on the highest similarity scores in the clusters, which could have allowed documents that were not previously recognized as relevant, to be more accurately ranked as relevant after being assigned a new ranking after forming the 20 clusters. This would contribute to a higher MAP value for the collection.

CACM, K = 10 Clusters

The MAP value for the CACM collection decreased from 0.3053443310849098 to 0.23041363204817988, after reducing the cluster size from 20 to 10 clusters. This is likely because clustering to form 10 clusters could be too forced, and could lead to otherwise irrelevant documents being grouped together simply to reach the termination point of 10 clusters. Then, inaccurate similarity scores are assigned to documents that do not belong in the same cluster, and the re-ranking of the documents is not actually helpful in retrieving a greater number of relevant documents first. In the CACM collection, only 17 queries had improved MAP values after reducing to 10 clusters, while 33 queries had lower MAP values after reducing to 10 clusters. It follows that the overall CACM collection would have a lower MAP value after the reduction to 10 clusters. In addition, we assigned similarity scores to documents based on the average similarities of documents in each cluster, instead of assigning scores based on the highest similarity within the cluster. Using the average similarities could have resulted in truly relevant documents being assigned lower ranks, because the relevant document's score was weighed down by the similarity scores of other documents in the same cluster that are not relevant. This would result in a poor re-ranking of the original 30 documents, and a lower MAP value for the collection.

MED, K = 20 Clusters

After reducing the top 30 documents to 20 clusters in the MED collection, the MAP value of the MED collection increased from 0.4978616753154456 originally to 0.5128289728472969, after the clustering. Of the 30 queries in the MED collection, 20 queries had higher MAP values after the clustering, while 10 queries had lower MAP values after the clustering. As mentioned in the evaluation for reducing the CACM documents to 20 clusters, the size of 20 clusters is a good size because the documents within each of these 20 clusters are probably relevant to each other, and are not forced into the same clusters simply to meet the 20-cluster size maximum. Because the majority of queries had improved MAP values after the clustering, it makes sense that the MED collection as a whole also had an increased MAP value after reducing to 20 clusters. Also, we assigned similarity scores to documents based on the highest similarity scores in the clusters, which could have improved precision because documents that were not previously recognized as especially relevant, may have been appropriately clustered with documents that were already recognized as relevant. Then, all documents in the same cluster would be assigned the same highest similarity score, which could lead to an improved ranking of the original 30 documents, which could lead to a higher overall MAP value for the MED collection.

MED, K = 10 Clusters

The MAP value of the MED collection decreased from 0.5128289728472969 to 0.4895644024598843 after reducing the collection from 20 clusters to 10 clusters. Of the 30 queries, 14 queries had improved MAP values, while 16 queries had poorer MAP values after reducing to 10 clusters. Reducing to a size of 10 clusters may have resulted in forced clusters that do not actually contain documents that are the most relevant to each other. Documents may have been grouped together just to satisfy the 10-cluster size maximum. Therefore, the re-ranking of the 30 documents after reducing to 10 clusters was probably not any better than the ranking of the 30 documents immediately following the reduction to 20 clusters, and thus resulted in an inaccurate distribution of similarity scores for the documents, which resulted in a poor re-ranking, and a lower overall MAP value for the MED collection. Furthermore, we assigned similarity scores to documents in each cluster based on the average similarities of each document in each cluster, which may have caused relevant documents to receive lower similarity scores than they deserved due to the scores of other irrelevant documents in the same cluster, which results in an inaccurate re-ranking of the original 30 documents, leading to a lower MAP value for the collection.

(f). Query 2, CACM library, with K = 10 clusters

We chose to focus on Query 2 of the CACM collection, after the top 30 documents have been reduced to K = 10 clusters. Prior to clustering, Query 2's top 30 relevant documents are ranked as follows:

CACM-3078,CACM-2863,CACM-2434,CACM-1364,CACM-0740,CACM-2304,CACM-2250,CACM-0727,CACM-1345,CACM-0047,CACM-0335,CACM-2874,CACM-1046,CACM-0397,CACM-0356,CACM-0971,CACM-0136,CACM-0598,CACM-0497,CACM-2688,CACM-1938,CACM-3086,CACM-2850,CACM-0597,CACM-1333,CACM-1755,CACM-3080,CACM-0276,CACM-0236,CACM-3136

After reducing these documents to 10 clusters, the clusters we had were:

Cluster 0 : CACM-3078,CACM-3086,CACM-3136,CACM-0727,CACM-1364,CACM-0335,CACM-0497
Cluster 1 : CACM-2863,CACM-2434

Cluster 2 : CACM-0740,CACM-1755,CACM-0356,CACM-0136
Cluster 3 : CACM-2304,CACM-2250,CACM-0276
Cluster 4 : CACM-1345,CACM-0971
Cluster 5 : CACM-2874,CACM-0397,CACM-2688
Cluster 6 : CACM-1046,CACM-1333
Cluster 7 : CACM-0598,CACM-0597
Cluster 8 : CACM-1938,CACM-2850,CACM-3080
Cluster 9 : CACM-0236,CACM-0047

After forming 10 clusters, the new ranking of the top 30 documents is as follows:

CACM-2434,CACM-2863,CACM-0497,CACM-0335,CACM-1364,CACM-0727,CACM-3136,CACM-3086,CACM-3078,CACM-0276,CACM-2250,CACM-2304,CACM-0971,CACM-1345,CACM-0136,CACM-0356,CACM-1755,CACM-0740,CACM-2688,CACM-0397,CACM-2874,CACM-0047,CACM-0236,CACM-1333,CACM-1046,CACM-0597,CACM-0598,CACM-3080,CACM-2850,CACM-1938

After the original run, Query 2 had a MAP value of 1.0. However, after reducing the documents to 10 clusters, Query 2 had a MAP value of 0.7777777777777778, a decrease of 0.22222222222 from the original MAP value. This decrease in precision could be due to the fact that documents which were not actually very relevant to each other were forced into clusters just to satisfy the 10-cluster size requirement. Query 2 is "I am interested in articles written either by Prieve or Udo Pooch Prieve, B. Pooch, U." However, taking a more in-depth look at the clusters that have formed reveal that some groupings of documents within each cluster do not quite make sense; documents sometimes do not relate to each other, and also do not relate to Query 2. For example, the documents in Cluster 7, CACM-0598 and CACM-0597, barely relate to each other and barely relate to Query 2. CACM-0598 is written by neither Prieve or Udo Pooch Prieve, B. Pooch, and is instead written by U W. Hicks, which means that CACM-0598 is not very relevant to Query 2. Similarly, CACM-0597 is written by none of the authors specified by Query 2, and is instead written by J. C. Emery. Thus, neither of these documents is particularly relevant to the query. Examining the content of CACM-0598 and CACM-0597 also shows that these documents were probably just forced into the same cluster, as their content is quite different. CACM-0598 is titled "The COBOL Librarian - A Key to Object Program Efficiency", and discusses the creation of a "well-constructed COBOL Library" to create an efficient object program. Meanwhile, CACM-0597 is titled "Modular Data Processing Systems Written in COBOL" and contains no further text about the document. While CACM-0598 and CACM-0597 have a few terms in common such as "COBOL", it appears that they have little content relevant to each other and were perhaps forced into the same cluster just because we needed 10 clusters to form. The CACM-0598 and CACM-0597 documents were probably merged together due to the few terms that they share. Although these documents are not particularly similar, they were assigned the same similarity scores based on the clustering method used in Part 2b, which resulted in an inaccurate re-ranking of the top 30 documents, that did not reflect each document's actual relevance to Query 2. Therefore, the MAP value of the CACM collection after the reduction to 10 clusters was lower than the MAP value after the original run, in which dissimilar documents were not forced into clusters simply to meet a certain cluster size requirement.

(g). The cluster hypothesis states that "documents in the same cluster behave similarly with respect to relevance to information need", as defined in the lecture slides. Our experimentation in Part 2 offers

evidence to partly support the cluster hypothesis. To some extent, the cluster hypothesis is supported K when is no lower than a certain K value limit, where K is the specified number of clusters at which to stop merging clusters. For example, we see that after the first iteration of clustering, in which we stop clustering when K = 20 clusters, the MAP values of both the CACM and MED collections increase, likely because the documents within each of these clusters are similar and are similar in terms of their relevance to the query at hand. However, after the second iteration of clustering when we stop clustering when K = 10 clusters, we find that the cluster hypothesis is not necessarily supported. Both the MAP values of the CACM and MED collections decrease after this second round of clustering, likely because 10 clusters is too low of a K value, and results in documents being forcefully merged together, just to satisfy the 10-cluster size requirement. As examined in Part 2f, some of these 10 clusters contain documents that are not very similar to each other at all, which means that these documents probably do not behave similarly with respect to relevance and should not be assigned the same similarity scores, as we did in the clustering method used for Parts 2a and 2b. Thus, it could be that there is a preferred value of K, at which to stop clustering. At this value of K, the documents are joined into enough clusters that make sense and reflect documents' similarities both to each other and to their relevance to queries, while allowing for an improved re-ranking of the top 30 documents for the query.

Part 3: Rocchio Relevance Feedback

(a and b). See code for implementation.

(c). The MAP scores for each run are as follows:

With Rocchio parameters A=4, B=8, C=4 (Part 3a):
MAP measure for CACM docs - Part 3a implementation: 0.35791730599130374
MAP measure for MED docs - Part 3a implementation: 0.5376393074258252

With Rocchio parameters A=4, B=16, C=0 (Part 3b):
MAP measure for CACM docs - Part 3b implementation: 0.3320628883834535
MAP measure for MED docs - Part 3b implementation: 0.540178834116404

(d). We count the number of queries that improved and got worse after the Part 3a and Part 3b implementations. We present these findings below:

	# Queries Improved (CACM Docs)	# Queries Improved (MED Docs)	# Queries Worse (CACM Docs)	# Queries Worse (MED Docs)	# Queries No Change (CACM Docs)	# Queries No Change (MED Docs)
With A=4, B=8, C=4 (Part 3a)	27	19	23	11	2	0
With A=4, B=16, C=0 (Part 3b)	27	20	23	10	2	0

(e).

CACM, With A=4, B=8, C=4 (Part 3a)

After expanding each query in the CACM collection by 5 terms, we found that the MAP value of the collection increased from 0.30481006937910665 originally to 0.35791730599130374. Of the 52 queries in the CACM collection, 27 queries had improved MAP values after expanding by 5 terms, while 23 queries had lower MAP values, and 2 queries experienced no changes in their MAP values. The overall MAP value increase among the queries and for the overall collection was likely, in part, due to the expansion of the queries by 5 terms. These extra 5 terms add specificity to the query to allow for more accurate retrieval of relevant documents, which will result in a higher overall MAP value for the collection as a whole.

CACM, With A=4, B=16, C=0 (Part 3b)

After expanding each query in the CACM collection by 5 terms with these parameters, we found that the MAP value of the collection increased from 0.30481006937910665 originally to 0.3320628883834535. Out of the 52 queries in the CACM collection, 27 queries had improved MAP values after expanding by 5 terms, while 23 queries had lower MAP values, and 2 queries had no changes in their MAP values. Expanding the queries by 5 terms probably contributed to the overall increase in MAP values for the queries and overall collection. These terms further specify the query so that more relevant documents can be retrieved, which will lead to a higher overall MAP value for the collection as a whole.

MED, With A=4, B=8, C=4 (Part 3a)

After expanding each query in the MED collection by 5 terms, we found that the MAP value of the collection increased from 0.4978616753154456 originally to 0.5376393074258252. Of the 30 queries in the MED collection, 19 queries had improved MAP values after expanding by 5 terms, whereas 11 queries had worse MAP values after expanding. Overall, the MAP values of the majority of queries and the overall MED collection increased probably because of the addition of 5 terms to each query. These additional 5 terms further specify criteria for the retrieved documents to match, so now, the retrieved documents are even more likely to be ensured to be relevant to the query. Additionally, the relatively small volume of 1,033 documents in the MED collection could lead to a more accurate selection of the top relevant and non-relevant documents used in this part, which would contribute to more accurate Rocchio weightings, which in turn, leads to more relevant terms selected for the new expanded queries, which leads to higher MAP values for the queries and entire collection.

MED, With A=4, B=16, C=0 (Part 3b)

After expanding each query in the MED collection by 5 terms, we found that the MAP value of the collection increased from 0.4978616753154456 originally to 0.540178834116404. Out of the 30 queries in the MED collection, 20 queries had improved MAP values after expanding by 5 terms, whereas 10 queries had worse MAP values after expanding. Adding 5 terms to each query likely allowed for the MAP values of most of the queries and the overall MED collection to increase, as these additional 5 terms make the query more specific, so that only the most relevant documents are retrieved. The smaller volume of the MED collection could also mean that more precise top relevant and non-relevant documents were selected to use in the tf*idf and Rocchio weighting calculations, which impacted which terms were

chosen to add to the expanded queries. These well-selected terms likely allowed for more relevant documents to be retrieved and for a higher MAP value for the collection to be achieved.

(f). We chose to analyze query 3 from part a, since there is a change in scores from the base run to the expanded query run.

Original MAP CACM Score: 0.0375
Expanded MAP CACM Score: 0.1833

We see that the relevant files are:
CACM-1134, CACM-1613, CACM-1807, CACM-1947, CACM-2290, CACM-2923

We observe the old top 7 docs are:
CACM-1768,CACM-1304,CACM-1154,CACM-2858,CACM-1134,CACM-3189,CACM-1496

We observe the new top 10 (out of 100) retrieved files as follows:
CACM-1134, CACM-1768, CACM-1737, CACM-2061, CACM-2179, CACM-2733, CACM-2921, CACM-1825, CACM-2423, CACM-1523, ...

Original Query Vector (dropping zero'd values)
languag: 2.301572524756275
target: 1.3689719402557934
intermedi: 2.090719159441382
construct: 0.8960980981869799
multi: 1.233850900875701
compil: 3.0285712526925375

Expanded Query Vector
intermedi : 2.401458513416461
target : 5.2590905913332575
construct : 0.9808512086501452
languag : 1.6795344029203902
compil : 1.98739870598720
multi : 1.1654614256085971
tcoll : 1.2812678517173521
machin : 1.0210833140434088
length : 2.205788814066656
address : 1.8467743412603765
parser : 0.5102003524888697
grammar : 1.6968314643722855

From our results we can see that the newly expanded query vector gives us a new set of weights. Similar to Part 1, we are able to maximize the difference between the average vector which includes the relevant

documents and that representing the non-relevant documents. However, unlike Part 1, this time C is not necessarily set to 0, so we are able to maximize the difference in both cases.

The expanded query vector is still not as much of an improvement as expected and we can analyze why by looking at the list of files that are returned. We can then find that we are actually detracting from the original intention of the query, and most like moving in the opposite direction. As a result, our similarity measurements between each document and query are much more skewed or incorrect. Therefore, this new set of terms, which are included in the expanded query, will end up detracting from the original meaning of the files that were initially the relevant terms. Also, similar to Part 1, the overall "meaning" of a successful query is detracted from its original contents of intention as the weighted values of the original queries did drop too. As a result, we can see that the final MAP value when looking at Query 3 are worse when the terms were expanded upon.

(g). Given that we did not do anything special with the two documents whose relevance is known, it may not be fair to look at the MAP change as in part f. The MAP change could be better substantiated if a greater number of documents were referenced and compared with regards to measures such as cosine similarity, as this would reduce the potential for error and result in more accurate and well-rounded weightings that contribute to a more representative MAP value and MAP changes.

Part 4: Retrieving with Document Collection on Disk

(b).

In the original Project 1, for external storage we used a primary key-based, schema-less, multi-dimensional database to hold the data we accrued from the iteration of the files. For internal storage, we used a combination of HashSets, HashMaps, TreeSets, and Heaps for data storage.

External Storage --

In the database, the key would be the original file name (e.g. CACM-1234.txt), with the data being a series of bytes, if interpreted in some set format, would be the multi-dimensional data related to the document. To address the design of our database, it is first necessary to talk about why a direct index store is sometimes more preferred to an inverted index store. Note that the inverted index is very common in discovery and file search.

There are actually two sets of data that needed to be stored into the database. First is the direct index access, from file to the terms and term frequency in the file. The second is the access point from document, to term, to document frequency of said term. If we did not include the second database multi-dimensional structure, there would be no efficient way of calculating the tf*idf values of any term. In fact, without the database, we would either have to hold the entire tf*idf term mapping in memory, or we would have to re-calculate on each usage. Either way is inefficient (time complexity for the latter, space complexity for the former).

In Project 1, we created a separate direct index containing document IDs and term frequency vectors for each term, and wrote this into a CSV file outside of memory to handle a large document collection that could not be brought into memory.We chose to implement a direct index instead of an inverted index, as a

direct index would allow us to better accommodate functionality for relevance feedback, pseudo-relevant feedback, and clustering, which are required for Project 2. Using a direct index allows us to directly access documents to find indexed vectors with the lowest amount of disk operations possible. A direct index also allows us to update information about each document and to update index vectors more quickly and directly than if using an inverted index. Using an inverted index would require a two-step process to update index vectors for each document, which involves determining the number of documents containing each term and finding the size of each inverted list, as well as navigating the order of documents to accurately update inverted lists for each document, whereas using a direct index makes this multi-step process more direct.

Now that we addressed why a direct index is sometimes necessary, we now talk about how the data was store and retrieved, and what the optimal access policy would be for the data.

For our purposes, the access speed of the data is a lot more critical than the write speed (we write once every relaunch of the system, vs we re-read per calculation, otherwise we would be holding all the information in memory). As such, the database should be designed towards such a specification.

In our multi-dimensional database, we key the values on the name of the file, for the fastest retrieval of all information related to the file. Within that, we have the second-layer key which are the terms in the file. This is why the database is multi-dimensional, instead of a flat key-value store. When resolving for the term, it will return the frequency of the term in the document.

So in terms of access policy, all elements are 2 retrievals away from us at all times. That is the design of the database, with the fast access policy. Note that it is initially worrisome that the database access will require a read of a whole file. But with respect to memory handling, spatial locality of file systems, it is very likely that reading the first half of a database value, vs reading the whole multi-dimensional value, will take the same amount of processor time (as the processor brings the other half into L1 cache anyways).

Now that we have addressed the fast-access policy, it is time to address the multi-dimensional feature of the database. We examined a couple of different ideas, from mapping from the primary key to the actual file. But that leaves that we need to re-run the file processing (stemming and stopping, and counting) on every retrieval. The intent of the direct access system was to decrease access time as well as computation, and that is exactly what we would like to avoid. As such, we agreed on a multi-dimensional data storage policy.

We would also like to address why we decided on a schema-less database, instead of something that is more structured, like relational systems. It is to simply data storage space. It is not hard to imagine that across the 3000+ files that we were given, there are at least 1000 unique words across all the files (this is a modest assumption, as longer files can introduce many unique words). If we had a schema based database, that would mean that we would need to have a column for every unique term present in the corpus. In terms of memory, assuming each value (located with the row and column id), would take 3000 * 1000 * 8 bytes = 24,000,000 bytes. Which sums to about 24 megabytes of storage. Now let's do the same set of math with our schema-less database. On average there are about 10 unique words per

document, and we only store those necessary words. That gives 3000 * 10 * 8 = 240,000. Which is 100 times smaller. Had we expanded the database to include more files, the difference would be even more profound.

Note that it is true that a relational schema database has a slightly faster (arguably) access time than non-relational databases. However, with the overhead associated with the relational system, we did not believe it was worth the trade off. Another aspect that we disliked about the relational system was the significant chunk of data that we would have to read per-calculation. To read information on any file, we are set to read a 8000 byte line from the database.

As explored in the BigTable/MegaStore papers by Google, for a spare database, a non-relational schema-less database outperforms the traditional relational database in all manners.

Internal Storage --

Internally, we had a variety of storage structures that acted as a layered cache to make our calculations much easier and much faster. With the fast access origin policy, we still needed to be aware of the difficulties of disk retrieval. Therefore, having a layer of cache values would be incredibly important to the runtime speed of our data systems.

The first thing that came to mind was a Least-Recently Used Cache. However, with respect to how most of the Rocchio, Cluster, and TF*IDF calculations work, it is pretty easy to see that a LRU would be very useless. Most values are read, used once or twice, and then immediately tossed away. As such, we thought that a normal HashSet/HashMap would be good for retrieval and data recovery.

We had several HashMaps, that acted as caches. One of the most important ones was the TF/IDF map. For every term we were currently processing, we read from the secondary database (described earlier as a document -> term -> idf, multi-dimensional map) to find the idf value. Note that this HashMap is cleaned out on each iteration, to reduce the number of files being stored locally by the java program.

Part 5: Miscellaneous

Jonya Chen worked on implementing Parts 1 and 3 and helping write the analyses for Parts 1 and 3.
Yuxiao Tan worked on implementing Part 2 of the project, and debugging and correcting Project 1 functionality to return accurate results needed for Project 2.
Valerie Hu worked on figuring out the clustering methods required for the implementation of Part 2, and writing analyses for Parts 1, 2, and 4.