Assignment 2 Report: Latent Semantic Indexing

Name: Ke Wang NetID: kw427

On the basis of Assignment 1, I implemented this latent semantic indexing program.

1. Mathematical Basis

For LSI, we need to create the term-document matrix, and apply singular value decomposition to create an LSI representation of the matrix. So I use the function from Assignment 1 to read the test files, stop list, analyze them and store them into corresponding data structures. Calculating the weighing tf.idf by function getTF() and getIDF().

Using JAMA package of Java, I can apply singular value decomposition and get the matrices of left and right singular vectors and the diagonal matrix of singular values. And based on the resulting matrices, I can reduce the dimension of the matrices to eliminate noise from synonymy and recognizes dependence. According to the selection of k, I will get new singular values, vectors for later similarity calculation.

To compute the similarity, I first analyze the input query and convert it to the query vector to the LSI space, that is to represent a query as a pseudo document by function getQueryV() and getNewQueryV() and convert document vector in word space to concept space by getDocV(). After that, compute the cosine of the two vectors as the similarity measure and sort the results by cosine value in a descending order. The documents appear first are more relevant.

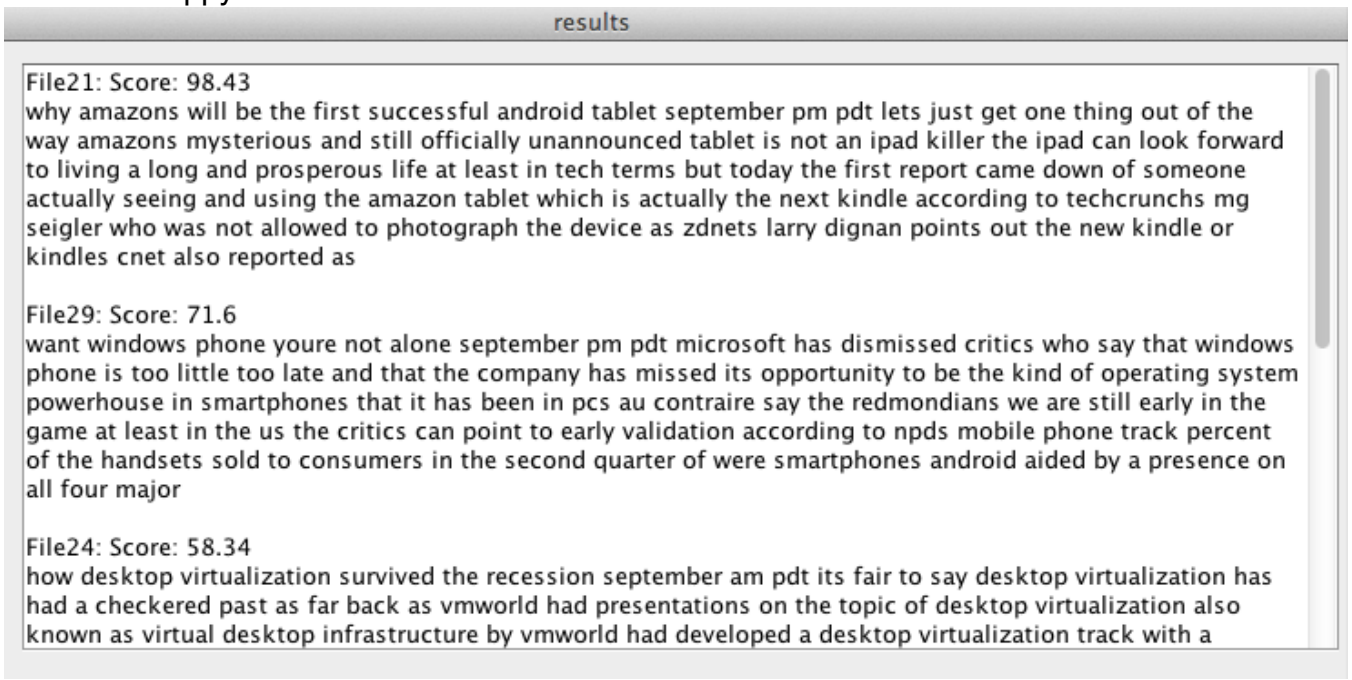
Also, I use cosine*100 as the score for each document, thus documents with high scores are most relevant and I only print the portion of the first 5 documents that are most relevant.

2. Test

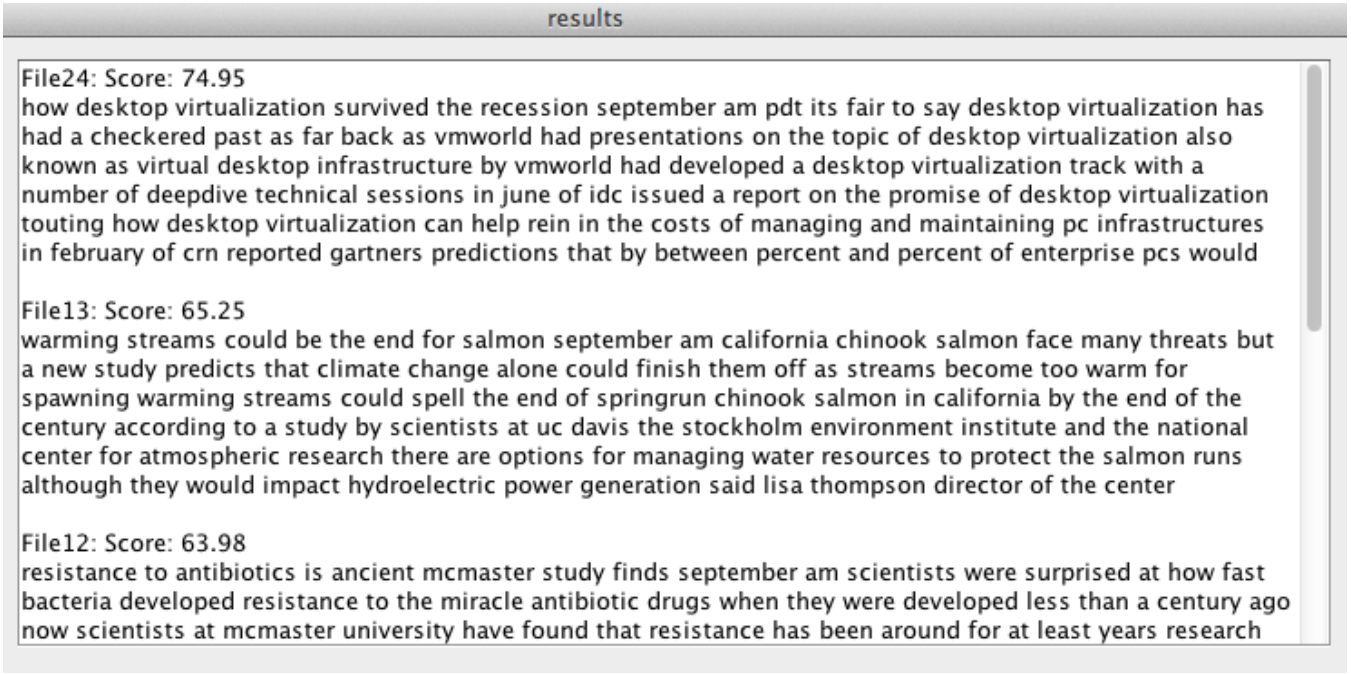
Just run the Test.java and then input the query like Assignment 1.

Case 1: if I use all the words in the stop list or words don't in the corpus,(i.e "have") the program will not get relevant words in the corpus and it will return "sorry, there is no corresponding result".

Case 2: If query "happy summer" and I will get following result. File21 scores highest in the rank and "happy" and "summer" do exist in file 21. The score is close to 100.



Case 3: If the query terms I inputs didn't show in the same document, it will still return the rank of relevant documents where one of them may exist. For example, "had higher". Actually only "had" appears in file 24, but there are 5 occurrences of this word, so the file ranks first. But the score is not as high as previous case because the two words doesn't show in the same document.



3. Varying k (number of singular values)

I use "happy summer" which shows the result as below. We can found that when k is large, the resulting score is not as much high as when it is small. Actually, the rank of relevant documents are not necessarily the same. By properly reducing the value of k, we can eliminate the noise data. After several tests, I found that when k is around 10, we will get more accurate result. For "graphene","kids","engineering", the documents listed will not rank high when k is large. But when k is around 10, the program will retrieve the documents listed.

"happy summer":

k=40	k=10	k=5
file21: 62.42	file21: 98.43	file26: 97.85
file13: 51.47	file29: 71.6	file25: 97.8
file18: 38.66	file24: 58.34	file21: 97.11

"engineering":

k=40	k=10	k=5
file37: 51.34	file32: 98.12	file32: 98.09
file31: 46.69	file30: 94.99	file33: 95.54
file12: 36.3	file33: 93.64	file31: 93.79

Actually, when k = 40, the document 33 ranks almost the last of the list.