Apoorva Arya (500831752)
Felicia Levina (500856106)
Aitirja Chowdhury (500832196)

PROGRAM REPORT

Introduction

For this project, we implemented a document clustering system and used the BBC news dataset provided. The program was coded using Java and major data structures used throughout the program are HashMaps and ArrayLists. The clustering algorithm used to create the document system was k means clustering where k = 5. Below, listed are the three programs used to create the clustering system.

Invert.java (*files*, functions)

The main purpose of this program is to scan the folders in *bbc* and put them in the two text files which are called *dictionary.txt* and *postings.txt*. Just like in the previous two assignments, *dictionary.txt* holds the document frequency of each term and *postings.txt* holds the term frequency of each term in their respective document. At the start of the program, the user is asked if they want to turn stop word "on" or "off". Depending on their answer, it is turned on or off. At the same time, the user is also prompted to enter if they would like to turn stemming "on" or "off", and again depending on their answer, the status of the stemming is changed.

Now, the program scans all the folders inside *bbc* and upon scanning each word in the documents, they are converted to lowercase for easy information retrieval. If the stemming is turned on according to the user, then each word is at first reduced to its root word. Otherwise, the program moves on to the next part of the scanning process which is to put them in HashMaps dictionary and postings. First, they are inserted into dictionary which keeps all the terms with their respective document frequency. If the word exists in dictionary then, the word is retrieved, and the document frequency is incremented. Similar strategy is used to create a file to store the term frequencies with their documents by creating a HashMap called postings. To create a unique key to store the information in postings, a naming convention was assigned where the document ID would start with either "b", "e", "p", "s" or "t". For example, when scanning document *199.txt* in folder *business*, the key in postings would be *b199*. This process is followed for all the files inside *business*, *entertainment*, *politics*, *sports* and *tech*.

Once the scanning is done and the required information is put into the HashMap they need to go into, a for-loop which removes the common words in both HashMaps, if stop word removal was turned on. Once that process is done, the contents of both HashMaps, dictionary and postings is printed on to the files *dictionary.txt* and *postings.txt*, respectively. This program also creates *titles.txt* which stores all the titles of the documents retrieved from the files which will be used in *Cluster.java* when printing out the evaluation metrics information.

Apoorva Arya (500831752)
Felicia Levina (500856106)
Aitirja Chowdhury (500832196)

Search.java (*files*, functions)

In this program, both the text files that were created previously in *Invert.java* are scanned to calculate the vector weights of each document. Everything is similar to how things were done in the second assignment but this time, once the program finishes calculating the weight of each document, it sends the document weights to *Cluster.java*. The weights of each term in their respective documents is stored in a HashMap called, docWeight which is used as the argument when calling the method Cluster in *Cluster.java*.

Cluster.java (*files*, functions)

This program calculates the similarity score of each document and creates 5 clusters which hold documents that are similar to one another. When choosing the initial centroid, the first document of every folder (i.e. *business*, *entertainment*, *politics*, *sports*, *tech*) is chosen. Before starting the first iteration, the 5 chosen centroids are temporarily removed from the docWeight HashMap that stores all the weights of the terms with their respective documents. The first iteration calculates the similarity between the centroid with all the documents and returns the document that has the highest similarity. This process is done through firstIteration() which returns the highest 5 clusters after their similarity score has been calculated. However, before returning the document with the highest similarity, the score is further normalized by dividing the dot product by the magnitude of the centroid and the document vectors. After the normalization, the one with the highest similarity score is put into their respective cluster array list which are cluster1, cluster2, cluster3, cluster4 and cluster5. Once it goes through all the documents, the final 5 sets of clusters are returned. After the first iteration is completed, the 5 documents that were chosen to be the centroids are placed back into docWeight. The next step is to perform 5 more iterations till it converges which is done in otherIteration().

This program also prints out the tightness of each cluster and their purity. The evaluation metrics of each cluster is calculated by initially creating a HashMap called *clusterList*, that stores all the clusters that were collected through firstIteration() and otherIteration(). The tightness between each cluster is calculated in otherIteration() but it is only done after completing the 5^{th} iteration. The distance is measured by retrieving the similarity score of each cluster which were stored in the simScore array list and is subtracted from 1. Then the distance is squared, and the sum of it is assigned to their relative cluster representing the tightness between them.

Apoorva Arya (500831752)
Felicia Levina (500856106)
Aitirja Chowdhury (500832196)

The purity is calculated by looping through the HashMap and incrementing every time a document is encountered in the clusterList that starts with one of the letters which are “b”, “e”, “p”, “s” or “t”. Whenever they are encountered, the count would increment up by 1 suggesting the total amount of document that exists in the cluster so far. After the loop is done, all the counts of the 5 clusters are added in an array list called, countList which is then used to extract the max count of a certain document type. For example, if there is more documents that start with “b” in cluster2, then that would be our max document type from that cluster. Once the count incrementation is completed, the max value is extracted from the *clusterList* and then it is divided by the total number of text files that exist in the *bbc* folder which gives the purity.

Printing Results (*files*, functions)

The major results required for this project are all calculated in *Cluster.java* where *Invert.java* and *Search.java* are the base functions that retrieve the information that we need for the calculations. After *Cluster.java* is terminated, the program prints the evaluation metrics in to *eval.txt* (which has the purity and the tightness of each cluster) and also, it prints the document information; such as the title of each document in the cluster on to *cluster.txt*.