Assignment 2 Report

Cheong Chee Mun Brandon (A0172029J)

Task 1

Implementation Details (More details can be found in comments in the code itself)

Pre processing

1. Inside rawPostings() function, the program loads the raw postings line by line and converts the input into an RDD of Postings.
2. Inside groupedPostings() function, the program will split the postings into questions and answers by filtering based on postingType 1 or 2, creates a pair RDD of the question's id and question or the answer's parentId and answer, and then performs a join on the questions and answers based on their id/parentId.
3. Inside scoredPostings() function, the program finds the max score for each posting ie the max score of all the answers for a question. It does this by iterating through all the groupedPosts from the groupedPostings() function, for each groupedPost which contains multiple pairs of the question itself and an answer to that question, it will iterate through each pair and find the max score of all the answers, and then return a PairRDD of the question posting and the max score.
4. Inside vectorPostings() function, the program maps through the PairRdd of the question posting and max score, and extracts out the domain of the question posting and its index, and outputs a PairRDD in the required vector format of (D*X, S), where D is DomainSpread, X is index of the domain of the question posting, and S is the max score for the question posting.
5. Inside sampleVectors() function, a sample of kMeansKernels = 45 vectors is taken to be the initial centroids out of all the vectorPostings.

K-means computation

1. Inside kmeans() function, the program first calls obtainCentroidWithGroupedVectors(), which will find the closest point for each vector, then group all the vectors with the same closest point/centroid together.
2. Then kmeans() function will then find the new centroid location for each group of vectors by averaging the vectors and finding the average point, and then it will update the centroids with the new centroid location.
3. The kmeans() function will then check if it has converged using Euclidean distance calculation or if it has reached kmeansMaxIterations=120, if so it will terminate, otherwise it will run 1 more round of kmeans().

Post-processing

1. After running finish kmeans(), the program will then run clusterResults() function, which will again run obtainCentroidWithGroupedVectors() to get the final centroids and their grouped vectors, and then return (centroid, clusterSize, medianScore, avgScore) sorted by avgScore for easy visualisation, then it will print the results out.

Final results using predefined parameters

Cluster centroid	Domain of cluster centroid	Domain Size	Median Score	Average Score
(0,0)	Machine-Learning	124563	0	0
(200000,0)	Data-Analysis	98797	0	0
(0,1)	Machine-Learning	104203	1	1
(100000,1)	Algorithm	314828	1	1
(250000,1)	Security	150853	1	1
(200000,1)	Data-Analysis	205215	2	1
(150000,2)	Big-Data	171325	1	2
(50000,2)	Compute-Science	379220	1	2
(350000,2)	Computer-Systems	113843	1	2
(0,3)	Machine-Learning	129777	3	3
(410033,4)	Deep-learning	119603	1	4
(300000,4)	Silicon Valley	55409	1	4
(599430,4)	Cloud-services	32200	2	4
(500000,5)	Programming-Language	13198	3	5
(200000,8)	Data-Analysis	53934	7	8
(250000,10)	Security	28726	9	10
(0,37)	Machine-Learning	6119	32	37
(200000,44)	Data-Analysis	5553	39	44
(150000,54)	Big-Data	3301	44	54
(50000,63)	Compute-Science	4266	52	63
(250000,64)	Security	1729	55	64
(100000,84)	Algorithm	1345	67	84
(0,142)	Machine-Learning	1022	132	142
(200000,149)	Data-Analysis	872	136	149
(250000,259)	Security	186	229	259
(150000,264)	Big-Data	322	227	264
(50000,281)	Compute-Science	436	240	281
(0,359)	Machine-Learning	291	336	359
(200000,399)	Data-Analysis	174	377	399
(100000,446)	Algorithm	82	383	446
(350000,452)	Computer-Systems	139	345	452
(250000,772)	Security	32	639	772
(0,818)	Machine-Learning	100	766	818
(150000,839)	Big-Data	54	736	839
(200000,1033)	Data-Analysis	22	905	1033
(50000,1077)	Compute-Science	34	961	1077
(0,1751)	Machine-Learning	33	1640	1751
(100000,2131)	Algorithm	4	1777	2131
(250000,3636)	Security	2	3636	3636
(150000,3770)	Big-Data	3	3335	3770
(0,5007)	Machine-Learning	5	4441	5007
(50000,10271)	Compute-Science	2	10271	10271

Insights from results

From the domain size, as well as the lack of centroids representing certain domains, one observation is that some domains of posts are more active. The largest domains (greatest cluster size) are from Compute Science, Algorithm and Data-Analysis indicating that these domains are more popular fields being talked about currently, while others like Silicon Valley or Embedded-System don't even make the list, indicating that they may not be so popular topics and may be parked under a centroid from one of the bigger domains.

We can also see that there is a high variation in the median and average scores of posts. In fact, it seems that those clusters with high cluster sizes are the ones with the lowest median/average scores, perhaps indicating that for high cluster sizes, many of the posts in these clusters have low scores, , thus diluting/lowering the overall median/average score for that cluster. One possible reason is that these posts could be very general, thus attracting many answers even from people

who don't know much about the topic. Whereas for clusters with low cluster sizes, these topics could be more specialised, thus attracting answers only from people who know about these topics, thus these answers are rated higher.

Further discussion on system performance

To optimise speed/performance:

1. Caching can be done on RDDs that are frequently iterated over. In this case, kmeans() is the function that keeps getting iterated, so the vectors RDD can be cached using Spark's persist() API, so that the vectors RDD is not recomputed each time the kmeans() function is run.
2. Currently the program is only run on 1 node, but running on multiple nodes can speed up the computation as computation can be done in parallel.
3. A smarter method can be done to choose the initial centroids instead of just randomly sampling a few out of the vectors RDD to be the initial centroids, as this will allow the kmeans() function to converge faster thus improving performance.

It should be noted that modifying the predefined values like kMeansKernels, kMeansMaxIterations etc will also affect performance, though changing these values to increase performance might lead to decrease in accuracy or usefulness of results.

It should also be noted that it is still better to use a kmeans ML library like those from SparkMLLib instead of coding it from scratch, as the one from the SparkMLLib is likely to address some of the above concerns to improve system performance.

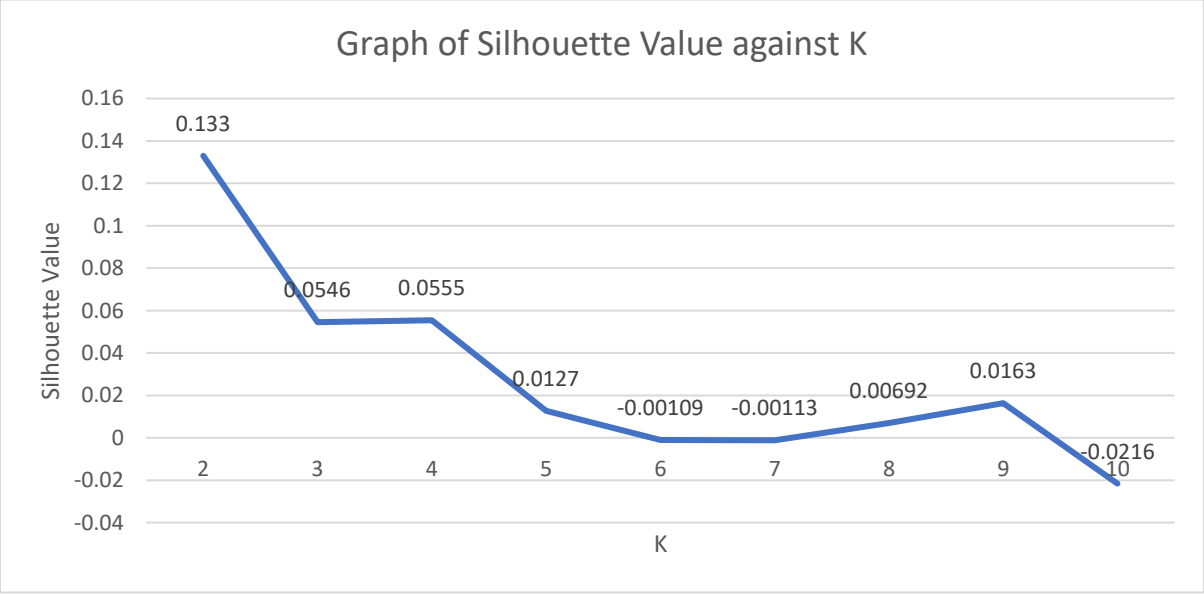
Task 2

Implementation Details

1. The program first reads in the stopwords txt file and the tweets data csv file.
2. The program uses Spark ML Tokenizer to read in the tweets column of the tweets data csv file, (this is column 5 of the csv file), and split the tweet string in each cell by space into words.
3. The program uses Spark ML StopWordsRemover to filter out the stop words from the raw tweet words based on the stopwords txt file.
4. The program uses Spark ML Word2Vec to convert the tweets into a Word2Vec format so that it can be used by the KMeans algorithm, and then caches this word2vecDataframe for use in kmeans to improve performance.
5. The program uses Spark ML KMeans to implement the k means algorithm on the featuresCol set, and then outputs the resulting cluster centers, and also uses Spark ML Clustering Evaluator to evaluate the silhouette value with squared Euclidean distance of the clustering result.

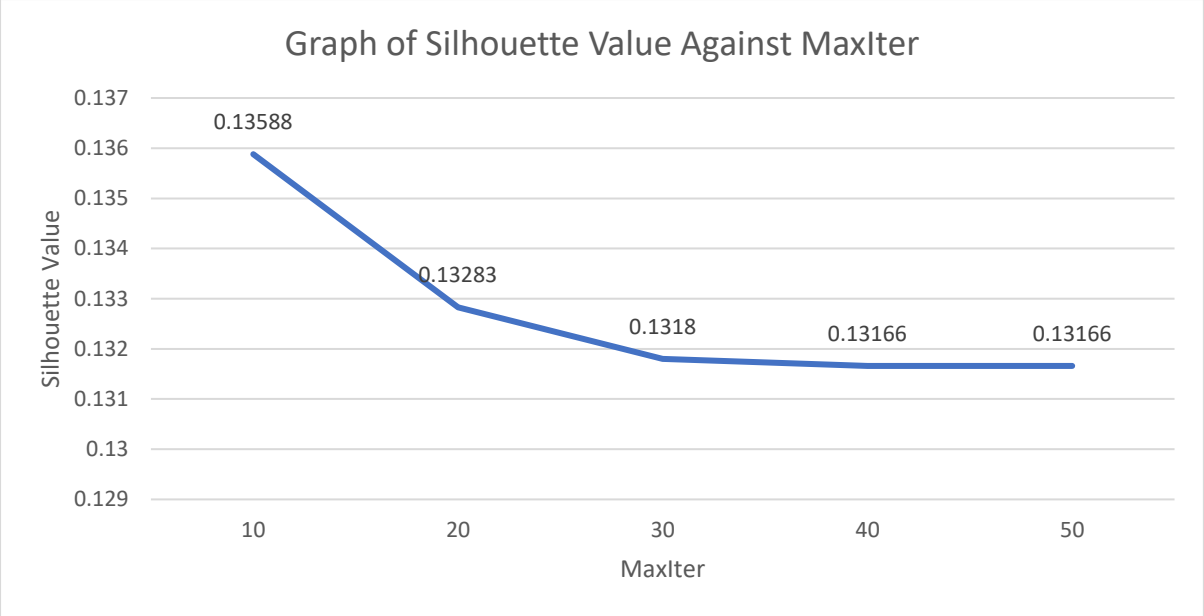
I was stuck at finding the most frequent words of each cluster, hence I didn't have enough time to do that portion.

Description and analysis of results



For silhouette value, https://en.wikipedia.org/wiki/Silhouette_(clustering) states that "a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters."

From the graph of silhouette value against k shown above, it is clear that k=2 is the best k value as it provides the highest silhouette value of 0.133, while the other K values from 2 to 10 give either a lower or even negative k value. Hence, this shows that the tweets data can be optimally clustered into 2 clusters.



From the graph above, it can be seen that a maxIter value of 10 is the best.

Analysis of parameters in kmeans

In terms of k, k is the number of clusters. An optimal k can be found either by doing the silhouette method as shown above, or by another method called the elbow method, which is elaborated in https://towardsdatascience.com/clustering-metrics-better-than-the-elbow-method-6926e1f723a6 and also in the lecture 4 slides, where we plot a graph of kmeans score (kmeans score is some form of intra-cluster distance relative to inner-cluster distance eg average distance to centroid) against number of clusters, and pick the k at the elbow. A too high k value means there is little improvement in average distance with the clusters very close to each other, while a too low k value may mean there's many points that are actually too far away from the centroid.

In terms of maxIter, it determines the max number of iterations run within 1 round, where 1 round is the kmeans() function in task 1. Theoretically, increasing max number of iterations will increase accuracy of results but will also increase the amount of time taken, though increasing it too high will not provide much benefit and will actually provide diminishing returns.