**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.

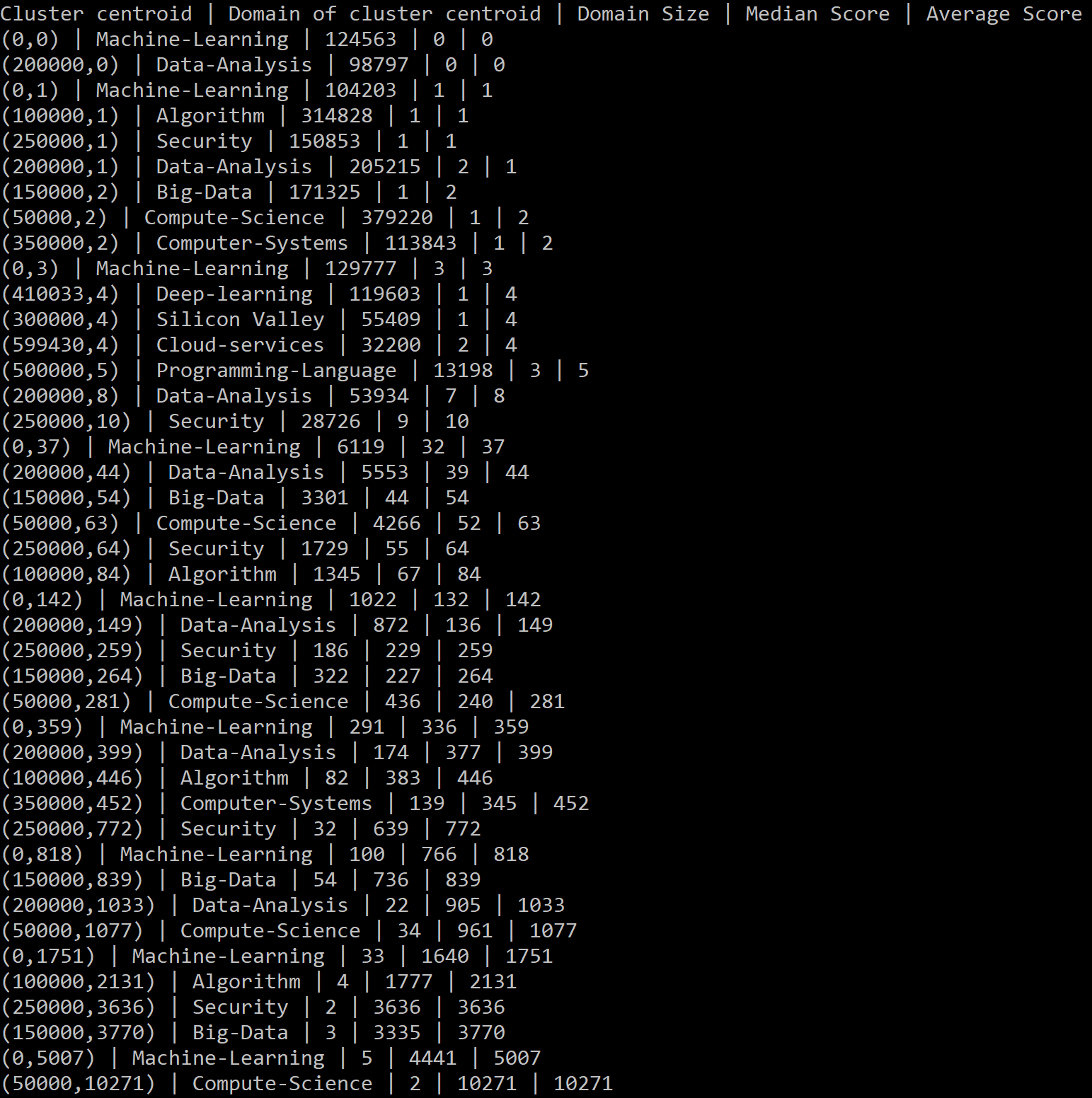
K-means computation

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**



**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.