CS450 Final Project

Katya Griffiths-Julien, Jerry Lau

**Introduction**

In this project, we used an implementation of Minhash and LSH provided by Clark Li from the University of Southern California. Specific to this implementation, Spark on Python was selected to speed up the run time of the similarity search.

The provided dataset from the US Health and Human Services consists of 420k rows on 109 columns representing hospitals during the COVID-19 pandemic. This search would be used to find similarities between the datapoints for the end goal of finding characteristics to coordinate a response to the affected hospitals and to potentially better manage resources should it occur again.

**Data sanitization**

To begin, the provided implementation calls for the use of a txt file which our dataset was not in. Then the next biggest issue was the columns of the dataset were required to be in integers. This is inline with *Task A* where it was required to extract binary features from the dataset. The calls to read the csv dataset and then convert the data in 109 columns over the sample began to show us that many redundant features would not be required to produce a good result.

As such, converting the data to integers to simplify the integration with the Minhash/LSH already placed a constraint that the dataset would be in integer format. Therefore, a binary representation using the integer data type essentially wasted the memory space that is allocated to us. That is why we converted all of the ASCII values of the strings and added them together to create a dataset of integers that would require minimal changes to the Minhash/LSH program. It also makes us of the data type by not wasting space to hold a binary value. Additionally, the dataset in this manner would allow us to properly cluster, group or reduce items in their respective columns.

Lastly, the “hospital\_pk” was changed to use the index number to facilitate hashing calculations. This choice was made since the values in “hospital\_pk” were not all consistently integers and strings were not allowed in the Minhash/LSH program.

**MinHash and similar pairs**

At this stage with the dataset converted, the call to main began the process to execute the hash and similarity search. The call to *get\_set\_signatures* and generate the signatures for each data point using the MinHash for each item in the dataset. A lambda reduction, *groupByKey*, filter and distinct minimize the matrix calculation for the initial hashing of the values. This process was relatively fast to complete. From a quick code analysis, the functions called seem to be in the range of Ɵ(n) or trending towards it.

Then it calls the processed candidate list to *get\_jaccard\_similarity* where each candidate generates a combination signature to calculate the similarity between the resulting candidates. Overall, this section ran into many runtime issues with Google Colab cloud servers either busting the ram capacity or having to reset because resources were allocated elsewhere. When we analysis the code of this function it class for 2 for loops meaning that at a minimum, the complexity will be Ɵ(n2). Then within the function, there is also the use of 3 lists going from 1-n for the data to be stored, analyzed and processed. On top of that, this function is called from the execute where each datapoint candidate must analyzed. Which mean that we have a time complexity of Ɵ(n3) best case scenario.

To overcome this issue, we had to reduce the test dataset so that the similarity search could be accomplished. Next, additional ram provided to the system would be beneficial to the process thereby providing some headroom for the backend to not allocate all resources to this task. Lastly, more CPU threads would help parallelize the processing. In the end, we had to use all 3 methods and run it on a local machine to provide us with a generalized similarity search outcome.

Once all those calculations are completed, a *merge\_result* in *reduceByKey* is used to prune the potential candidates in the similarity search. This process does not take too long to completed compared to the first two tasks. It returns the similarity set dictionary which can be finally outputted to a text file.

**Experimental results**

We have included the output files that were run using a truncated dataset with the number of test items in the title. When we run the MinHash/LSH, we can increase the prospective candidates to adjust the similarities pairs from 10 to 30 and play around with the rows and bands that are setup as global variables at the start of the program. The constraints are once again the hardware that is used to run this program. The main issue with running this program was the unknown factor which the python code is being executed. In the regular mode, the console would show us the steps 1-10 and the number of threads assigned. It would just blink on the screen. This meant that when running the test file, we cannot be sure if the processed crashed, still working on the issue or exceeded memory. Then in debug mode, the console would print out all pairs and other information would just scroll. So this trade off is a hit in performance since the console is not only tasked with calculating the matrixes but also printing the data.

**Possibility of improvements**

As previously noted, the MinHash/LSH take very long to run and it also requires a lot of resources. Optimization for the *get\_jaccard\_similarity* would allow for the progress to be much fast and also less resource dependent. We dared not to push our own local machines further using a docker container running jupyter notebooks on 32GB of ram and 16-24 threads. This is a brute force method which is also highly parallelized meaning that more resources would do the trick. However, this is not optimal because larger and larger datasets would require more and more resources.

A possible suggestion would therefore be better data sanitization where columns such as *hospital\_name*, *address*, *city* could be discarded. Other duplicate values such as *cnn* and *hospital\_pk* are the same. Using other functions to implement the features could also provide better data extraction such as using a logistic function.

The potential use of a streaming data algorithm as per assignment 1 can also be used in this project. The only downside would be the estimation of the dataset. This also means that the tradeoff is additional processing power and less ram resources. It would mean increasing processing time.

**Conclusion**

In this project the use of MinHash/LSH proved to be capable of doing similarity searches. We saw that the use of a very large dataset has it’s own challenges to be first parsed and sanitized then to be run using the PySpark libraries. Due to a resource constraint on our machines, the test dataset was not able to be fully run as it would result in system overload and constant looping.

We would also suggest to use different classifiers to run a similarity search using machine learning. The one that comes to mind is the use of K-Nearest Neighbors. It would allow us to cluster the dataset with a specific K in mind and then the nearest neighbors would be the most similar. Perhaps, preprocessing the dataset using dataset transformations then clustering the associated data points may yield a higher classification score. However, that would be outside the scope of this project where data classification was the primary objective.