CSE 515 Phase 3

Jacob Mims, Nagarjuna Myla, Siddharth Sujir, Tsung-Yen Yu
Professor
K. Selcuk Candan

Abstract

In this project we worked on indexing, graph analysis, and classification. We have experimented on Locality Sensitive Hashing technique and Vector approximation file technique for indexing, page ranking for graph analysis and k-nearest neighbor,decision tree based for Classification. We have used these techniques on a set of simulation files and a given query finding the similar simulation files to the query file also assigning labels to the simulation files.

Keywords:
LSH, L2 hash family, nearest neighbor search, PageRank, Personalized PageRank, Weighted Epidemic Simulation Similarity Graph, classification, decision tree, k-nearest neighbors

1 Introduction

The third phase of the semester project for this course discusses indexing, graph analysis, and classification algorithms. Our team implemented a variety of algorithms to investigate these topics. In Task 1, we cover Locality Sensitive Hashing as a form of indexing and a similarity based search function of that index. In Task 2, we cover a VA-based index structure with it's corresponding similarity based search. In Task 3, we analyze graphs of epidemic simulation data, using weighted similarity and PageRank. Finally, in Task 4, we investigate the k-nearest neighbors and decision tree based classification algorithms in the context of epidemic simulation data.

1.1 Terminology

- **LSH**: Locality sensitive hashing a technique for mutli dimensional index structuring. Which can be used for nearest neighbor search.
- **L**: the number of layers.
- **K**: the number of hashes per layer.
- **VA**: Vector Approximation file.
- **t**: Number of similar simulations needed to be visualized.
- **w**: size of bucket
- **PageRank** : An algorithm that calculates ranking of individual files by going through each of the connected nodes with a pre-determine chance of jumping to any random files.
- **Personalized PageRank** : A subset of PageRank that basically use the same logic to calculate the ranking except that it does not jump to any random files, it only jumps to the files interested.
- **Weighted Epidemic Simulation Similarity Graph** : A graph that consist of epidemic simulation similarity and each edge in the graph is given a weight, which is the value of the Euclidean distance similarity measure.
- **K-Nearest Neighbor Classifier** : Utilizes the k-nearest classified objects to label unclassified objects.
- **Decision Tree Classifier** : Builds a tree from classified objects to use as decision points to classify new objects.

1.2 Goal Description

1.2.1 Task 1

In this task our goal is to implement a Locality Sensitive Hashing technique which will group the similar objects into same buckets or at least into near by hash buckets. Given a L, number of layers of hashes, and K ,number of hash per layer, and a set of word files we need to index the set of vectors and give the size of index structure in bytes. Also given a query file we need to identify the t similar simulation epidemic files, number of unique vectors considered, overall number of vectors consider and the number of bytes of index structure access to process they given query file and out put required results.

1.2.2 Task 2

This task aims at creating a multi-dimensional index structure using the vector file approximation approach. Given a set of simulation files, we need to create an approximation for each of the vectors in the epidemic word file. The index file consists of approximations of the given vector sets. Once the index structure has been created, we implement nearest neighbour search for the given query file. The program returns 't' most nearest neighbouring simulation files.

1.2.3 Task 3

In the first sub-task we are suppose to create a weighted simulation similarity graph, where if the two simulations' similarity are above a user entered threshold then we will create a link between them using that similarity value. For the second sub-task we are asked to identify and visualize the K most dominant files using PageRank(6) with the given simulation similarity graph that we obtained from the previous sub-task. For the last sub-task, it is similar to the previous sub-task, except now we will be doing personalized PageRank(4) on the simulation similarity graph with two query file given as input and asked to identify and visualize the K most dominant files.

1.2.4 Task 4

In this task, we visit object classification through two separate algorithms: k-nearest neighbors and decision trees. As such, we will implement both algorithms in MATLAB and run them on the given sample data. The k-nearest neighbors algorithm will parse the preassigned labels, receive a number of neighbors to examine from the user, and then classify the remaining files based off of voting from the training data. In the decision tree algorithm, we will adapt the concept to fit a multi-dimensional data set. The algorithm will determine the most desirable feature at any given node by applying Fischer's discrimination ratio to the labels and taking the largest of these values. It will then classify all of the remaining files based on their similarity to each set of training data for

each label. It will then complete another pass on files that can fit into multiple classes and try to take advantage of the data garnered from the newly classified files.

1.3 Assumptions

1.3.1 Task 1

We have assumed that w as number of rows in the word file divided by 50 which is large. L2 hash family is used and Euclidean distance similarity measure is used to find the similarity between files. We assumed each row i.e window vector as a dimension.

1.3.2 Task 2

The Task 2 has been developed based on the following assumptions. The epidemic word file has to be generated before executing the task. The space partitioning is done using equal space partitioning approach. The algorithm will create approximations based on the partition in which the vector falls into. The program takes into consideration only the unique vectors of the epidemic word files. In order to return the t most nearest neighbours, the program uses hamming distance measures.

1.3.3 Task 3

For this task we assumed that all input simulation file including the two query files are of type ".csv" and are contained in the same given directory without nesting. We also assumed that the similarity measure we are asked to use is the Euclidean distance similarity and the weights of the simulation similarity graph is the value of the similarity. For second and third sub-task, the paper suggested that a restart probability of 0.8 to 0.9 will return good result, we chose the midpoint, 0.85, as our restart probability. For the final task, we assumed that the given two query simulation files are within the original set of simulation files and they are not considered when choosing the top K dominant files.
.

1.3.4 Task 4

In task 4, we assume that that separation of the epidemic simulation files into distinct labels is possible. If there are many cases where multiple labels can apply to a single file or where the training data does not represent the type of file that is currently being examined , the result will likely be a large class. Since these classifiers depend on the addition of newly classified files for additional comparisons, if a file does not fit well into any particular class, it can cause it to begin absorbing new files into whichever one it happens to fall into. This will cause the class to become diluted, and reduce its discrimination value (3).

2 Proposed Solution

2.0.1 Task 1

For the given L layers and K hashes per layer we have created a hash family which is L2 hash family which uses a stable distribution for projecting data on to a 1-dimensional line. Computed each window vector dot product against a constant vector [1..windowsize] and calculated the weight of that vector. Once all the weights of window vectors are calculated they are taken as the weight across that dimension. Similar way weight across each dimension are calculated for all the word files. This new vector which is weight vector for entire simulation file is projected on to a 1-dimensional space using L2 hash family K times and they are concatenated and L times which are union together. These concatenated K length has code is stored as index and which all file name that generated same hash code are stored in one bucket. This process is continued for L iterations and on all the given set of files. Once our mapping is completed to find the size of index structure just calculate the total size occupied by hash codes and buckets.

Given a query file the same hash family created above is used and mapped in similar fashion. The hash values which got generated here are intersected with the index structure which we have created above. For those intersection part of hash values the objects which are files name are retrieved from the bucket. These files are processed using euclidean distance similarity measure against the query file. These similarities are sorted in descending orders and the top t similarities are taken and visualizes as heat maps. If the total number of files retrieved from the intersected hashes are less than t the files from the near by matching hashes are taken until "t" files are retrieved (Locality Sensitive feature of our hash family got applied here).

2.0.2 Task 2

a) In Task 2a, we get the number of bits per dimension as the input from the user. The number of dimensions is obtained from the size of the window in the epidemic word file. For each dimension, it is split into 2 power b regions. The number of partition will be 2 power b +1 partitions. The regions are split equally as mentioned in the assumptions. Once the regions and the partitions for each dimension has been created, the vector's region is identified and approximation is created for each vector. For each simulation file, the unique vectors are picked and approximations are created for them.

b) Using the index structure created in the Task 2a, we perform the nearest neighbour search on the given query file. An index structure is created even for the query file as per the procedure followed in the previous step. Now we take each unique query vector and we use the hamming distance measure on the query approximation and the each of the vector approximations to measure the similarity. For each query vector we pick the most similar vectors from the simulation file, ie we pick the unique vectors until we get count greater than

equal to 't'. Then we calculate the euclidean distance on the list of files obtained to get the actual similarity measure of those files with the query file. The top 't' most similar files are returned as the nearest neighbours.

2.0.3 Task 3

a) As mentioned in Section 1.3.3, we used the Euclidean distance for our similarity measure. As for the code implementation, we reused the code from Phase 2 Task1a and created a adjacency matrix to represent a graph. Since the requirement asks for a weighted graph, so if the similarity measure value is greater than the given threshold we stored that value instead of just "1" to indicate there's an edge. After finishing computing the similarity measure and creating the adjacency matrix, we return the adjacency matrix and a matrix containing a list of files in the input directory for later use.
b) For this sub-task we followed the algorithm given in (4). We used the pre-calculated graph (G) from Task3a and column normalized it. We also used the suggested restart probability (c) of 0.85. Then we set the restart vector (RV) to be 1 over number of files, initially we also set the steady-state vector (SSV) equal to the restart vector. Then we do the following calculation until steady-state converges.

$$SSV = (1 - c) * G * SSV + c * RV \tag{1}$$

Finally, we picked the top K, given as input parameter, files and printed out the solution and visualized it using heatmap.
c) This sub-task is very similar to the previous sub-task, except instead of doing a regular PageRank, we are asked to a personalized PageRank. For this sub-task, we have two extra input, the two query files. Since we will be doing personalized PageRank, instead of making every entry in the restart vector 1 over number of files, we set all entry to zero except the two query file as 1 over 2. Then we do the same calculation as above until the steady-state vector converges. Finally, we picked the top K, given as input parameter, files and printed out the solution and visualized it using heatmap.

2.0.4 Task 4

a) The implementation of the k-nearest neighbors algorithm was relatively simple. All that was required for this algorithm was to accept a set of epidemic simulation files, a label file, and the number of neighbors to search and then read in each non-labeled data file in sequence. Once the file is read in, we measure the Euclidean similarity between it and all classified files. The top k are taken and votes are tallied. The label with the most representatives in k has the new file classified under it. This task is repeated for each unclassified file. b) The decision tree algorithm was more complex than k-nearest neighbors. For this algorithm, we had to adapt the proposed algorithm to fit our multidimensional data. First, the pre-labled files are classified and made into "leaves" of the tree. At this point we do an initial classification of all of the unclassified files. To do

this, we find the most discriminating feature between the already classified files using Fischer's discriminant ratio (3).

$$FDR = \frac{(\mu_{i,1} - \mu_{i,2})^2}{\sigma_{i,1}^2 + \sigma_{i,2}^2} \tag{2}$$

Then, we find the label(s) that best fit the unclassified file along this feature using the minimum of the pairwise distance between the average of the files classified under that label and the file being examined. If two or more labels are within 20% of the best candidate, then we classify the file under all of these labels initially.

After the first pass, we rebuild the decision tree using the data we gathered from the initial classification. We reexamine all of the files with heterogeneous labels, and repeat the process. First, we find the most discriminant feature, then we find the label that best fits the file using this feature. In this pass, we only choose the best fit.

3 Interface Specifications

3.0.1 Task 1

This task is implemented on MATLAB and so requires MATLAB to run this program. When Task1a is executed over command line and prompts for required input parameters like L,K, Directory path where word files are located. It then generates an index structure which is reused by Task1b for query file to be indexed using the same hash family as we used in Task1a. Do not clear the session unless index structure of data files needs to be changed. When invoked Task1b it prompts for query file name, simulation files directory path, query file directory path and word file of query file directory path and t which are number of similar simulations needs to be visualized. Projection onto 1-D space is done using formulae.

$$hash = floor((dot([r\{row, col\}], x) + b(row, col))/w); \tag{3}$$

where r is a random vector of same dimension of our vector x with dimensions chosen from stable distribution.

3.0.2 Task 2

This task is implemented in MATLAB function calls and requires MATLAB command line to execute. The program takes as input the simulation file directory, query file directory, the query file names and the required paramters as specified by the user. The program outputs the size of the index structure in bytes.

The task 2 b accepts the query file name, simulation file directory, query file directory and the t values as input through the command prompt. The output of the program will be the t most similar files to the query file. It also outputs

the size of data accessed from the index structure and the number of vectors expanded.

3.0.3 Task 3

This task is implemented in MATLAB function calls and requires the MATLAB command line to execute. The function takes in parameter to be used in the program, the parameter are specified in Section 4.0.3. It is important to note that the output of Task3a is assumed to be used in the "graph" parameter in Task3b and Task3c.

3.0.4 Task 4

Both tasks are implemented as MATLAB functions using the command line. For the k-nearest neighbors classifier, the required inputs are the directory of the epidemic simulation files, the full path to the label file, and the number of neighbors to examine. The output will be text stating each label and its related files.

For the decision tree classifier, all that is required is the epidemic file directory and the full path to the label file. It will output the same as part A, a list of labels and their related files.

4 Requirements and Execution

4.0.1 Task 1

Task1 out puts the size of index structure in bytes. Given a query file and t it outputs the t similar simulation files and also the number of unique vectors considered, total number of unique vectors considered and also the number of bytes accesses from the index structure to find the t similar simulation files for the query file. These t similar simulation files are visualized as heat maps.

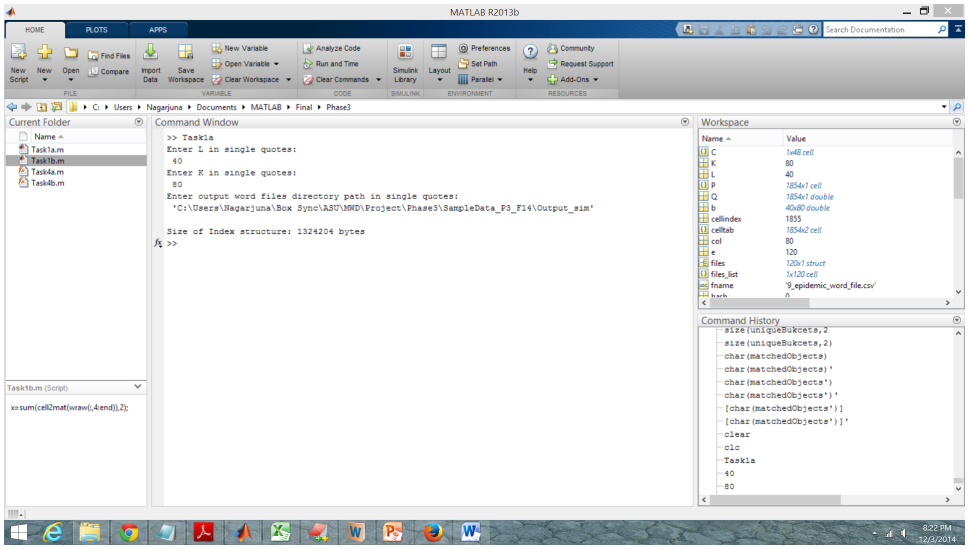


Figure 1: Task1 Size of Index structure

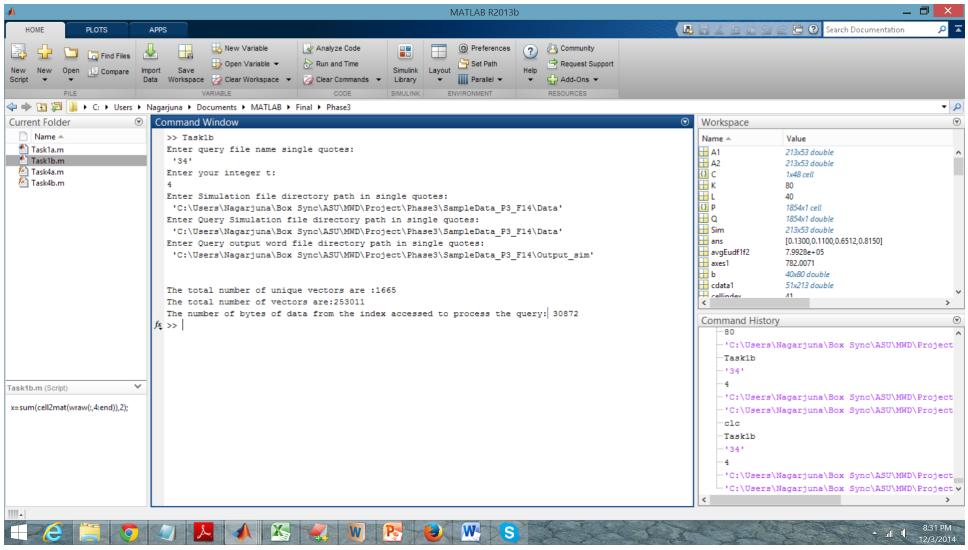


Figure 2: Task1 No. of Unique Vectors, Total No. of Vectors & No. of bytes access from Index

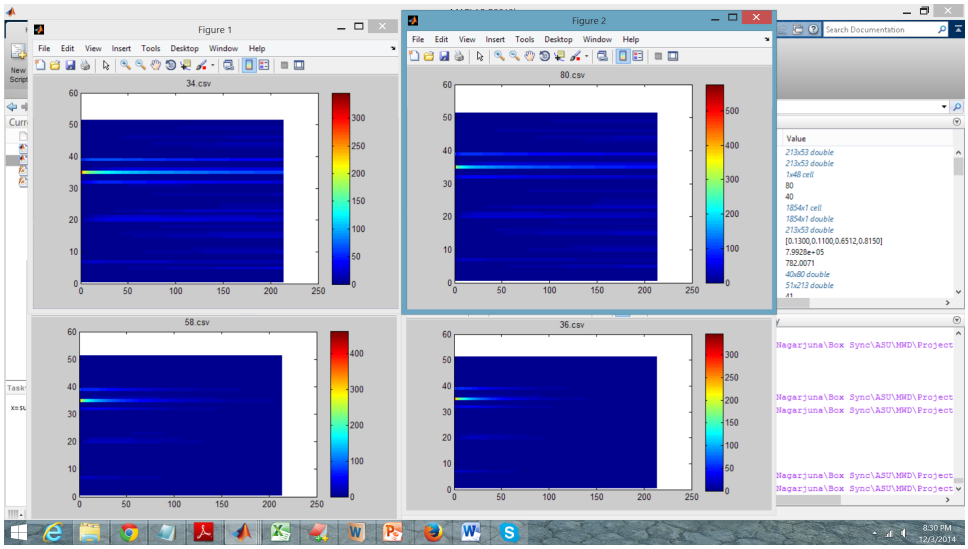


Figure 3: Task1 t similar simulations to query file

4.0.2 Task 2

Each task has been put into different program. The Task2a.m program creates the index structure. It taked the following input. 1) Output word files directory. 2) b - The number of bits per dimension.

The program outputs the following:

```
Enter output word files directory path in single quotes:
 'C:\Users\siddhu\Downloads\SampleData_P3_F14\SampleData_P3_F14\Output_sim'
Enter b:
 3

Size of Index structure: 622768 bytes
```

Figure 4: Task2a Size of Index structure

```
    '1.csv'
    '3.csv'
    '23.csv'
    '21.csv'
    '19.csv'
    '45.csv'
    '2.csv'
    '41.csv'
    '39.csv'
    '22.csv'


  Number of vectors accessed 244

  Number of bytes accesses from index 9798 bytes
>>
```

Figure 5: Task2a t most similarly

4.0.3 Task 3

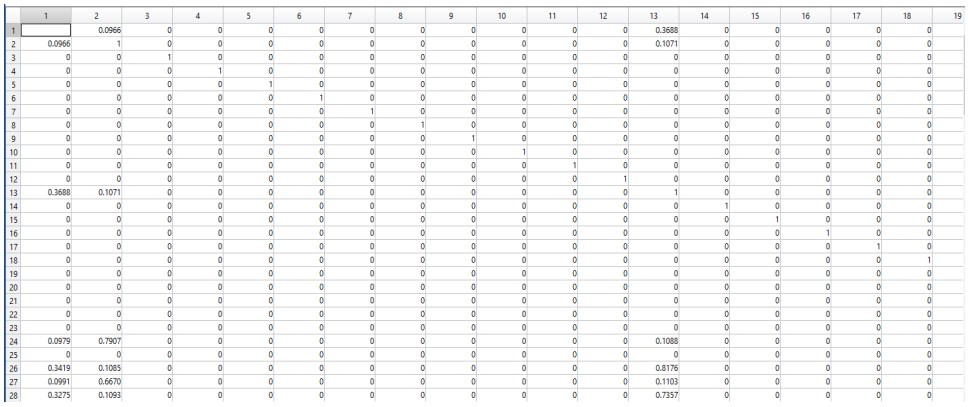
Each portion of this task has been put into its own MATLAB function. However, to execute part a through c, you must call function with parameters.

Run Task3a with the parameters:

- **dirName1** - The name of the directory in which the simulation files are stored.
- **tau** - An integer representing the threshold.

The output for Task3a are the following:

- **weightedAdjacencyMatrix** - A weighted simulation similarity graph that will be used in the latter 2 sub-tasks. The following figure shows a snapshot of the adjacency matrix.
- **fileSet** - A list of file names, act as a reference index to be used in the latter 2 sub-tasks.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1		0.0966	0	0	0	0	0	0	0	0	0	0	0.3688	0	0	0	0	0	
2	0.0966	1	0	0	0	0	0	0	0	0	0	0	0.1071	0	0	0	0	0	
3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	
7	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	
8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	
9	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	
10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	
11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	
12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	
13	0.3688	0.1071	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	
14	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
24	0.0979	0.7907	0	0	0	0	0	0	0	0	0	0	0.1088	0	0	0	0	0	
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
26	0.3419	0.1085	0	0	0	0	0	0	0	0	0	0	0.8176	0	0	0	0	0	
27	0.0991	0.6670	0	0	0	0	0	0	0	0	0	0	0.1103	0	0	0	0	0	
28	0.3275	0.1093	0	0	0	0	0	0	0	0	0	0	0.7357	0	0	0	0	0	

Figure 6: Task3a snapshot of the adjacency matrix with tau equal to 0.03

For Task3b the following parameters are required:

- **graph** - An adjacency matrix that represents a simulation similarity graph.
- **k** - An integer representing the top-k dominant files.
- **dirName1** - The name of the directory in which the simulation files are stored.
- **fileSet** - A list of file names, act as a reference index.

The output for Task3b prints out the solution and outputs the heatmaps. The following figure is a sample output.

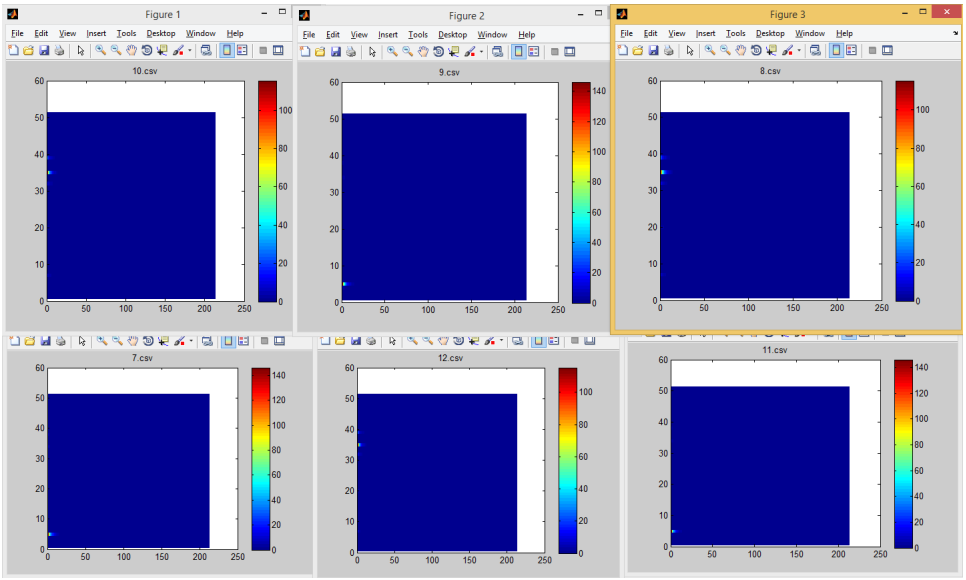


Figure 7: Task3b heatmap output with K equal to 6

For Task3c the following parameters are required:

- **graph** - An adjacency matrix that represents a simulation similarity graph.
- **file1** - First query simulation file
- **file2** - Second query simulation file
- **k** - An integer representing the top-k dominant files.
- **dirName1** - The name of the directory in which the simulation files are stored.
- **fileSet** - A list of file names, act as a reference index.

The output for Task3c prints out the solution and outputs the heatmaps. The following figure is a sample output.

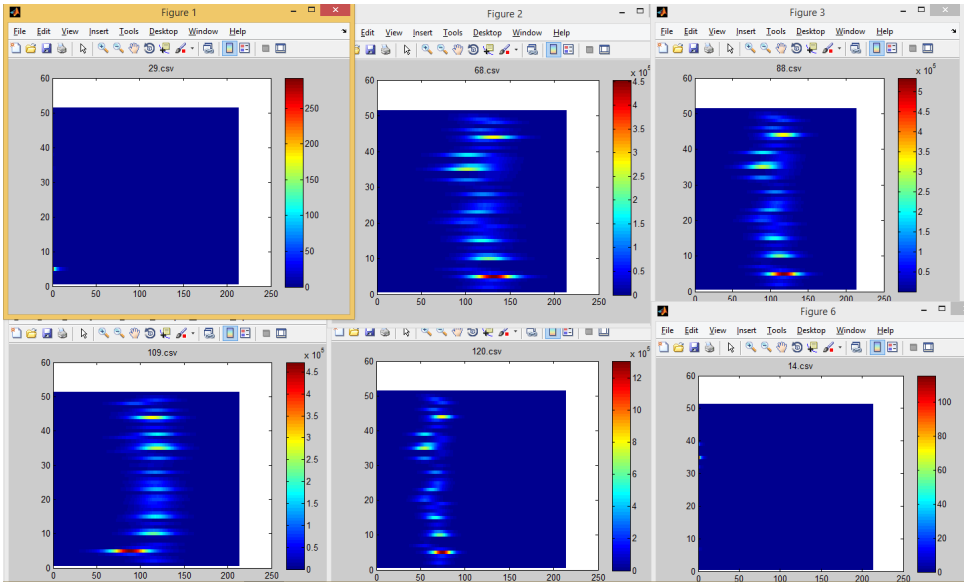


Figure 8: Task3c heatmap output with K equal to 6 and file 1 and file 6 as the query file

4.0.4 Task 4

Part A requires the following parameters:

- **dirName** - directory of epidemic simulation files.
- **labelFile** - full path to the label file.
- **k** - number of neighbors to examine for voting.

It outputs:

- **Label:** followed by the name of the label.
- **File List**
- This repeats for all labels.

Part B requires the following parameters:

- **dirName** - directory of epidemic simulation files.
- **labelFile** - full path to the label file.

It outputs:

- **Label:** followed by the name of the label.
- **File List**
- This repeats for all labels.

5 Related Work

This phase works on the deliver-ables of the previous phases 1&2. Word files generated from Phase1 and the similarity measures created in phase2 are reused.

6 Conclusion

In this phase we have learned and experimented on multi dimensional index structures, graph analysis, classification. We have run the test runs with mutiple different input parameters and observed the results.

References

[1] "Near-OptimalHashingAlgorithms for ApproximateNearest Neighbor in HighDimensions" (byAlexandr Andoni and Piotr Indyk). Communications of the ACM, vol. 51, no. 1, 2008, pp. 117-122.

[2] MultiProbe LSH: Efficient Indexing for High-Dimensional Similarity Search Lv Qin, William Josephson, Zhe Wang, Moses Charikar, Kai Li

[3] Candan, K. Selc, and Maria Luisa Sapino. "Classification." In Data Management for Multimedia Retrieval, 297-301. Cambridge: Cambridge University Press, 2010.

[4] Pan, Jia-Yu, Hyung-Jeong Yang, Christos Faloutsos, and Pinar Duygulu. "Automatic multimedia cross-modal correlation discovery." In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, 653-658. ACM, 2004.

[5] Brin, Sergey, and Lawrence Page. "The anatomy of a large-scale hypertextual Web search engine." Computer networks and ISDN systems 30, no. 1 (1998): 107-117.

[6] Stephen Blott, Roger Weber. "A Simple Vector-Approximation file for similarity search in high dimension Vector space." 1997. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.129.9708

Appendix

Member Roles

- Nagarjuna Myla - Task1
- Siddharth Sujir Mohan - Task2
- Jacob Mims - Task4
- Tsung-Yen Yu - Task3