Group: 15
Team Members:

1. **Abhimanyu Kulkarni**
2. **Nadim Mahesaniya**
3. **Shreyas Mahimkar**

Section: 01 (Tue 6pm -9pm)
Home Work: Project Final Report

INTRODUCTION

As per current statistical records, there are approximately 1.5 million violent crimes recorded in United States. Each city in United States has a trained and well-equipped police department. Currently local area police randomly go out on patrols to keep a check on illegal activities carried out. Our aim, through this project, is to reduce this randomness and prioritize the patrol timings by learning specific patterns of crimes so that the patrolling teams can be allotted to visit an area with a higher probability of crime. By expanding this project to a large scale (countries all over the world), we can actually prevent a real world problem which is on the rise – 'Crime'.

FBI and local police departments have taken a collective effort to maintain records of all crimes, reported or unreported, occurring all over US. This data is being used for statistical analysis, specifically to analyze the rise or fall of crime rate per year. We are leveraging the availability of this large data-set to create a mechanism to predict where probability of crime is high. We have taken care of the nitty-gritty details of predicting streets for a given zip-code, month and hour. What the police department will see is all the streets with particular weights assigned to them. These weights are significant in the sense that they give a more granular priority to the street when the police are making decisions of going out for patrolling.

The basic flow of the project is that we extract crime data available on web and clean the data as per our requirements. Missing information in the data is added using helper tasks and then given as input to the graph conversion program. This program creates a graph structure consisting of nodes and edges and stores it in the form of an adjacency list. The adjacency list works as an input to the Clustering Program which extracts interesting structures in the graph which will help us create tree-like structure. The clustered output is used to create the K-D tree which in-turn predicts the streets with higher probability of occurrence of crime for a given zip-code, month and hour.

One of the key observations made when experimenting with K-D Trees was that choosing the correct attributes as dimensions is a very important step. Prediction made by incorrect or even fewer dimensions are less accurate and hence cannot be set as the base for deciding patrolling locations. Another breakthrough during experiment phase was adding an extra dimension which improved the accuracy by significant amount. This report gives an in-depth details about experiments carried out and results obtained throughout the project life-cycle.

DATA

1. Description:

The dataset reflects reported incidents of crime that occurred in the some major Cities of United States from 2001 to present (2014). Data is extracted from the local Police Department's CLEAR (Citizen Law Enforcement Analysis and Reporting) system. In order to protect the privacy of crime victims, addresses are shown at the block level only and specific locations are not identified. This data can be downloaded in a CSV Format. This data set can be found in below locations:

https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2
https://data.cityofboston.gov/Public-Safety/Crime-Incident-Reports/7cdf-6fgx

2. Attributes:

There are approximately 8 million records each with around 20 attributes describing basic properties of crime like nature of crime, place of crime (latitude and longitude), time, etc. This data is updated daily, Tuesday through Sunday. This will increase exponentially to even a billion records after receiving more data from police department.

3. Sample Data:

Following is a fragment of dataset for the City of Chicago:

	ID	Case Number	Date	Primary Type	Block	Latitude	Longitude
1	9842104	HX491598	11/01/2014 11:56:00 PM	NARCOTICS	015XX W MARQUETTE RD	41.772314188	-87.663299511
2	9842094	HX491586	11/01/2014 11:51:00 PM	NARCOTICS	063XX N WESTERN AVE	41.996949863	-87.68970916
3	9842299	HX491803	11/01/2014 11:45:00 PM	MOTOR VEHICLE THEFT	050XX N ELSTON AVE	41.972916776	-87.74774186
4	9842547	HX492178	11/01/2014 11:45:00 PM	THEFT	029XX N SHEFFIELD AVE	41.934767715	-87.653784625
5	9842167	HX491650	11/01/2014 11:45:00 PM	CRIMINAL DAMAGE	054XX S NEW ENGLAND AVE	41.792908729	-87.79389102
6	9842126	HX491605	11/01/2014 11:45:00 PM	BATTERY	080XX S CRANDON AVE	41.749396455	-87.568777964
7	9842115	HX491565	11/01/2014 11:40:00 PM	PROSTITUTION	047XX W ARTHINGTON ST	41.869435091	-87.74461926
8	9842073	HX491566	11/01/2014 11:40:00 PM	NARCOTICS	003XX S LOTUS AVE	41.874912731	-87.761477633
9	9842105	HX491578	11/01/2014 11:38:00 PM	BATTERY	007XX W CORNELIA AVE	41.945336535	-87.649305513
10	9842157	HX491583	11/01/2014 11:35:00 PM	CRIMINAL TRESPASS	078XX S COTTAGE GROVE A	41.751569677	-87.60506616

4. Zip-Code Challenge:

a. What was the challenge?
   i. Currently the crime data-set has the latitude and longitude values to identify the crime-location. If we depend on latitude and longitude to give uniqueness to Crime-location node in our graph structure then the number of nodes will increase exponentially requiring heavy computational and IO costs.
   ii. Hence, we decided to combine set of latitude and longitude values. The only logical way was to assign zip-code to each location since it will be a collection of latitude and longitude and also provide uniqueness.
   iii. The challenge was that we only have 2500 free requests per day for getting the Zip-code using the reverse geocode API of google. If we buy the google geocode API it allows us to get 100,000 requests per day at the cost of $10,000 minimum package. This seems to be infeasible.

b. How We Solved This?
    i. We wrote a batch program to run on multiple machines simultaneously. Whenever the program reaches 2500 requests, it terminates until we follow the same process again the next day. Owing to this logic, we were able to populate around half a million records with the correct zip-codes for given set of latitudes and longitudes. The program is attached as part of this report – updatezip.java.

TECHNICAL DETAILS

A. Use Of Relational Databases (MySQL):

a. Purpose Of Task:
    i. The large crime data set obtained initially had some irrelevant columns. These columns were removed using basic DML operations in MySQL and code in Java to parse the files.
    ii. Updating the zip-codes we obtained from batch files was easier using basic SQL updates.
    iii. Also, we needed to cross-check our accuracy calculation program by checking some statistics from the data. By running aggregate queries on data made our manual cross-checking easier for comparing the predicted data and actual data.
b. Algorithm/Pseudo-code
    i. Read the two files with Chicago and Boston data.
    ii. Extract Block, date, type of crime, city, State, zipcode (this is calculated later using latitude and longitude), and latitude and longitude.
    iii. Save the data as shown above for converting it into a graph structure.
c. Queries Frequently Used And Why:
    i. The output of our program is a zip-code and all the streets lying in that zip-code where the probability of crime is high. The Query used to extract distinct streets for given zip-code and city was:

    SELECT distinct(block) from crime where city = <given_city> and zipcode = <given_zip>;

    ii. Correctness of our prediction algorithm is calculated as follows: Suppose we have crime records up to November 2014. We run our prediction framework and train the model for all years and months up to October 2014. We needed to extract the data for November from Database manually to compare it with our predicted output. The query used for this was:

    SELECT block,date,type,city,state,(SUBSTRING(zipcode,1,5)),lat,longi,clat,clong FROM crime where city = <any_city> and zipcode is not null date like '%11/%/2014%'

    iii. We needed to extract distinct zip-codes for a city for predicting crimes on that zip (input to K-Nearest Neighbor calculation). Query used:

    SELECT distinct(zipcode) from crime where city = <given_city>

    iv. Extracting records for a specified city,zip,date combination ordering by date descending

*SELECT * from crime where city = <given_city> and zipcode = <given_city> date like "11/%23:%" order by date desc*

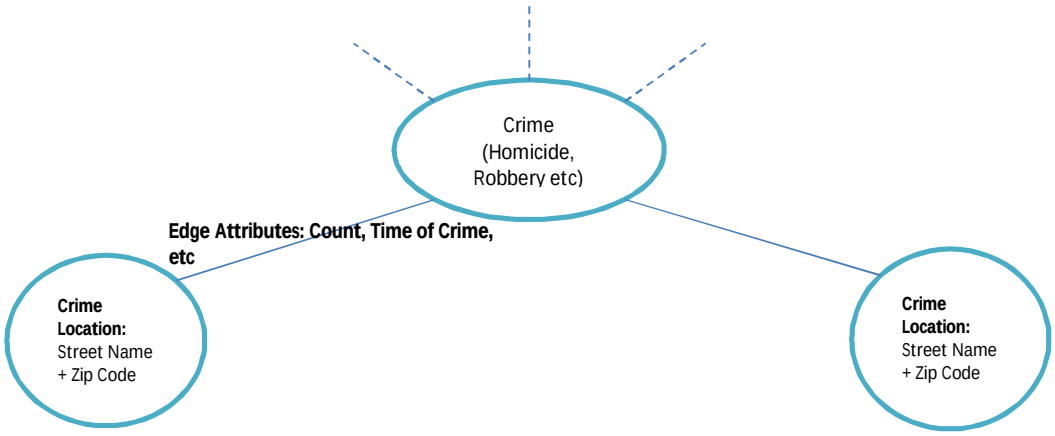
d. File Name: UploaddatatoRDB.java

B. Convert Crime Data To Graph (Map-Reduce):

a. Purpose Of Task:
   i. One of the first task was to write a Map-reduce program to convert flat data structure into graph data structure.

b. Approach To Solve Task:
   i. The first column stores the location of the crime and the subsequent columns are all the crimes that occurred in that location. This corresponds to the structure of adjacency list representation of graph where there are edges from each location node to all crime nodes that occurred in that location.
   ii. High Level overview of the Graph is as follows:
      1. Nodes: There are two types of nodes in this graph –
         a. Crime Location: The content of this node will be a combination of 'Street Name' and 'Zip Code' of the crime. These values will always be unique.
         b. Nature of Crime: The content of this node will be the type of crime.
      2. Edges: Edges of the graph describe the relationship between the nodes mentioned above. Each edge holds multiple attributes such as list of times at which the crime occurred in a given location, the number of times the crime occurred at that crime location etc. The delimiters used to differentiate between two crime-types or between 2 or more multiple date-time values was pre-decided.

      3. Rough Representation Of Graph:



c. Algorithm/Pseudo-code:

```
Class ConvertToGraph{
    Mapper{
        Input: Offset of split as key and CrimeData record as value
        Ouput: Key = (Street, City, State, Zip) combination
                    Value = (Crime Type, Crime Location) combination
        emit((Street,City,State,Zip), (CrimeType, CrimeLocation));
    }
```

```
    Partitioner{
        // Extract Zip from Key using CSV Parser
        getPartion(key, numberOfReduceTasks){
            zipcode = extractZip(key);
            return (zipcode.hash)%numberOfReduceTasks;
    }

    Reducer{
        Input: key emitted from Map and list of all crime types and locations
combined.
        Output: Key = Street, Zip combination
                Value = All CrimeTypes, countOfCrime, List of crime Times
        Setup(){
            // Inititialize: Nested HashMap to store street name and all its crime types.
         }

         Reduce(){
             // create hashMap Structure.
            //  The structure has a nested HashMap to Store CrimeType as key and all the list of
date-times as values.
         }

         Cleanup(){
             for each street in outer hashmap:
                 for each crimetype in inner hashmap:
                     emit((Street,Zip), (CrimeType,count,ListOfTimes));
         }
    }
}

Class CrimeLocationNode{
        Properties: Zip-code, Street, City, State

        CrimeLocationNode(){
            // Constructor to initialize all properties
        }

        CreateTextFromNode(){
            // Return text representation of Crime Location Node
        }
}

Class CrimeNode{
        Properties: Crime Type, Count, ListOfDateTimes

        CrimeNode(){
```

```
            // Constructor to initialize all properties
        }

        CreateTextFromNode(){
            // Return text representation of Crime Location Node
        }
        addCount(){
            // Function to add count
        }
        addTime(){
            // Function to add time of crime to list of crime built up-till now.
        }
}
```

d. File Name/s: ConvertToGraph.java , CrimeLocationNode.java, CrimeNode.java

e. Key Observations And Implementations:

i. Firstly, we observed that keeping latitude and longitude will unnecessarily increase the size of the graph as the number of unique CrimeLocationNodes will increase exponentially. At this juncture we decided to extract zip-codes and improve the I/O performance of the graph and use less storage on AWS.

ii. While creating the graph, we had an option of keeping CrimeNode as the source and all the CrimeLocationNodes as the outgoing nodes from it. Eventually we realized that this will create a very wide graph, that is, for a single crime-type there will be many streets from different countries and hence a very dense graph.

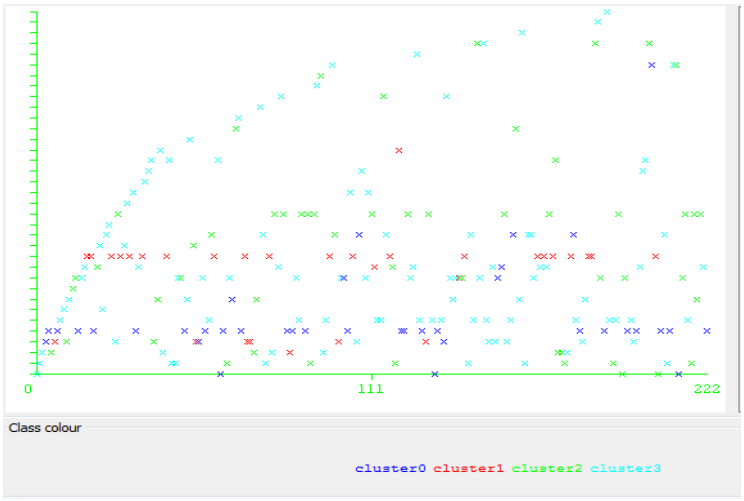
iii. The best option was to keep CrimeLocationNode as source and all the crimes occurring at that location as outgoing nodes in adjacency list. This will create a tall graph – which is not a problem for Map-Reduce since it anyways splits the graph horizontally and parallelize computing.

iv. Running Times: *(Refer SysLogs Folder)*

Number of Machines	Running Time(Secs)	Size Of M/C
5	125	Medium
10	88	Medium

C. Interesting Structures of Graph – Custom Clustering Using Weighting Schemes

a. Purpose Of Task:



i. For initial clustering we tried to cluster the data using k-means clustering for the following combinations:
   1. Zip-Code only
   2. Zip-Code vs Crime
   3. Zip-Code vs Date
   4. Zip-Code vs Street

ii. As you can see in the above data we were not able to find and interesting clusters in the data and also we unable to make any sense from the data. We got similar mixed results from the other comparisons.

iii. Later we figured that normally K-means clustering is used for Unsupervised Learning problems. But we actually wanted to implement a supervised learning approach.

iv. Seeing at the data we came up with our own custom clustering approach for finding interesting structures in the graph.

b. Approach To Solve Task:

i. We needed to cluster our graph output in such a way that it will work as a model to train our data against. Since the final output of our project was to predict all streets where the probability of occurrence of crime is high, we decided to implement a custom clustering algorithm which clusters based on some attributes. In our case these attributes were Zip-code, hour and month of occurrence of crime.

ii. One more additional implementation we added was to give the streets certain weights. Assigning weights was a challenging task since we had to take multiple factors like time of crime, month of crime, total count etc. into consideration. After many attempts the perfect weighting scheme we got was to give total count the weight of 1, recent count (did crime

occur in the last one year) as weight 50, month of occurrence of crime as weight 20 and hourly count as weight of 75.

iii. Since all the data we need is in the single record input to Map, we can calculate weights and then emit it to the Reducer. Therefore, the function of Map is to get all streets for a zip, hour and month combination and emit it along with weights assigned to each street.

iv. A grouping comparator was written to send all streets with the same combination of zip, hour and month to the same reducer. In this way, the only function of the reducer is to emit all the streets by removing duplicates and sorting by weights.

c. Algorithm/Pseudo-code:

```
Class ClusterCrimeData{

    class Mapper{
        Input: Input record from graph
        Output: Key = (zip-code,hour,month)
                    Value = Street1:weight1@Street2:weight2@....

        map(){
            // Use hashmap Structure to create a structure for each zip,hour,month combination.
            /*
             * weight assignments:
             *          1) total count of crime on a given street -> 1
             *          2) recent count of crime on a given street -> 50
             *          3) month-wise count of crime -> 20
             *          3) hour-wise count of crime -> 75
             */

             emit((zip,hour,month), (street1:weight1@street2:weight2...)));
        }
    }

    class Partitioner{
        // Extract Zip from Key using CSV Parser
    getPartion(key, numberOfReduceTasks){
    zipcode = extractZip(key);
    return (zipcode.hash)%numberOfReduceTasks;
    }

    class GroupingComparator{
        // send all streets with the same combination of zip,hour,month to the same reduce call
        compare(key1, key2){
            ziphourmonth1 = extractZipHourMonth(key1)
            ziphourmonth2 = extractZipHourMonth(key2)

            return ziphourmonth1.compareTo(ziphourmonth2);
        }
    }
```

```
        class Reducer{
                // simple reducer to emit calculated weights for streets for all combination of zip,hour and month
                reduce(ziphourmonth, [street1:123@street2:34@street3:50 ...]]){
                        emit(ziphourmonth, [street1:123@street2:34@street3:50 ...]]);
                }
        }
}
```

d. File Name/s: ClusteringCrimeData.java

e. Key Observations And Implementations:

Weights Assigned	Data Used For Experiment	Special Changes Made	Accuracy Achieved (Approx)
Total Count = 1 Count Per Month = 5 Recent Count = 12	Boston Crime Data (2011 - 2014)	-----	32.5%
Total Count = 1 Count Per Month = 5 Recent Count = 12		Ignore months where the crime count is 0.	16.18%
Total Count = 1 Count Per Month = 50 Recent Count = 25		Ignore months where the crime count is 0.	20%
Total Count = 1 Count Per Month = 50 Recent Count = 25		a. Ignore months where the crime count is 0. b. Zip Code that are not present in the actual data but present in the predicted data were not counted when calculating accuracy	30.31%
Total Count = 1 Count Per Month = 20 Recent Count = 50		a. Ignore months where the crime count is 0. b. Zip Code that are not present in the actual data but present in the predicted data were not counted when calculating accuracy. c. Months in which no crime occurred were only given total weight and not the recent weight	39.53%
Total Count = 1 Count Per Month = 20 Recent Count = 50 Hourly Weight = 75		~~Same As Above~~ Added: Hour at which crime occurred given weight	42.58%

i. Running Times For Clustering: *(Refer SysLogs folder)*

Number of Machines	Running Time(Secs)	Size Of M/C
5	121	Medium
10	65	Medium

D. K-Nearest Neighbors Calculation Using K-D Tree:

a. Purpose Of Task:
   i. Given a file to storing all streets grouped by the combination of zip-code, hour and date of crime, predict crime on the streets.

b. Approach To Solve Task:
   i. Read 2 input files:
      1. File-1: Output of the clustering file – This is a training file for making the k-d tree.
         Key: zip-code, hour, date
         Value: [Street1:weight1,Street2:weight2, …Streetn:weightn]
      2. File-2:
         a. The Street where crime is to be predicted.
         b. This file has the format: **"input", zip-code, hour, date**: For a given zip-code for a specific hour and for a given date we have to predict the streets where the crime would happen.
   ii. Mapper: For creation of the k-d tree in the reducer mapper emits 4 dimensions (zip-code, hour, month, Street).
   iii. Custom Partitioner: The 2 input files are partitioned considering only zip-code. This ensures that the data of the "to be predicted zip-code" goes to the same reduce task where the training of that zip-code is done for creation of the k-d tree.
   iv. Reducer: The Clustering output zip-code data as well as the "to be predicted zip-code" comes in to a single reduce task in this we achieve parallelization based on zip-codes.

c. Pseudo-Code:

```
Class kdtree_knn{
        Mapper{
                Input:
                Map key: Offset of split as key.
                Map Values : input file 1 -> Output of the Custom Clustering
                input file 2 -> Combination of zip-code,hour,date to predict crime on the streets for this combination
                Output:

                Input file 1{
                        Map key: zip-code,hour,date
                        Map value: StreetName:Weight
                }

                Input file 2{
                        Map key : "input",zip-code
                        Map output : zip-code,hour,date
                }
                emit(key,value) //for both input files
        }
        ZipPartitioner{
                Input file 1 and file 2{
                        Partition on zip, so that the input goes to the partition where the KDTREE is
                        created for a given zip. Both land on the same reduce task for prediction.
                }
        }

        Reducer{
                For Each Reduce Task{
                        Due to partitioning on Zip, the data of input file 1 and file 2 will land onto a single reducer.
                        Or creating a 4D Tree create training data (until October 2014) with the 4 dimensions as given
    below.
                                1st dimension -> zip-code
                                2nd dimension -> hour
                                3rd dimension -> month
                                4th dimension -> StreetName
                        Provide the input combination of (zip-code,hour,month) for calculation of the 3 nearest neighbors.
                        Sort the StreetNames based on the weights of the output of the predicted streets.
                        key = zip-code,hour,date
                        value = [StreetName-1:weight-1,StreetName-2:weight-2, ... , StreetName-n:weight-n] // Predicted
    Streets
                        emit(key,value)
                }
        }
  }
```

d. File Name/s: kdtree_knn.java

e. Key Observations And Implementations:
(Accuracy Files placed in Folder: / Accuracy-Outputs/ PredictedOutputs)
(Syslog Files for RunTimes in Path /Syslogs/<Descriptive Folder Name>)

Model	Description	No. of Machines	Running Time	Accuracy
3D – 1NN	3 dimensional 1 nearest neighbor	2	783secs	42.58% (Ref:
4D – 1NN	4 dimensional 1 nearest neighbor	5	95secs	63.81% (Ref: 4D 1 nn accuracy.txt)
		10	83secs	
4D – 3NN	4 dimensional 3 nearest neighbors	5	89secs	63.93% (Ref: 4D 3nn accuracy.txt)
		10	94secs	

i. 3D – 1NN (Attempt 1)
    1. Input dimensions – 3: zip-code, hour, month.
    2. Output – Top 10 Streets sorted on weight descending.
    3. This model gave us an accuracy of 42.58% after adding hour to the calculation of weighting schemes.
    4. As the prediction accuracy was less, we did not bother it running on larger set of machines.

ii. 4D – 1NN (Attempt 2)
    1. Input dimensions – 4: zip-code, hour, month, Street.
    2. Output – Streets sorted on weight descending.
    3. This model gave us an accuracy of 63.81% after adding hour to the calculation of weighting schemes.
    4. Adding one dimension for prediction for an input combination of (zip-code, hour, month) we get more granular streets compared to the 3D - 1NN, and also ordering of the weights given to the number of streets changed significantly. Hence the output accuracy a lot better than that of the 3D - 1NN.

iii. 4D – 3NN (Final Attempt)

    For adding just one more dimension increased the accuracy very significantly. So we went a little further checking by taking 3 Nearest Neighbors. But the weights given to the number of streets do not change significantly. Hence the output accuracy a little better than that of the 4D - 1NN. But this will always give more results than that of the previous model.

iv. If we individually train the model (4D – 3NN) on Chicago then we get an accuracy of 85% (Refer File: Chicago_4D_3nn.txt in /Accuracy-Outputs/ PredictedOutputs/) as the data of Chicago was from 2001 onwards till 2014, more as compared to Boston.

E. Accuracy Calculation:

a. Purpose Of Task:

i. This program finds accuracy based on the output of the K-D tree. The K-D tree predicts the streets where crime will occur for each combination of zip-code, month and hour. We compare this predicted streets against the future data of actual streets where crime occurred and calculate the percentage accuracy.

b. Approach To Solve Task:

i. We iterate over the actual data where crime occurred and store list of streets for each zip-code, month and hour combination.

ii. Once we have lists of actual streets where crime occurred, we take predicted streets and iterate over actual streets.

iii. If predicted street exists in actual streets than we increment count. Based on all matching predicted street we calculate accuracy.

iv. We keep on adding accuracy for all combination of zip-code, month and hour to get final accuracy. This is the accuracy of our crime prediction model.

c. Algorithm/Pseudo-Code:

```
Read Actual Crime occurrence file:
Map<key,Set> actualStreetMap
For each line in file do:
        Key = zipcode + hr + month
        Get streets where crime happened for given Key and store in Set

Read Predicted crime file:
For each line in file do:
        Get all predicted streets of crime
                For each predicted street do:
                        Check if predicted street exists in Actual actualStreetMap
                                Increment matching street count
                accuracy = accuracy + accuracy for each record
```

d. File Name/s: CalculateAccuracy.java

CONCLUSION

After carrying out series of experiments, the final combined accuracy for both Boston and Chicago crime data obtained is approximately 62%. The failed attempts and inaccurate algorithms formed the stepping stones for this percentage. One of the major road-blocks of the project was getting the zip-code. Another time-consuming task was to ask for more crime-data from police department since the only open-source data available was for cities Boston and Chicago. Creating a database out of crime-data assisted a lot during the final phases like accuracy-calculation.

This percentage can be increased by substantial amount if data is trained against more robust prediction models. In our experiments we used KD-Tree with different number of dimensions and getting either 1 or 3 nearest neighbors. During the course of experimenting we realized the importance of adding dimensions to KD tree as well as the significance of weighting streets based on multiple attributes. One such attribute could be the type of crime that occurred. Although predicting the type of crime that could occur is less probable to be accurate, one alternative is that we could give a list of probable crime types that can occur at a given street.

Future scope of this project is that it can be integrated with real world applications and mobile applications. Few examples can be an application - “Crime-Map” where the predicted streets can be plotted on Maps and colored based on the weights assigned to them. Other application could be a simple alert message through Email or Text which sends out text alerts to people entering a zip-code with high probability of occurrence of crime at that very hour. Trend analytics is a growing field in the world of data-mining where the three types are descriptive, predictive and prescriptive. Descriptive analytics is analyzing data what we have and creating interesting results and charts from it. Predictive is similar to what we tried to do in our project, which is, based on past data learn a specific pattern and predict the future results. What this project in future needs is prescriptive analytics, where we need to basically prescribe an action based on predicted results. Prediction outputs multiple possibilities and it is in the hand of person doing Prescriptive analytics to decide what action needs to be performed for the most significant predicted output.