Anomalous Trajectory Detection to Determine Taxi Frauds

Siva Kiran Bhamidimarri,
Master's Student,
Department of Computer Science,
Texas Tech University.

1. Introduction:

GPS tracking has become one of the essential requirements for any taxi these days [9]. The GPS traces left by the taxis can be used to perform various types of analysis and information extraction [1]. Information about the roads, traffic, popular places and other metropolitan details can be extracted by using these traces [10]. One of the best examples to understand the importance of finding the anomalous trajectories of Taxi trips is to observe how the drivers in tourism based cities exploit the visitors by taking longer routs to earn more money. If we figure out a way to find if a path is anomalous, we can dig deeper into the details of this particular trip and then understand why this trip was anomalous. Such studies on trajectories have gained considerable focus in the recent years [4]. A trip being anomalous doesn't necessarily mean that the driver is a fraudulent driver. It could also mean that there might be unusual traffic in the regular route or there might be some construction or road repair going on. This information can also be concluded by investigating these unusual occurrences.

One of the most common ways to commute within a visiting place is to hire cabs [6]. When we are new to the place, we don't even know the directions to where we want to go. We trust the cab driver and just go with it. Some greedy cab drivers might exploit the fact that we do not know the directions and take us in routs which take longer to reach so that they can earn more. This has been a very common practice to exploit the tourists and visitors. There is a dire need to automate the process of detecting fraudulent trajectories taken by the greedy drivers. By observing these anomalous trajectories, we can also determine the city road's exceptional behaviors like construction works or other disturbances that occurred on the path.

2. Motivation:

GPS traces are a continuous stream of updates of a device's (in this case, the taxis) location at the current time. This stream of location data is key for this project. The data that we used also contains information about the occupied status along with the time stamp. The information about the occupied status can be used to extract the information about the source and destination. In order to follow the approach of Isolation Based Detection, we need data which has the set of all trajectories which has the same source and destination.

This problem is an interesting problem, as we can find a lot of interesting observations from the data. We can make use of the latest big data platforms to analyze and extract important information from the taxi location data. The method called Isolation-Based Anomalous Trajectory Detection [6] has been explored in this project. This approach focuses on isolating each test trajectory by comparing it with the set of trajectories with same source and destinations pair. To achieve this, we need data which can provide information about – taxi location and trip source and destination information.

3. Related Work:

3.1 Anomaly Detection:

Anomaly Detection is an important area of study because the anomalies in data can translate to significant actionable information in a wide variety of application domains. For example, an anomalous pattern in computer network traffic could mean that some of the computers are sending out information out to unauthorized destinations. This information could be sensitive and detecting such occurrences is necessary. Presence of anomalies in the MRI scan can indicate the presence of tumors. One of the most popular anomaly detection is the fraud detection by credit card companies [18]. Anomalies in credit card transaction data could mean that there is an identity theft.

Anomalies can be broadly categorized into three types [16]. If an individual data instance is observed to be considerable different from rest of data, then the instance is considered to be a point anomaly. An observation is a contextual anomaly if it is an anomaly is specific conditions, but it is categorized to be a normal observation otherwise. If a collection of related data instances from the whole data is anomalous with respect to the entire data set, it is categorized as collective anomaly. The problem of determining anomalous trajectories is a collective anomaly detection problem. We consider the set of grids that are available in the trajectories. Now, these grids become the observations in the problem. We compare these set of grids with the entire dataset.

3.2 Types of Anomaly Detection Techniques:

Supervised anomaly detection: The techniques that use supervised anomaly detection methods expect some sort of labels to be present for the observations in the data. These labels describe the observations – both normal and anomalous observations. Typical approach for a supervised anomaly detection is to build a predictive model for normal classes vs. anomaly classes. The new observations are sent as the input and compared with the remaining observations. It is expensive to get data which has labels to all the observations.

Semi-Supervised anomaly detection [17]: The techniques which use semi supervised detection methods assume the presence of labels too. But they require labels only to the normal observations. As these techniques do not require labels for anomalous entries or observations, they are more widely applicable than the supervised anomaly detection methods. Typically, we build a model for the class corresponding to normal behavior, and then we use this model to determine anomalies in the testing data.

Unsupervised anomaly detection: The techniques which use unsupervised anomaly detection methods do not require labels at all. These techniques make implicit assumptions on the data and categorize the instances as normal and anomalous. These assumptions are usually made based on how frequently the observations occur. The frequent ones are usually considered as normal observations.

To implement a taxi trajectory anomaly detection, we should use an unsupervised anomaly detection method. Based on the number of occurrences of the grids in the trajectories from the source and destination, we consider the observations to be either normal or anomalous. If the number of occurrences of a particular grid is very low, we categorize the entry to be anomalous. Since we have multiple observations for a single trajectory, we consider the collective anomalous nature of all the observations in a trajectory as opposed to all the observations in all the trajectories in the entire data with the dame source and destination.

3.3 Output of Anomaly Detection:

One of the important aspects of anomaly detection is the way they are represented to be anomalous. Typically, there are two ways of representing the anomalies [16]:

Scores: Scoring techniques compute an anomaly score to each of the instances in the testing data. This score is computed depending on the degree to which that instance is considered to be an anomaly. The output of such techniques is a list of entries with assigned scores. We may choose to consider a certain number of top entries with highest anomaly score as anomalies, or we may assign a threshold value to categories the observations as anomalies.

Labels: Labeling assigns labels to each of the instance in the data as either anomaly or normal.

In the project, we considered scoring and setting a threshold to be more appropriate. This is because we do not know how many anomalies exist for each source-destination pair. So, computing a score and then setting a threshold to categorize the observation to be anomalous is essential. The basic idea is to compute the score based on how frequently a certain grid occurs in the trajectories that have the same source and destination.

To find out anomalies, we need to define what is normal. The anomalous observations which lie close to the boundary with the normal observations can be falsely detected as normal and vice-versa. Here are some of the problems that we encounter while trying to determine anomalies in taxi trajectories.

—We are dealing with vehicle data in which the vehicles are traveling from a particular source to a particular destination. So, we need to define the normal behavior based on all the trajectory information we have.

—There may be multiple normal trajectories between the same source and destination. This is because different drivers may have different route preferences.

—When the city roads change over time, new routes are preferred by the drivers. So we will have to account for new normal routes that come into existence over time.

—We need to define what are anomalous trajectories. In other words, how will we say that a path is anomalous? How different must it be from a group of normal routes?

3.3 Unsupervised Anomaly Detection Techniques:

3.3.1 iBOAT:

Due to simultaneous development of the taxi industry and GPS technology accuracy, there has been quite some interest in the field of anomalous trajectory detection [5][7]. Isolation Base Online Anomalous Trajectory [15] detection focuses on determining anomalous paths from a set of paths with same source and destination. The idea of iBOAT is to maintain an adaptive working window. This working window consists of the latest incoming GPS points. These points are compared against the historical trajectories that are already fed into the system. As a new incoming point is added to this window of adaptive workflow, the set of historical trajectories is analyzed and then they remove any trajectories that are inconsistent with the subtrajectory in this window. New points are continuously added to this window as long as the support of the subtrajectory in the adaptive working window is above a certain value. If the support drops below this specified value, then the adaptive working window is reduced to contain only the latest GPS point.

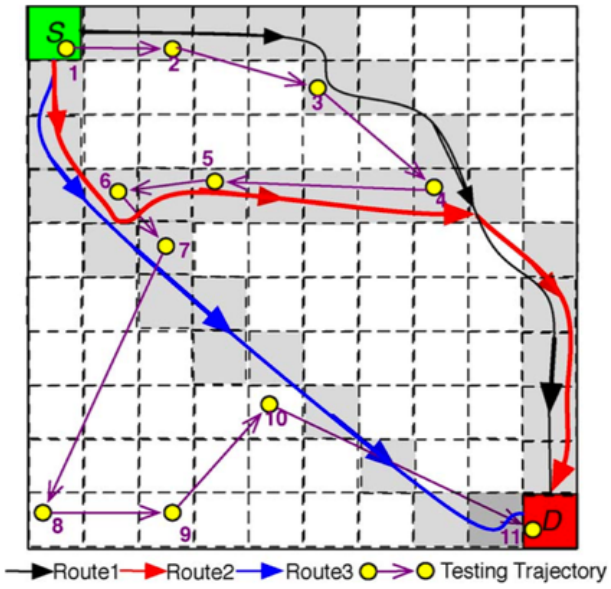

Image 1: Working of iBOAT

The image represents the working of iBOAT. The points in yellow belong to the test trajectory and the other routes belong to the trajectories in the historical data. The Source and Destination of these trajectories is the same. The points from 1 through 11 are continuously checked against the points that belong to the remaining trajectories. When points 1 to 7 are fed into the system, the score of the points is definitely under the threshold that is mentioned. But once the point 8 is fed to the system, the overall score of the test trajectory increases based on the iBOAT scoring system. If this value happens to be over the threshold value, then the trajectory is identified as an anomalous trajectory. If the value still remains under the threshold, the next point, point 9 is fed to the system. The same process is repeated until all the points in the trajectory are compared or until when the score becomes greater than the threshold.

3.3.2 F-Trail:

There is another approach called F-Trail [14] which handles the trajectory of a taxi locally, rather than the whole path. This method focuses more on the behavior of the taxi journeys. This approach tries to cluster the different taxi behaviors. They categorize the taxi's whole travel into occupied and unoccupied trails. They compute the fractal dimensions for the individual trajectory analysis. After computing this for individual trajectories, they extend this to find out the sociability of the taxi movement for two statuses – full and empty. They then compute the fractal dimension of each trail. This helps in determining how the taxi drivers find the passengers. They try to analyze the behavior and cluster the trails based on the fractal dimensions. They categorize the trails based on the scores in this way:

FD = 1: This happens when the trajectory is a line or a smooth curve
FD > 1: This happens when the taxi does twists and turns
F D < 1: This happens when the taxi does many stops
FD = 0: if the taxi is completely static.

But we are more interested in finding out how different a trajectory is, from the rest.

3.3.3 Origin Destination Matrix:

Another popular approach is the creation of OD Matrix [7]. Origin-Destination matrix represents the density of a selected grid area with the information of how many instances of data are located in that particular row and column. The OD matrices can be used to understand any anomalous patterns that appear in the flow of traffic.

$$P = VN_1 \rightarrow N_2 \rightarrow \cdots \rightarrow N_{k-1} \rightarrow VN_k,\ k = 1, 2, \ldots,$$

Here, P is a path, VN1 is origin virtual node and VNk is the destination virtual node
The OD matrix looks like this:

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1,w-1} & x_{1w} \\ x_{21} & x_{22} & \cdots & x_{2,w-1} & x_{2,w} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{T-1,1} & x_{T-1,2} & \cdots & x_{T-1,w-1} & x_{T-1,w} \\ x_{T,1} & x_{T,2} & \cdots & x_{T,w-1} & x_{T,w} \end{bmatrix}.$$

Image 2: OD Matrix Representation

The OD matrices are used to represent the densities of any given location, mapped to a grid in a time changing fashion. Time based OD matrices are updated [7] so that comparison of these time evolving OD matrices will help in detecting unusual traffic or unusual activities of a set of trajectories.

3.3.4 Isolation Tree Method:

One of the approaches to detect if the trajectory is anomalous or not is to use the isolation tree approach [5], which follows a depth based isolation process. It is called the isolation tree approach, which creates an inverted indexed tree of the trajectories and checks for the grids. There is a possibility of an anomalous trajectory to contain the common paths, but contains loops of the same path. The isolation tree doesn't take into account this aspect of the anomalous detection.

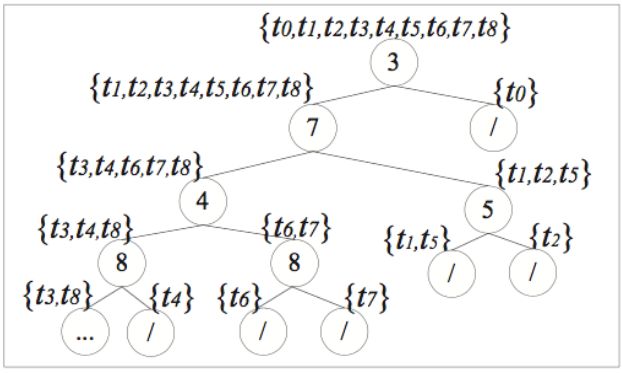

Image 3: Isolation Tree for Anomalous Trajectory Detection [6]

Image 3 shows that the set of tracks {t1 … to ti} contain the point n. The tree represents the inverted indexes. As we can see from the diagram, the point or cell number 3 is traversed in all the trips that are present in the set. Then, the tree contains child nodes which represent other points from the whole set of trajectories. From the diagram, we can see that the grid 7 has been accessed by all the trajectories/trips, except the trajectory t0. And, from the diagram, we can see that t0 does not pass through any other grid. Grid 3 is the only grid that has been accessed by the trip t0. The remaining grids undergo the same kind of binary division into the tree. When there are no more grids present in the trajectory, the tree stops going further deeper with a stop node.

But such an approach would miss the effect of loops that might occur in the trips. Once the trip is assigned to a grid and goes to the next level, it cannot go back to the upper level of node to account for the loop.

4. About the Data:

Initially, we explored data which contains information about the trip distances, pick up and drop offs and also the fare rates [11]. But this is not sufficient to understand the anomalous trajectories. To understand the trajectories taken by the taxi drivers from source to destination, it is essential to have a continuous stream of 'longitude and latitude' updates from the taxis. The perfect data would contain at least – location stream in quick successions, occupied status and the time at which the location was recorded. Here is the snippet of data that was used to analyze the trajectories.

```
37.75901 -122.3925 0 1213084172
37.77053 -122.39788 0 1213084092
37.77669 -122.39382 0 1213084032
37.78194 -122.38844 0 1213083971
37.78999 -122.38909 0 1213083910
37.79728 -122.39609 0 1213083855
37.79838 -122.40239 0 1213083811
37.79779 -122.40647 0 1213083736
37.79779 -122.40646 1 1213083715
37.79657 -122.40521 1 1213083655
37.79305 -122.40471 1 1213083600
37.78945 -122.40405 1 1213083535
37.78833 -122.40859 1 1213083475
```

Image 4: Data Snippet

The data is acquired from Crawdad [12], which is a known for its repositories of IOT based data. The data contains location updates of over 500 taxis collected over 30 days in the San Francisco Bay area. The dates on which the taxi data was collected and recorded is from 17th May 2008 to 10th June 2008. The first column is the latitude position; the second column is the longitude position. The third column represents the occupied status of the vehicle at that instance. The fourth column contains the time stamp at which the entry was recorded.

4.1 Selecting the region of Observation:

It is essential to select the proper region of observation. That is because, we are going to follow a grid based approach to divide all the traces of GPS plots, and if the region does not contain enough trails, it is just a waste of computational power. The total number of entries in the data set is 11,219,955.

Here are some findings of the region:
The latitude position farthest from the equator is 50.30546.
The latitude position closest to the equator is 32.8697.
The longitude position farthest from the prime meridian is -127.08143.
The longitude position closest to the prime meridian is -115.56218.
The length of the region: 817 km.
The height of the region: 1938 km.

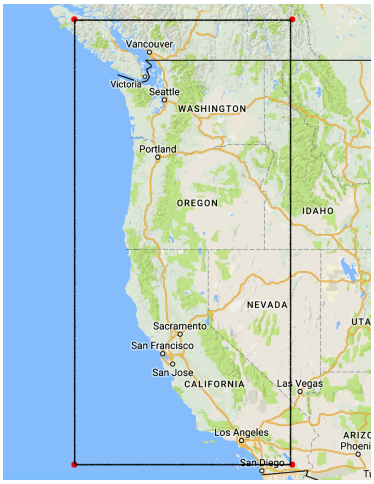

Image 5: Rectangular boundary of the extreme points from the whole data

Since the region covered by these extreme locations is very large, we have reduced the region to accommodate points which fall within the areas of San Francisco. Doing this drastically reduces the computation power required, by losing only a very small portion of data. After trying out various boundaries for the dataset, the optimal boundary with minimum area of observation and maximum containment of data are:
37.820000, -122.525000
37.700000, -122.350000
37.820000, -122.350000
37.700000, -122.525000
Here is the plotting of these points on the map:

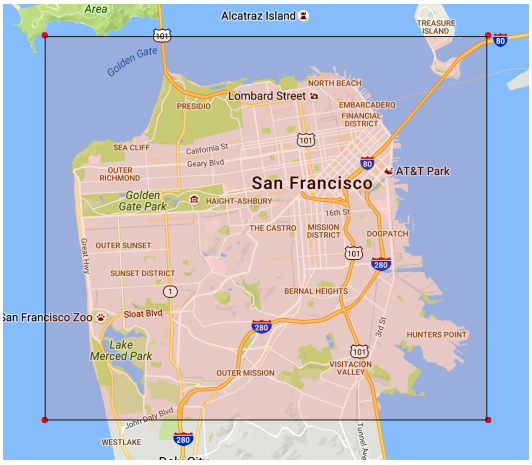

Image 6: Selected Region and Boundaries

This new boundary eliminates around 1.2 million entries but the density of points is maximum in this area. Reducing the region by few meters from here on starts removing thousands of data entries. To understand more about the region in observation, here are some details:
The length of the region is 15.371548650370013 kilo meters. [Approximately 15000 meters]
The height of the region is 13.343391197346925 kilo meters. [Approximately 13000 meters]
The area covered by this region is 203.5 km^2.

After eliminating the points that fall outside of this boundary, we have 9,977,340 entries. This is not bad, as the region under analysis has been reduced by several hundreds of times and the analysis can now be performed with much attention to detail, and also helps in reducing the computation power that is required. Also, after dividing the map into grids, the grids will not contain as many blank cells because of higher density.

5. Tools and Implementation:

5.1 Tools:

Since we are dealing with a lot of data entries, we need powerful tools which can work well with huge datasets. The text files containing the GPS traces is 380 MB with more than 10 million entries. So, we used Spark as the platform which suits well for big data computations. We used Spark 1.6 version, as it supports various other components of the big data infrastructure, keeping in mind the possible future updates of the project. Since Spark inherently supports Scala, we used the Spark interactive shell to run the Scala scripts. These scripts were essential to perform curtail information retrievals and also to transform and refine the data into the required form. The Spark version we used comes with Scala 2.10.5. We also used Java to perform some core computations on the data. The main algorithm was also implemented in Java. To understand the trajectories that are traversed visually, we used the GPS visualizer tool, which works with GPS files containing latitude and longitude entries.

5.2 Spark:

Spark is a fast and general engine for large-scale data processing. When users have large and massive datasets like web logs, biometric data, click streams and other real time data, Spark can slice and dice the data by distributing the processing to huge cluster of computers. It can handle data analysis problems for huge datasets and distribute it to multiple machines.

Spark is Scalable. It runs on top of cluster manager. Spark scripts are everyday scripts written on Java, Scala or Python. Under the hood, when we run the program, Spark handles all the process of distributing and managing the process. Spark runs on multiple clusters. If we have a Hadoop cluster, Spark can run on top of YARN cluster manager. Spark will split up and create multiple executors per machine. Ideally, we will require 1 per CPU core. It does all the coordination using cluster manager and driver program itself to farm out work and distribute it to different nodes and also provide fault tolerance - when one of the executor stops working, it manages in such a way that the program still runs without any interruption and without needing to start the program all over again. The flow of control is represented in the graph below:

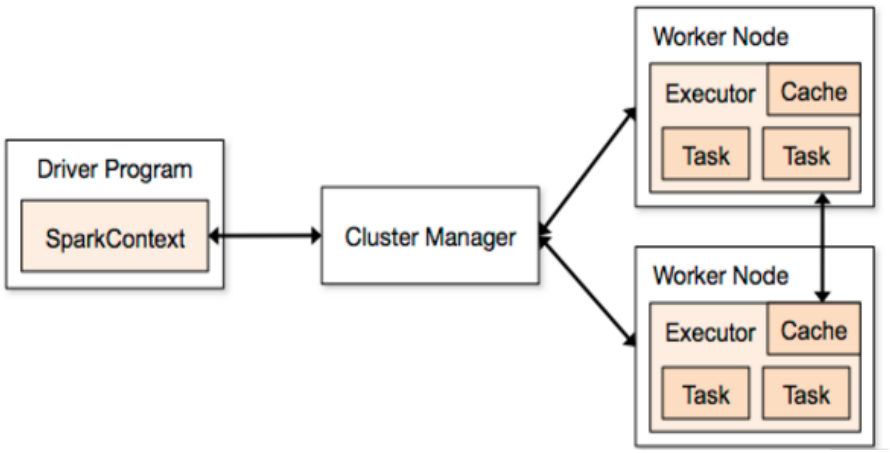

Image 7: Internal working of Spark [19]

One of the main reasons for using Spark is that it is faster than Hadoop MapReduce. On the Apache Spark website, they say it is 100 times faster in memory and 10 times faster on disk. Although practically it is around 2 times faster for regular computations. It achieves such speeds by making use of DAG Engine. Directed Acyclic Graph optimizes the workflows. Spark doesn't perform any action until it is actually requested to perform a task or to deliver a result. And, as soon a request is made. it creates a graph describing the steps to complete the task. And, this path is optimal. Some of the popular companies that use Spark are Amazon, Ebay, NASA, Groupon, TripAdviser, Yahoo.

5.2.1 Components of Spark:

The components of Spark include Spark Core and various libraries on top of it. The components are represented in the diagram below:
Spark Streaming Library: This library gives ability to analyze real time data stream, like web logs coming from web servers, which are continuously updated.
Spark SQL: It lets us run spark on HIVE context. It works on structured data. It lets us run SQL queries on top of it.
MLLib: It is a series of machine learning algorithms. Any computations which require machine learning components can be performed using MLLib, as it has a lot of common operations like Pearson Correlation. This is the library which can be utilized to build a recommendation system.
GraphX: This library deals with network theory. It can be used to work on networks like social graphs and documentation citations.

5.2.2 Resilient Distributed Datasets:

One of the main objects that we used that is made available by Spark is the RDD (Resilient Distributed Dataset). It is an abstraction for a giant set of data. They can be split across many computers for computation. They are also fault tolerant. Spark and cluster manager manage the specifics on how it works. Nothing actually happens until an action is called. As soon as an action is asked for, an optimal path is computed by Spark.

We formed grids and categorized all the 10 million refined entries into these grids. We used Java 2D arrays to achieve this. So, an array with size 50x50 was created. Each of the cell represents the physical location on the map. Now, comparing each of these 10 million entries with these 2,500 grids is a really huge task for Java, even with the latest computing power. We used a MacBook pro with 2.2 GHz Quad core Intel i7 processor with Java v1.8. It took 13 minutes to complete the computations. Next, we used Apache Spark's RDDs to perform the same set of computations. That is, categorizing 10 million entries into 2,500 grids. It took 11 seconds for Apache Spark Platform (with Scala 2.10.5) on the same machine. Initially, the map was divided into 100x100 grids. The computation time when it was computed using java was 54 minutes. And when Apache Spark handled the same task, it took around 49 seconds. The performance comparison can be seen in the graph below:

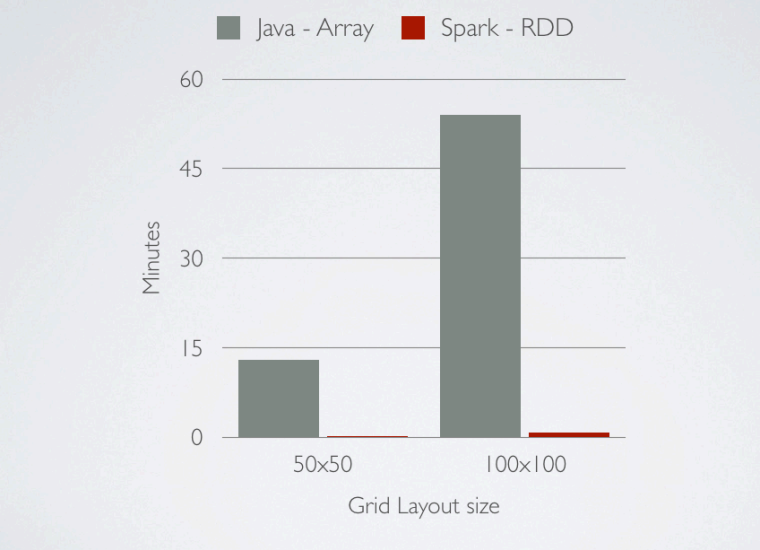

Image 8: Java Array vs. Spark RDDs – Computation Speed

5.3 Forming Grids:

It is essential to transform the data points into grids [6]. This is because, we will later consider aspects like trips which have same source and destination pairs. If we consider the exact longitude and latitude positions, we would get only a few trips that satisfy such conditions, as the pick-ups and drop offs even to the same source location and destination location will vary. The region of observation is divided into grids, which we will consider for the forming regions of trajectories to determine the path followed by a taxi. To understand, let us consider all the pick-up and drop off locations of one of the taxis and plot them on the map. The points in green are the pick-up points and the points in red are drop off points. After forming the grids, we will allocate all the points that are present in the grid with that cell number. And later in the experiments, we will be using this id to reference all the points that are present in that particular cell. If we see the picture below, we can clearly understand that in spite of the pick-up or drop off being associated to the same location, the plotting of the points is different. But when you group them to a grid, they will represent the same 'area' (rather than point) to which they belong.

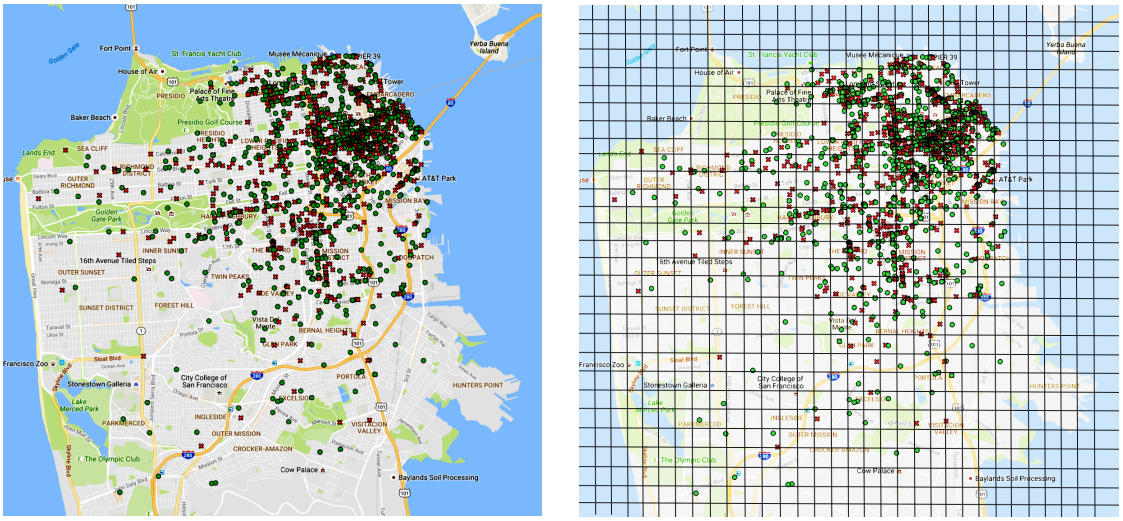

Image 9: Pick up and Drop offs of Taxi_1 and Grid Formation

The region of observation is divided into 50x50 grids. So, each grid length is approximately 300 meters and the height is approximately 260 meters. So all the points in the mappings fall in one of these 2,500 grids. So, the 10 million points which we have in the observation region will be in one of 2.5 thousand path blocks. To achieve this, we created an object called Coord in Java and insert the values of latitude and longitude values into it. Then, compare each of the latitude and longitude values with the grid boundary values and assign that Coord object the value of the grid.

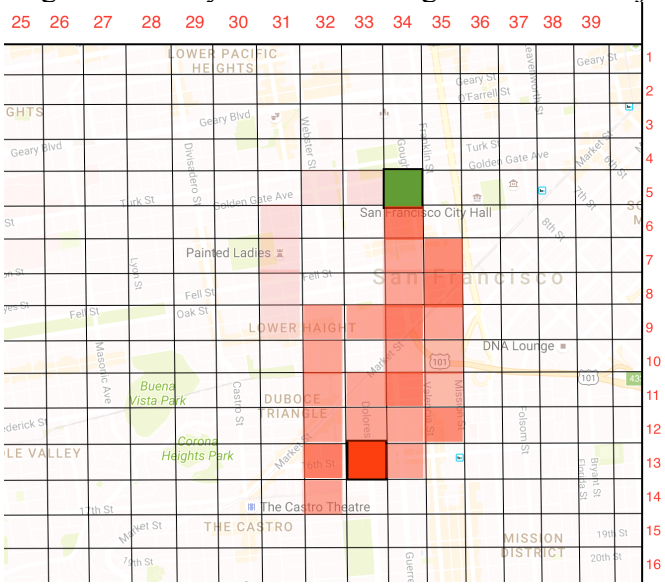

Image 10: Cells traversed from the Source to Destination.

Image 8 is the density representation of all the points that appear in the trajectories that contain the source and destination that are plotted. The green box is the source and the red box is destination. By making use of Spark's various Actions and Transformations [13], we can accumulate/ categorize the whole data as follows for each Source and Destination Pair. We acquired about 107 thousand SD pairs for the data.

```
(p 12 29|p 5 29,t.12.902|t.30.328)
(p 11 33|p 13 35,t.16.582|t.37.940|t.44.839|t.48.373|t.7.207)
(p 19 24|p 12 32,t.35.533)
(p 10 37|p 10 37,t.12.287|t.20.340|t.22.52|t.37.856|t.8.299)
(p 20 29|p 12 33,t.23.1073)
(p 13 31|p 16 7,t.8.638)
(p 24 25|p 22 30,t.0.774|t.40.323|t.46.407|t.48.682)
(p 22 16|p 28 30,t.26.91)
```

Image 11: Refined and Restructured Data with SD pairs and T Set

This data indicates that - for a particular source and destination cell (S,D) the trips made by taxis are a set of T= {t.x.y}. t.x.y means that it is the taxi with taxi id x and it is the yth taxi trip by that taxi. Such a naming convention makes it easier to go to the actual trail of the taxi and look at the data where actual entries were made. As an example, consider the following entry from the data:

(p 24 25|p 22 30,t.0.774|t.40.323|t.46.407|t.48.682)

This indicates that the (source,destination) (p 24 25, p 22 30) has been taken by taxi with id 0 (his 774th ride), taxi with id 40 (his 323rd ride), taxi with id 46 (his 407th ride), taxi with id 48 (his 682nd ride).

6. Isolation Based Anomalous Trajectory Detection:

Now that the data is converted into required form, we acquire the set of the trajectories that with the same (S,D) pair. As an example, here is a set of trajectories with the same (S,D) pair.

```
a = {"p 5 34","p 5 34","p 6 34","p 8 35","p 9 33","p 9 32","p 9 32","p 9 32","p 10 32","p 12 32","p 13 32","p 14 32","p 13 33"};
b = {"p 5 34","p 5 34","p 6 34","p 9 35","p 9 34","p 10 34","p 12 34","p 12 34","p 13 33","p 13 33"};
c = {"p 5 34","p 7 34","p 9 34","p 10 34","p 12 33","p 12 32","p 13 32","p 13 33"};
d = {"p 5 34","p 6 34","p 7 34","p 9 35","p 10 35","p 11 35","p 12 34","p 12 34","p 12 32","p 13 32","p 13 33","p 13 33"};
e = {"p 5 34","p 5 34","p 6 34","p 8 34","p 9 35","p 11 35","p 11 35","p 12 34","p 12 34","p 12 33","p 12 32","p 13 32","p 13 32","p 13 33"};
f = {"p 5 34","p 5 34","p 6 34","p 7 35","p 8 36","p 9 34","p 10 34","p 12 34","p 13 34","p 13 33","p 13 33","p 13 33"};
g = {"p 5 34","p 5 34","p 5 33","p 5 32","p 6 31","p 7 31","p 8 31","p 9 31","p 11 32","p 12 32","p 13 32","p 13 33","p 13 33"};
h = {"p 5 34","p 5 34","p 6 35","p 8 36","p 8 36","p 8 35","p 9 34","p 9 32","p 10 32","p 10 32","p 11 33","p 12 33","p 13 33","p 13 33"};
i = {"p 5 34","p 6 34","p 7 34","p 9 34","p 9 34","p 11 34","p 11 34","p 11 33","p 11 32","p 12 32","p 13 32","p 13 33"};
j = {"p 5 34","p 5 34","p 7 34","p 9 34","p 9 34","p 11 34","p 11 33","p 12 33","p 12 32","p 13 32","p 13 33"};
k = {"p 5 34","p 5 34","p 6 34","p 9 35","p 10 35","p 12 35","p 12 34","p 12 34","p 12 34","p 13 33"};
l = {"p 5 34","p 5 34","p 6 34","p 6 34","p 9 35","p 11 34","p 11 32","p 12 32","p 13 33","p 13 33","p 13 33"};
o = {"p 5 34","p 6 34","p 9 34","p 10 34","p 11 34","p 13 34","p 13 33"};
p = {"p 5 34","p 6 34","p 6 35","p 8 36","p 10 37","p 11 37","p 11 36","p 12 36","p 13 35","p 14 34","p 14 33","p 13 33"};
```

Image 12: Trajectory Set Representation

Now, T is the set which contains all these trajectories.

T = {a,b,c…..o,p}

So, the points/cells that are present in this test trajectory are taken into consideration. The Algorithm focuses on testing one trajectory against the remaining trajectories. This means that, we provide an input trajectory with the set of paths that include this trajectory, and then the iBAT algorithm detects whether it is an anomalous trajectory or not. The factors affecting the score are

the set of trajectories that is taken, the number of points that are different from the set{points} which are present in the input trajectory, the sampling factor and the vector size.

iBAT focuses on the test trajectory. It separates the test trajectory from the rest trajectories. It does this by selecting cells randomly, from the input test trajectory. We have T to be the set of trajectories that are available in the data. We select a random cell from this test trajectory 't' and then remove all the trajectories that do not have this particular cell in them. This process is repeated until all the trajectories are removed or until we come to a point where all the points of these trajectories is the same.

From the set of trajectories that we see above, we can clearly see that the trajectory 'g' contains cells that are not often found in other trajectories. This indicates that the anomaly vector of the trajectory, when it is considered as the test trajectory will have higher values. This causes the anomaly score to be higher than others. In the experiments, we have taken the size of the sample to be '8' trajectories and the size of the anomalous vector to be 20. With these values, the trajectories which have anomaly score greater than 0.8 are likely anomalous.

```
Input: t = Set of path cells traversed
       (test trajectory)
Set the sample size ψ and vector size m

Load the trajectories with same SD Pair into set T
let n be a vector of m zeros (ni's are zeros)
for i = 1 to m do
    T' ← randomly sample ψ trajectories from T
    repeat
        ni ← ni + 1
        randomly choose a cell p from t
        T' ← select the trajectories that include p from T'
    until T' is empty
    or until T' has all the points as in t
end for
s(t, ψ) = s(t,N) = 2^(-E(n(t))/C(N))
```

iBAT Algorithm [6]

The output is the anomaly score. S(t,N) represents the anomaly score of trajectory t for the set of N trajectories. N represents the sample number. E(n(t)) is average number of cells used for isolating t, N is the number of trajectories from which we separate t, and c(N) is the average of n(t) given N (it is the average path length of the unsuccessful searches in a binary search tree)

$$c(N) = 2H(N-1) - 2(N-1)/N$$

Here, H(i) is the harmonic number, and the value is ln(i) + 0.57721566 (Euler's constant). As the value of E(n(t)) approaches 0, the value of s(t, N) approaches 1. From the experiments on various test trajectories and Trajectory sets, we came to a conclusion that, to a given set of trajectories and corresponding sample size and anomaly vector size, the test trajectory t can be categorized as not anomalous if the score is <0.8 and it is anomalous if the score is >0.8.

Since the algorithm performs random selection from the set of trajectories and also the set of grids that are present in the test trajectory, we calculate the scores multiple times and find the average of the anomaly scores.

7. Results:

We collected all the trajectories that have the same source and destination pair. The SD pairs which had considerably large number of trips associated were filtered. We selected an SD pair (p_13_33, p_4_32) out of these. The number of trajectories that have this same source and destination pair is 324. We applied the same algorithm to the whole set of trajectories. We sent out each trajectory as the input test trajectory as explained earlier. The program was run 100 times and here are the results:

Trajectory SD: (p_13_33, p_4_32)
Number of Tracks: 324
Average Number of Anomalous Tracks: 23.53
Average of Anomaly Scores: 0.8345

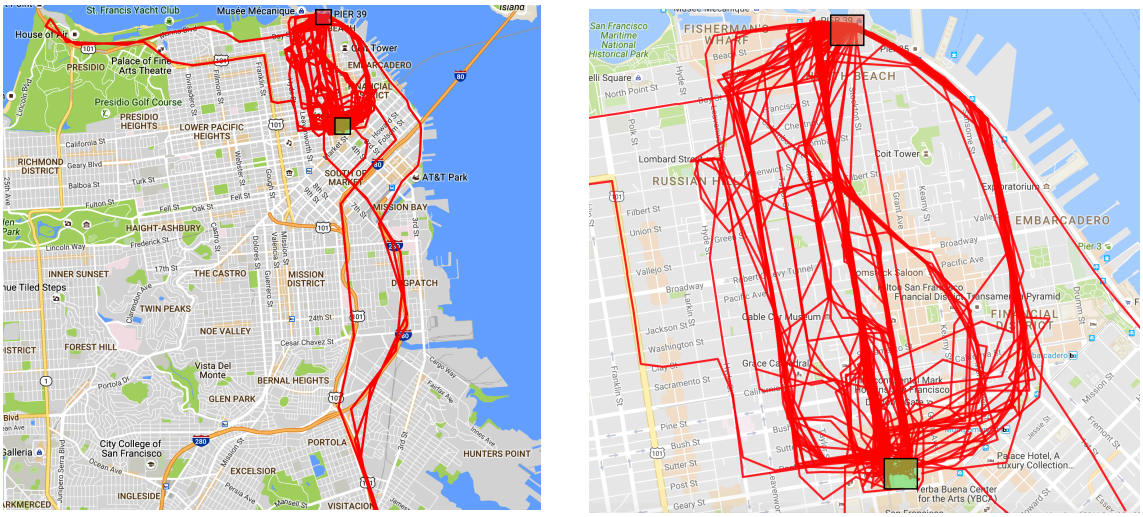

Image 13: Plots of Trajectories for SD (p_13_33, p_4_32)

The Green Box represents the source grid and the Red Box represents the destination grid. The sizes of these grids are 300m x 260m. As we can clearly see from the plotting, there are quite a few anomalous trajectories, and we shouldn't just worry about the intention of the driver. These trajectories are anomalous, and are clearly out of the S to D path.

We also selected another SD pair and performed similar analysis on this.

Trajectory SD: (p_11_32, p_14_33)
Number of Tracks: 234
Average Number of Anomalous Tracks: 6.10
Average of Anomaly Scores: 0.82

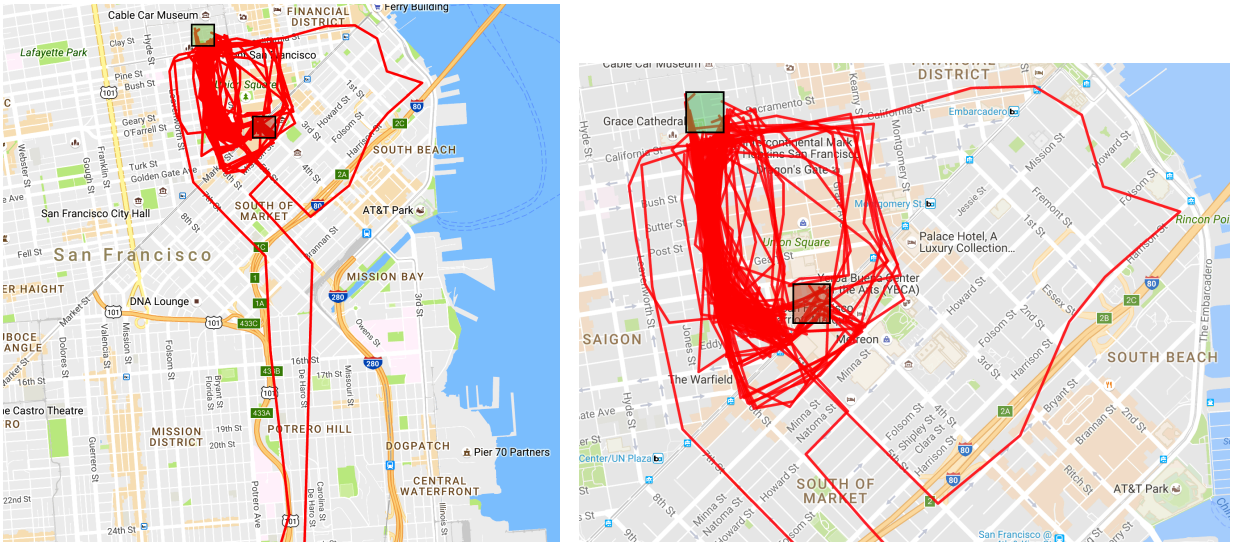

Image 14: Plots of Trajectories for SD (p_11_32, p_14_33)

The number of anomalous trajectories is considerably less for this SD pair when compared to the earlier pair. By observing the trajectories visually, we can see that there are around 5 tracks which are anomalous. These 5 tracks are always identified as anomalous by the program. Some tracks which follow slightly anomalous paths are detected as anomalous paths sometimes. We compared how many times an anomalous trajectory has not been detected for both of the trajectory sets. We also compared how many times a normal path has been identified as an anomalous path.

Trajectory Set	Number of Trails	Avg. of Anomalies	Percent of Anomalies
T1	324	23.53	7.3%
T2	234	6.10	2.6%

We can observe that the number of anomalous trails are considerably low in case of T2. Could it mean that passengers that travel to this SD pair are prone to be hit by fraud drivers? Is it that the drivers in this region are more fraudulent? Could it also mean that this SD pair attracts new visitors and the taxi drivers are attracted to exploit these new visitors? We can make conclusions such as ‘high activities like repairs and constructions in the path of SD1’.

We fed a perfectly normal route and a clearly anomalous trajectory to both of these trail sets. We compared how often the normal routs are shown as anomalous and how often the anomalous paths are not detected. The findings are plotted on the graphs below:

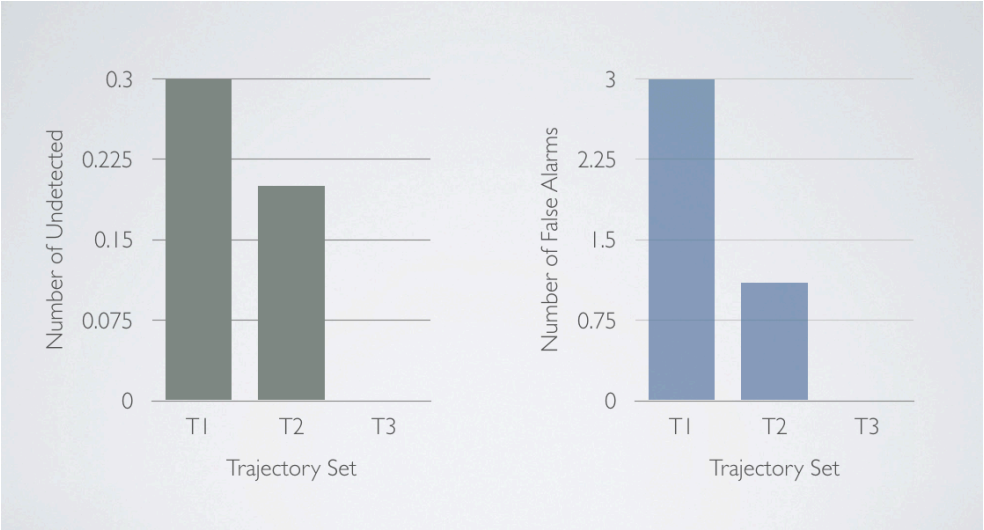

Image 15: Average number of Undetected Plots and Fasle Alarms

From the observations, we can see that with limited information, i.e., if the number of sets is in the trajectory list is less, the chances of false alarms is less. So is the undetected anomalous paths. As the number trajectories in the set increases, we encounter more false alarms and also few anomalous paths go undetected. But, as we can observe, the number of undetected anomalies is very low. Most of the times, the anomalous paths go detected, but sometimes the normal paths can be seen in the output of the anomalous paths.

The graph below is the plotting of performance of the algorithm for the two trajectory sets. We fed a selected number of trajectories from the trajectory set in the order of 100, 200, 300 and 400. For the sets contacting less than 400 trajectories, we just duplicated the existing entries and computed the time taken by the algorithm. Performance of the computations is observed to be linear. It took 11.3 seconds to calculate the anomaly scores of all the trajectories T1 (when the number of trails in T1 is adjusted to 400), which has an average of 14.47 grids / points in each trajectory. It took 7.9 seconds to calculate the anomaly scores of all the trajectories T2 (when the number of trails in T2 is adjusted to 400), which has an average of 6.61 grids / points in each trajectory.

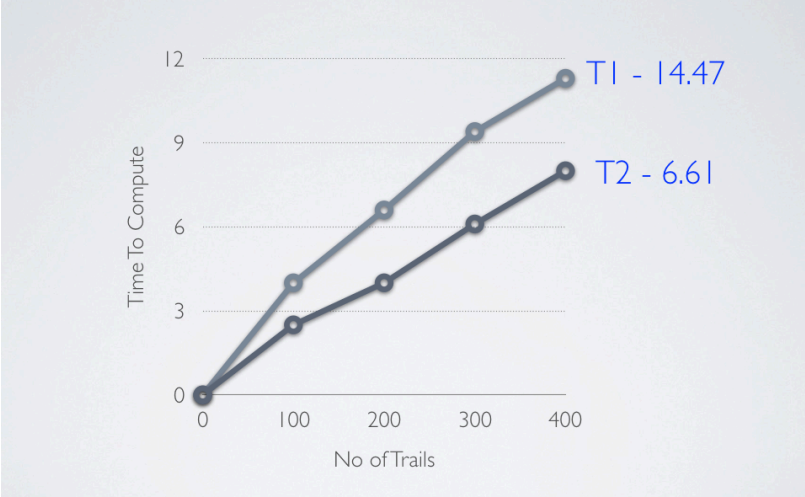

Image 16: Performance of the Algorithm

7.1 Another Test SD pair is as follows:

The source and destination of this new set of trajectories are p 5 34 (row 5, column 34) and p 13 33 (row 13, column 33).
We fed an input trajectory which has the values

{"p 5 34","p 7 34","p 9 34","p 10 34","p 12 33","p 12 32","p 13 32","p 13 33"}

After sending this as the input trajectory to the program and running it, the anomaly vector is {8,8,8,8,8,8,11,8,8,8,8,8,8,8,8,8,8,8,8,8}. The score that is obtained for this input trajectory is 0.5. The anomaly score below 0.8, so we can safely consider that it is not an anomaly. As we can see, the anomaly vector contains values which are not so high. This indicates its similarity with most of the other trajectories.
We fed another input trajectory which has the values

{"p 5 34","p 6 34","p 7 34","p 9 34","p 9 34","p 11 34","p 11 34","p 11 33","p 11 32","p 12 32","p 13 32","p 13 33"}

After sending this as the input trajectory to the program and running it, the anomaly vector and the anomaly vector is {8,9,8,9,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8}. The score that is obtained for this input trajectory is 0.3535533905932738. The anomaly score below 0.8, so we can safely consider that it is not an anomaly. As we can see, the anomaly vector contains values which are not so high. This indicates its similarity with most of the other trajectories.
This is yet another input trajectory, but the results of this trajectory are different.

{"p 5 34","p 6 34","p 6 35","p 8 36","p 10 37","p 11 37","p 11 36","p 12 36","p 13 35","p 14 34","p 14 33","p 13 33"}

After sending this as the input trajectory to the program and running it, the anomaly vector is {8,24,8,30,8,19,22,8,8,21,8,30,8,8,16,8,8,17,8,17}. The score obtained is 0.9833701391699664. The anomaly score above 0.8, so this is an anomalous trail. Also, we can see that the anomaly vector has values which are high. It means that the path is deviating considerably from the rest of the paths. This is the reason for high anomaly score.
The plots of the above set are as follows:

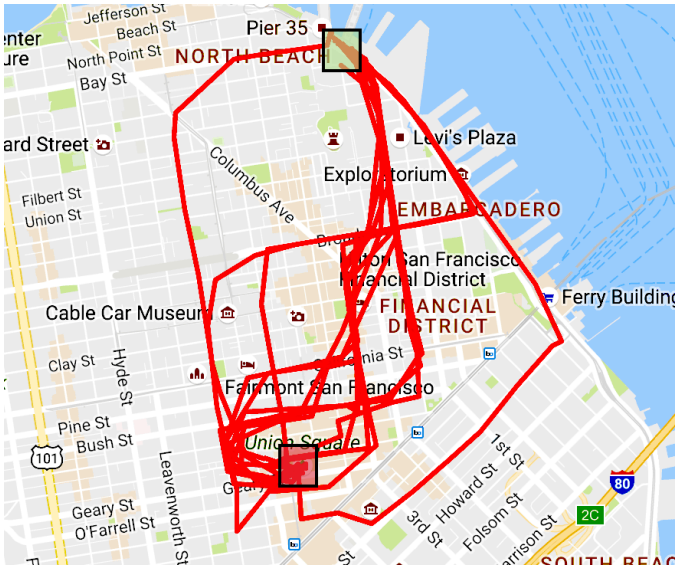

Image 17: Plots of Trajectories for SD (p_5_34, p_13_33)

From the plots, we can clearly see that there are two paths which are anomalous. This manual checking clearly says that we have 2 anomalous trajectories. We got the same result most of the times. The program also results another anomalous trajectory. It is the trajectory which is passing has common cells with the other two anomalous trajectories. Even manually checking the trajectory, we do not know if the path is anomalous or not. Such situations need more data. If there are more paths that contain cells from this trajectory, then we can say that the path is not anomalous. This means, if we feed the system with tracks containing similar paths, we wouldn't get this path as an anomalous trajectory.

8. Conclusion:

The isolation method focuses on comparing each of the test trajectories with the rest of the set to find anomalous paths. Repetitive random comparison of each cell of the trajectories with a random subset makes the fault rate really low. The performance of the algorithm is done by performing comparison with manual analysis of each of the trajectory if it is anomalous or not. As we have seen in the results, the undetected rate is really low. So, if a path is clearly anomalous to a set of humans, it will likely be detected by the algorithm.

These anomalous paths have trip ids associated with them, as mentioned earlier. This can be used to find out the exact trace along with the time stamp of each of the record. With this information, we can see if there is any time based dependency for the anomalous behavior of the taxis [7]. This problem can also be extended towards real time stream analysis. By constantly updating the trajectories of current path, if we perform the iBat analysis on the whole data with the current path and its current grid position as the test trajectory and test point. The real time fraud alert can be sent out in such a system [8].

9. References:

1. P. X. Zhao, K. Qin, Q. Zhou, C. K. Liu, Y. X. Chen. Detecting Hotspots From Taxi Trajectory Data Using Spatial Cluster Analysis, 2015 International Workshop on Spatiotemporal Computing, 2015.
2. J. Krumm and E. Horvitz. Predestination: Inferring destinations from partial trajectories. In Proc. Ubicomp 2006, pages 243–260, 2006.
3. D.J.Patterson, L.Liao, D.Fox ,and H.A.Kautz. Inferring high-level behavior from low-level sensors. In Proc. Ubicomp 2003
4. Y.Ge,H.Xiong, Z.-H.Zhou, H.Ozdemir, J.Yu, and K.C. Lee. Top-Eye: Top-k evolving trajectory outlier detection. In Proc. CIKM 2010, pages 1733–1736, 2010.
5. F.T.Liu,K.M.Ting, and Z.-H.Zhou. Isolation forest. InProc. ICDM 2008, 2008.
6. D. Zhang, N. Li, Z. H. Zhou, C. Chen, L. Sun, Shijian L. iBAT: Detecting Anomalous Taxi Trajectories from GPS Traces, 2011.
7. L. Moreira-Matias, J. Gamab, M. Ferreiraf, J. Mendes-Moreirab, L. Damasg. Time-evolving O-D matrix estimation using high-speed GPS data streams, 2015.
8. C. Chen, D. Zhang, P. Samuel Castro, N. Li, L. Sun, and S. Li. Real-time Detection of Anomalous Taxi Trajectories from GPS Traces, 2011.

9. N. Chadil, A. Russameesawang and P. Keeratiwintakorn, Real-time tracking management system using GPS, GPRS and Google earth, ECTI-CON 2008. 5th International Conference, 2008.
10. C. Zhang, Z. Zheng, F. Zhang and J. Ren, Multidimensional traffic GPS data quality analysis using data cube model, Transportation, Mechanical, and Electrical Engineering (TMEE), 2011 International Conference, 2011.
11. C. Whong, NYC Taxi Trip and Fare Data, downloaded from http://www.andresmh.com/nyctaxitrips/, 2014.
12. M. Piorkowski, N. Sarafijanovic‐Djukic, M. Grossglauser, CRAWDAD dataset epfl/mobility (v. 2009‐02‐24), downloaded from http://crawdad.org/epfl/mobility/20090224, https://doi.org/10.15783/C7J010, Feb 2009.
13. G. Satish, A. Rohan. Comparing Apache Spark and Map Reduce with Performance Analysis using K-Means, International Journal of Computer Applications, 2015.
14. Y. Matsubara, L. Li, E. Papalexakis, D. Lo, Y. Sakurai, and C. Faloutsos. F-Trail: Finding Patterns in Taxi Trajectories, Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2013.
15. C. Chen, D. Zhang, P. Samuel Castro, N. Li, Lin Sun, S. Li, and Z. Wang, iBOAT: Isolation-Based Online Anomalous Trajectory Detection, IEEE Transactions on Intelligent Transportation Systems (Volume: 14, Issue: 2, June 2013), 2013.
16. V. Chandola, A. Benerjee and V. Kumar, Anomaly Detection: A Survey, ACM Computing Surveys (Volume: 41, Issue:3, Article 15), 2009.
17. D. Dasgupta. and F. Nini,. A comparison of negative and positive selection algorithms in novel pattern detection, IEEE International Conference on Systems, Man, and Cybernetics, 2000.
18. S. Ghosh and D. Reilly, Credit card fraud detection with a neural-network, 27th Annual Hawaii International Conference on System Science (Vol. 3), 1994.
19. K. Frank, Taming Big Data with Apache Spark and Python, https://www.udemy.com/taming-big-data-with-apache-spark-hands-on/learn/v4/overview, 2014.