Predicting Destination of Taxi Rides

A PROJECT REPORT

submitted by

ARPIT AUGUSTINE	**B120877CS**
GADI VENKATA SAI RAHUL	**B120715CS**
HARSH PARSURAM PURIA	**B120775CS**
VINEEL PATNANA	**B120811CS**

in partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology
in
Computer Science and Engineering

under the guidance of

DR. K A ABDUL NAZEER
MR. IBRAHIM ABDUL MAJEED

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY CALICUT
NIT CAMPUS P.O, CALICUT
KERALA, INDIA 673601

MAY 2016

ACKNOWLEDGEMENTS

We would like to thank our advisors Dr. K A Abdul Nazeer, CSED, NITC and Mr. Ibrahim Abdul Majeed, CSED, NITC for their continuous support and understanding spirit during the entire period of this project. Besides our advisors, we would like to thank Mr. T A Sumesh, Software Systems Lab in-charge, CSED, NITC for providing us with necessary workstation to work on. Also the guidance we received from the entire panel and all of our friends is gracefully acknowledged.

ARPIT AUGUSTINE
GADI VENKATA SAI RAHUL
HARSH PARSURAM PURIA
VINEEL PATNANA

DECLARATION

"We hereby declare that this submission is our own work and that, to the best of our knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgement has been made in the text".

Place: NIT Calicut
Date: 11/05/2016

Signature :

Name : Arpit Augustine
Roll No.:B120877CS

Signature :

Name : Gadi Venkata Sai Rahul
Roll No.: B120715CS

Signature :

Name : Harsh Parsuram Puria
Roll No.:B120775CS

Signature :

Name : Vineel Patnana
Roll No.:B120811CS

CERTIFICATE

*This is to certify that the project report entitled: "***PREDICTING DESTINATION OF TAXI RIDES***" submitted by Sri/Smt/Ms* **Arpit Augustine B120877CS, Gadi Venkata Sai Rahul B120715CS, Harsh Parsuram Puria B120775CS**, *and* **Vineel Patnana B120811CS** *to National Institute of Technology Calicut towards partial fulfilment of the requirements for the award of Degree of Bachelor of Technology in Computer Science Engineering is a bonafide record of the work carried out by him/her/them under my/our supervision and guidance.*

Signed by Guide(s) with name(s) and date

Place: Calicut
Date: 09/05/2106

Signature of Head of the Department

Office Seal

Abstract

The taxi industry uses electronic dispatch systems to direct taxis from the taxi stands to the customers. Electronic dispatch systems make it easy to see where a taxi has been, but not necessarily where it is going. In most cases, taxi drivers operating with an electronic dispatch system do not indicate the final destination of their current ride or there could be loss of GPS signal in transit. The presence of information about the final destination of the trip along with the trip time not only improves the service to the customer but also is economical for the taxi service. It can also reduce the idle time of taxi drivers between trips. If a dispatcher knew approximately where their taxi drivers would be ending their trips, they would be able to identify which taxi to assign to each customer request and hence could save both time and money by not sending a taxi all the way from a taxi stand. This project tries to focus only on predicting the final destination of the taxi rides, given their initial partial trajectories and we leave the trip time prediction as a task that could be carried out in future to improve the electronic dispatch systems further. The design and implementation of the various models are discussed along with their results. A comparison is made between the models and further improvements were carried out on the most effective model. Further the tasks that could be done in future are discussed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The taxi industry uses electronic dispatch systems to assign taxis to each pickup request. These dispatch systems traditionally followed broadcast-based (one to many) radio messages for service dispatching which was not economical as a number of taxi drivers would respond to a request. The taxi industry has been evolving rapidly with the shift to unicast-based (one to one) messages from broadcast-based. With unicast-messages, the dispatcher needs to correctly identify which taxi they should dispatch to a pick up location. New competitors and technologies are changing the way traditional taxi services do business. While this evolution has created new efficiencies, it has also created new problems.

Electronic dispatch systems make it easy to see where a taxi has been, but not necessarily where it is going. In most cases, taxi drivers operating with an electronic dispatch system do not indicate the destination of their current ride. Since taxis using electronic dispatch systems do not usually enter their drop off location, it is extremely difficult for dispatchers to know which taxi to contact.

To improve the efficiency of electronic taxi dispatching systems, it is important to be able to predict the destination of a taxi while it is in service. Particularly during periods of high demand, there is often a taxi whose current ride will end near or exactly at a requested pick up location from a new rider. If a dispatcher knew approximately where their taxi drivers would be ending their current rides, they would be able to identify which taxi to assign to each pickup request.

Our problem is to find a scalable framework for the above problem. In chapter 2, we will be discussing the design specifically the cleaning and data reduction. We then went on to describe the clustering approaches which we chose to cluster the GPS coordinates to get a smaller set of coordinates which we use as support points in our predictive models. Finally, we discussed the various predictive models and the improvements carried out on them. In chapter 3, the implementation part of the clustering algorithm, the various predictive models along with the approach followed for improving each model is discussed. Chapter 4 basically lists down the results for each model and tries to compare and analyze each of them. The tasks that could be carried out in the future along with the conclusions are mentioned in chapter 5.

1.1 Problem Statement

In this paper, we try to build a predictive framework that is able to infer the destination of taxi rides based on their initial partial trajectories irrespective of the city. Our approach is to learn the behavior of taxis by building a probabilistic model. Each trip can have hundreds of points, and the inclusion of all trips in the statistical model is desired. Hence it is necessary to reduce each trip to only a few manageable points called "Support Points". Furthermore, this reduction leads to another advantage: the computational time is reduced. Thus before designing the predictive model, we have to find the support points using a suitable clustering algorithm. The pattern of the taxis depends on a variety of factors such as the day of the week, the way the taxi was booked, etc. So given a sequence of GPS coordinates, we need to predict the destination GPS coordinate.

1.2 Literature Survey

In this section, we describe related work on location prediction based on Markov Models. We also discuss the results of these studies.

In [1], the model was developed on the assumption that users live a sedentary life where displacement of a person follows a pattern. The data they worked on consisted of 6 different users. The destination places were labeled as house, office, etc (maximum of 15 to 20 different destinations). The users used different modes of transportation (car, bicycle and foot). The procedure followed by them, comprised of first finding out support points by clustering. These support points would then represent the original coordinates. This approach resulted in a drastic reduction of points which in turn reduced the computational cost. They then applied a Hidden Markov Model with "goal to reach" as the hidden state and "sequence of support points" as the visible states. They observed that if the first few support points were closer to each other, the prediction became harder, but the subsequent points clearly leads to discriminate the destination.

The authors used a hybrid method for predicting human mobility in [2]. This hybrid method was based on HMM. A dataset from Geolife project- a GPS trajectory dataset collected in the context of the GeoLife project from microsoft research Asia, by 178 users in a period of over four years from April 2007 to October 2011. A Hierarchical Triangular Mesh is used to convert the real continuous values associated to the geospatial coordinates of latitude and longitude, into dis rete codes associated to specific regions which makes the learning of HMMs more efficiently. They clustered the data into three clusters according to the temporal period namely weekdays daytime, weekdays night and weekends. These clusters are used to train different HMMs corresponding to different types of location histories, for each cluster. They used MLE to estimate the parameters of HMM. The prediction accuracy of 13.85% with the best performing method and they suggested that the accuracy could be improved if a better clustering technique was used.

In the following paper[3], the authors addressed the issue of predicting the next location of an individual based on the observations of his mobility behavior over some period of time and the recent locations that he has visited. Basically, they extended the mobility model called Mobility Markov Chain(MMC) in order to incorporate the 'n' previously

visited locations and they developed a novel algorithm for next location prediction based on this mobility model that they coined as n-MMC. Their accuracy prediction ranged from 70%-95% when the value of n was chosen to be 2. The accuracy was calculated by dividing the number of correct predictions to the total number of predictions.

One of our approach needed to group taxis based on similarity measure and each group could then be trained to improve our predictive model. Hence we read the following paper [4] in which the authors tried to compare three different similarity coefficients namely Jaccard coefficient, Dice coefficient and Cosine coefficient. A similarity coefficient is a function which computes the degree of similarity between a pair of samples. For example, a similarity coefficient represents the similarity between two documents, two queries, or one document and one query. The conclusion was that the Cosine coefficient is a better measure of similarity than Dice and Jaccard coefficient.

Uber, an online transportation network company also having worked on similar problem to predict the destination of taxi rides. They have published an article[6] where they used Bayesian Model to infer the destination. By this approach they could predict the correct destination 74% of the time.

Chapter 2

Design

In this chapter, we will first discuss the different methodologies adopted to pre-process the data set. Later we explain the clustering algorithms used to obtain the support points. Next we started how we designed the final matrix model starting from the primitive matrix models. Finally we discuss how a forest model is used to predict the final destination.

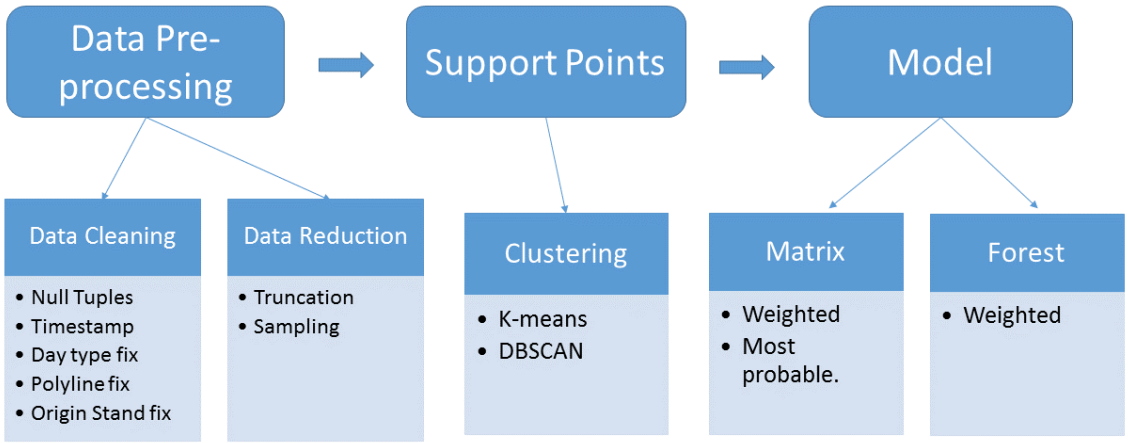

Figure 2.1: Design of Our Model

2.1 Data Pre-processing

In this section, we describe the data set and different techniques used for data pre-processing.

2.1.1 Data

The data set[5] we use contains the trajectories for all the 442 taxis running in the city of Porto, Portugal during a complete year (from 01/07/2013 to 30/06/2014) and is provided by Kaggle.

Each data sample corresponds to one completed trip. It contains a total of 9 (nine) features, described as follows:

1. *TRIP_ID*: (String) It contains an unique identifier for each trip;
2. *CALL_TYPE*: (char) It identifies the way used to demand this service. It may contain one of three possible values:
   - A if this trip was dispatched from the central;
   - B if this trip was demanded directly to a taxi driver on a specific stand;
   - C otherwise (i.e. a trip demanded on a random street).
3. *ORIGIN_CALL*: (integer) It contains an unique identifier for each phone number which was used to demand, at least, one service. It identifies the trips customer if CALL_TYPE=A. Otherwise, it assumes a NULL value.
4. *ORIGIN_STAND*: (integer) It contains an unique identifier for the taxi stand. It identifies the starting point of the trip if CALL_TYPE=B. Otherwise, it assumes a NULL value;
5. *TAXI_ID*: (integer) It contains an unique identifier for the taxi driver that performed each trip.
6. *TIMESTAMP*: (integer) Unix Timestamp (in seconds). It identifies the trips start.
7. *DAYTYPE*: (char) It identifies the day-type of the trips start. It assumes one of three possible values:
   - B if this trip started on a holiday or any other special day. (i.e. extending holidays, floating holidays, etc.)
   - C if the trip started on a day before a type-B day.
   - A otherwise (i.e. a normal day, workday or weekend).
8. *MISSING_DATA*: (Boolean) It is FALSE when the GPS data stream is complete and TRUE whenever one (or more) locations are missing.
9. *POLYLINE*: (String) It contains a list of GPS coordinates (i.e. WGS84 format) mapped as a string. The beginning and the end of the string are identified with brackets (i.e. [and], respectively). Each pair of coordinates is also identified by the same brackets as [LONGITUDE, LATITUDE]. This list contains one pair of coordinates for each 15 seconds of trip. The last list item corresponds to the trips destination while the first one represents its start.

2.1.2 Statistics

This section represents the general statistics of the data set in a tabular form (Table 2.1) like the number of trips and number of different call-types which are used in further pre-processing.

Total number of trips	17,10,670
Number of trips with NULL Polyline	43,904
Number of co-ordinates	7,83,63,691
Resultant number of trips	16,66,766
Number of taxi stands	64
Number of different passengers	57,105
Number of trips with no missing values	10

Table 2.1: Few Stats of the Data

Call_Type	Number of trips
A	3,64,770
B	8,17,881
C	5,28,019

Table 2.2: Number of trips corresponding to each Call_Type

Day_Type	Number of trips
A	17,10,670
B	0
C	0

Table 2.3: Number of trips corresponding to each Day_Type

Table 2.3 shows that number of trips for day-type 'B' and 'C' is zero which is not possible since they represent weekends and public holidays. Thus it is necessary to clean the data before processing it.

2.1.3 Data Cleaning

Data cleaning is an important step in data mining as it helps remove all the noises present in the data set. As a real-world dataset, there are several inconsistencies in the data. The sequence of coordinates for trips can be quite noisy due to the imprecision of the GPS. Noises include the erroneous data or the data which is missing. This process helps to improve the accuracy of the trained model.

1. **Exclusion of tuples**: As there were trips with no polyline values to process them, the trips were removed. As a result, the number of trips reduced by 43,904.

2. **Timestamp conversion**: The unix time stamp field was converted to general day type like month, day, year etc.

Day_Type	Number of trips
A	11,02,229
B	41,704
C	41,336
D	4,81,497

Table 2.4: Number of trips corresponding to Day_Type after fix

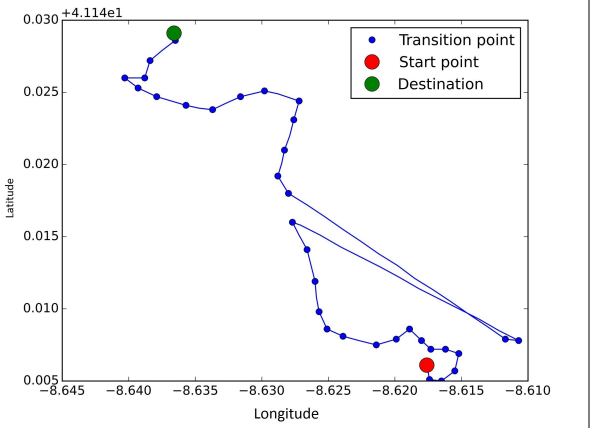

Figure 2.2: Noise Data in Trips

3. **Day_type fixing**: The entries for this field had a value of zero for day-type 'B' and 'C'. So we used public holidays of Porto, Portugal to fix the B and C type entries. We further checked for weekends and changed the entries to D. The D type entry was initially not present.

4. **Polyline fixing**: The speed limit for the city of Porto is 50km/hr. So on an average a vehicle would cover 160m if it is traveling at approximately 40km/hr. For any two pair of coordinates, if the distance between them was exceeding 160m, we added entries between them. For example, if the distance between two points was more than 160m but less than 320m we added one entry by taking the midpoint and if the distance was between 320m and 480m, we added two points which were equidistant from each other and so and so forth.Number of coordinates after cleaning was approximately 11 crores.

   There were a few trips in the data set in which the distance between two consecutive points (say point A and B) is greater than 1 km but the distance between point A and the point next to B is much smaller. This could be because of a glitch in the GPS device which gives a wrong GPS coordinate in between and then starts giving the correct coordinates. Since the number of such trips were less compared to the whole data set, these trips were removed from the training set.

   In a certain number of trips the distance between two points is more than 1 KM. This happens whenever taxi driver switches off the GPS during the trip or due to malfunction of GPS device. Inserting points equidistant to each other was an ineffective way for larger gaps. We found that only 3 percent of data has huge gaps.

Thus, we removed them from our training set.

5. **Data Segregation**: Data was segregated based on the Day_Type value so that each group of trip can be clustered separately. Since there could be particular trends on weekdays, weekends and during public holidays, such approach will give a more customized result. This also reduces the computational cost.

6. **CALL_TYPE *'B'* ORIGIN_STAND Fix**

   In CALL_TYPE *B*, there is a attribute called ORIGIN_STAND which tells from which taxi stand the taxi started the trip. But for some trips the value of this field was null. To fix this problem, we found the coordinates of each taxi stand and then the taxi stand which was closest to the starting coordinate of the trip (using haversine distance) was assigned the ORIGIN˙STAND for that trip.

2.1.4 Data Reduction

In the unprocessed data, the distance between two subsequent coordinates is 160m. This resulted in too many points for clustering. So we decided to consider 1km as the distance between two consecutive coordinates. After clustering there were too many clusters between two consecutive points. This resulted due to losing of too much information by reduction. Hence we stuck with our previous 160m approach.

- Number of coordinates with 160m as the distance: approximately 11 crores
- Number of coordinates with 1km as the distance: approximately 2.82 crores

Truncation

Truncation upto 4 digits(approximately 7mts) after decimals places of GPS coordinates has been done. This resulted in better clustering.

Sampling

Clustering was carried out on different months of data separately to check if the pattern of the trips followed in various months was different or not. We observed that there wasn't much variation in data of different months. Therefore, it was safe to assume that data of two months would be sufficient for training the model.

2.2 Predictive Model

In this section, we are going to discuss in detail about the various models which we used in predicting the final destination.

2.2.1 Clustering

Clustering is the task of grouping the objects in such a way that objects in the same group are more similar (in one or more sense) than the objects in the other group.

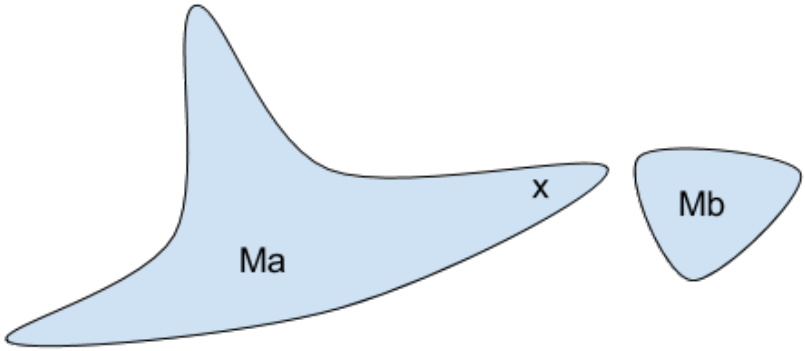

Figure 2.3: Limitation to DBSCAN

As we have mentioned in the previous section, our data consists of the GPS coordinates. Thus we need to find a representative point(support point) for a group of points. Few of the clustering algorithms which can be used for clustering data consisting of GPS coordinates are PAM, CLARA, DBSCAN and k-means.

Both PAM (Partitioning Around Medoids) and CLARA (Clustering Large Applications) are types of k-medoid clustering. In k-medoid clustering, the center point or the representative point is the most centrally located object in the cluster. PAM is used generally for small data sets and has high computational cost. CLARA on the other hand, is used for large data sets and works similar to PAM by taking a sample of the data set. So, the result of clustering will depend on how well the data is sampled.

Density-based spatial clustering of applications with noise (DBSCAN) is a density-based clustering algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away). DBSCAN is one of the most common clustering algorithms. It is a good method for Geocoordinates clustering as it works bottom-up by picking a point and looking for more points within a given distance. It then expands the cluster by repeating this process for new points until the cluster cannot be further expanded.

One particular problem with DBSCAN is that, given a new data point it is difficult to predict which cluster it belongs to. We used haversine distance (Section 3.2) to predict the cluster which is nearest to the given point. But if the cluster size is too big (in case of highly dense areas), the distance of the point from some other cluster may come out to be less than the cluster to which it actually belongs. For instance in the Figure 2.3, the point 'x' is more closer to the support point 'Mb' than the support point 'Ma'. So haversine distance approach will result in 'x' belonging to cluster 'Mb' and not 'Ma'.

k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells. *k-means* method allow us to predict the cluster, given a new data point. But we have to decide the number of clusters before starting the clustering process. Since the data set is too

large, deciding the number of clusters is difficult. Also *k-means* clustering is sensitive to outlier data.

2.2.2 Support Points

Data of 3 months was taken to cluster and support points were found out. All the coordinates in the data set was then replaced by their corresponding support point. Clustering of the starting points, transition points and end point was done separately because there were high density of start points and end points in some areas and if clustering is done for all the points together, the clustering process will not be efficient.

2.2.3 Primitive predictive model

In this subsection we will show how we have applied different clustering algorithms and the methods to our model. We have considered only the polyline to train the models and no other attributes present in the data set have been used.

1. In the first model, we only consider start points to predict the destination of a trip and found out the maximum probable destination cluster.

2. In our second model, instead of a single highest probability for start points, the product of each coordinates and its respective probability was taken to obtain the weighted destination coordinate for each trip. For instance, given a starting point S_i we have a list of end coordinates $\{E_1,E_2,...,E_n\}$ and corresponding probability $\{p_1,p_2,...,p_n\}$ to reach the destination from the matrix mentioned in 2.2.3.a. Destination D is given by

$$D = \sum_{i=1}^{n} p_i E_i \tag{2.1}$$

3. Now, we extend the first model by considering even the transition points of the trips along with the start points. That is now we have start point and transition point to predict the destination. In testing, we considered only most recent transition point to determine the destination under the assumption that the path in which this transition point is reached from start is not so improvement.

4. We extend the second model by considering even the transition points of the trips to train and most recent transition point and start point to test.

2.2.4 Improvements

In addition to the models discussed above, we also considered specific cases that could improve the efficiency of the model.

Taxi grouping

There are around 430 different taxis in the city. We found few taxis which were similar in the way they operate within the city. Thus grouping similar taxis will narrow down and improve destination prediction. So to check the pattern between taxis, we considered

taxis and their taxis stands. A taxi can start from multiple taxi stands, thus we took the feature vertex for each taxi as number of times it started from the taxi stand. To measure the similarity among taxis, we used cosine coefficient to obtain similarity between taxis. Cosine coefficient is a better similarity measure as compared to Jaccard coefficient as discussed in the literature survey.

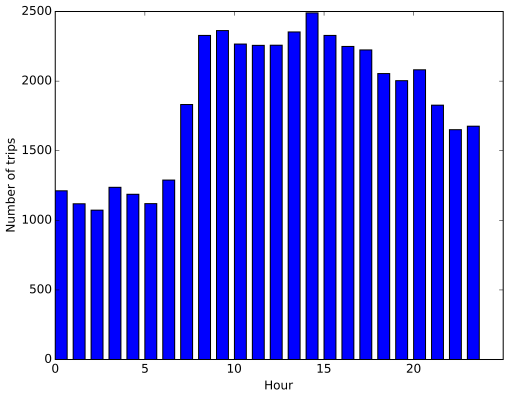

Figure 2.4: Hours vs Number of Trips

Time Segregation

The data set was divided on the basis of time the journey was started. We decided to divide into 4 divisions after analyzing Figure 2.4. Dividing 24 hours into 4 time ranges (0AM-6AM, 6AM-12PM, 12PM-6PM, 6PM-0AM), we take the taxis starting in one particular time range and train the model. This is done to find patterns in the data and to train accordingly. This is carried out for all the four time ranges and the average result is taken.

Day-type Segregation

In the data set, we have four different day types which were labeled as A,B,C,D. Taxi routes follow different patterns depending on the type of the day whether its a holiday or a weekend, etc. To exploit this information the model was trained for each of the day types separately.

Call-type Segregation

The data set has been segregated into three different parts, based on the way the taxi was booked, which were labeled as Call type A, Call type B, Call type C. So based on the method followed for taking a taxi, different patterns could be observed. Hence, we decided to train the model for each of the call types separately.

One of the limitation for this model is that it does not take it into consideration the direction of the route which the taxi follows. It fails for Round trips like the one shown

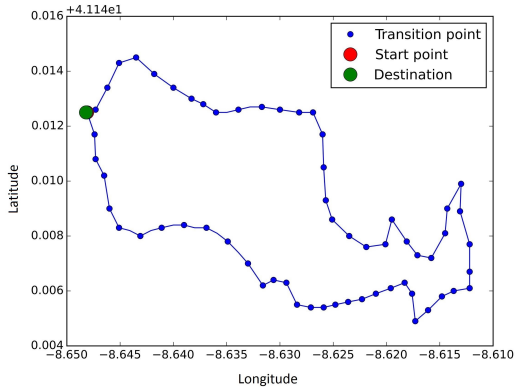

Figure 2.5: Round Trips in the Data

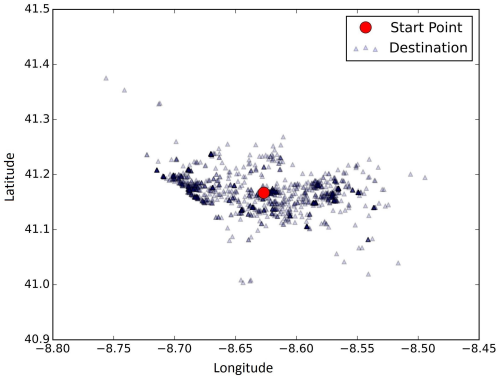

Figure 2.6: All destinations for a Starting cluster

in Figure 2.5. Figure 2.6 shows the start point cluster and all destinations that can be reached from it. We can clearly see that there are quite a few round trips in our data set from Figure 2.6. The next model covers this aspect.

2.2.5 Model Forest

In this model, we construct a forest where the number of trees equals number of start clusters. Each tree root node is starting cluster label and intermediate nodes are transition clusters label and leaf nodes are the destination cluster label. Each node in the tree contains the cluster label, weight (number of times the node being visited) and pointer to its children nodes. For any given Trip T1 = {S,t_1,t_2,t_3...,t_n,E}(S is starting cluster number, t_i be i^{th} transition cluster number and E be the destination cluster), the tree with root node S is selected and t_1 added to the tree such that t_1 is child of S. Next t_2 is added so that t_2 is child of t_1 and so on.

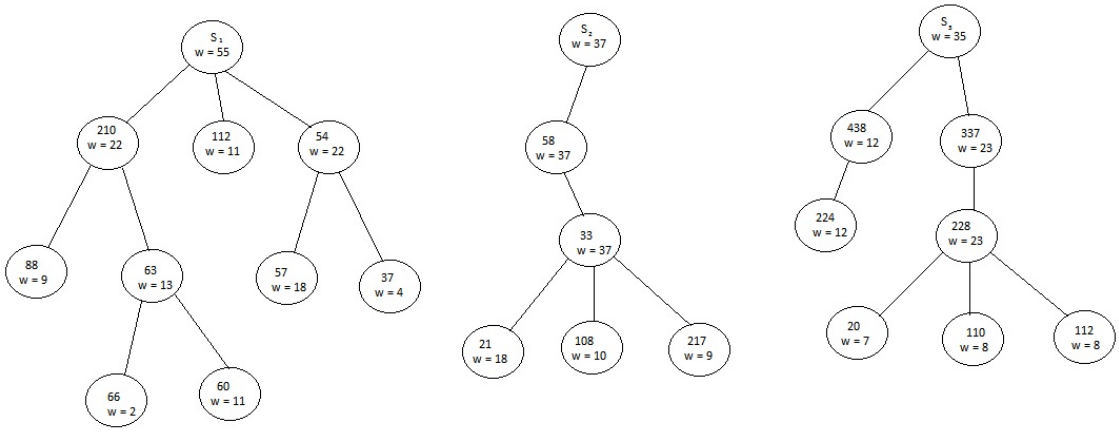

Figure 2.7: Model Forest Example

Training

Constructing of the Forest

Testing

For any given Trip T1 = {S,t_1,t_2,t_3...,t_r} matching was done with Tree S till t_I matches along levels. If further match was not possible or it matched till t_r then all the possible leaf nodes from that node are considered and weighted approach similar to what mentioned earlier in Eq: 2.1 i.e product of each probability and the respective destination coordinates is applied to find the destination.

Chapter 3

Implementation and Results

3.1 Implementation

In this section we discusses the various implementation details as to how the data set was stored, accessed and modified to build and train the model. Since the data set was large with around 10 features and approximately 17 lakh tuples, MongoDB was used to store the data. Python was used to connect with the database.

3.1.1 Clustering

k-means and DBSCAN clustering algorithms were implemented using the scikit python package. DBSCAN has two input parameters, minPts, the minimum number of points which should be present inside the cluster and the epsilon value. The value of minPts was taken as 10 and the epsilon value was fixed at 0.003. For transition points, DBSCAN was not used because there are clusters of very large size due to clustering of all coordinates in high frequent trips into a cluster.

Since k-means clustering algorithm requires the number of clusters as the input parameter,DBSCAN was used to get a rough approximation of these initial values. To further improve the number of clusters, a hit and trial method was followed with the condition that the distance between two supports points is always greater than the distance between two consecutive points. The number of clusters obtained from this method are -

- Start points - 250 clusters
- Transition points - 500 clusters
- Destination points - 250 clusters

For Call-type 'B', instead of clustering the starting points we used the taxi stand id to group the points. Haversine distance was calculated between the starting point and the taxi stand. The taxi stand which was closest to the point was taken as the support point for that starting coordinate.

3.1.2 Taxi grouping

Using the matrix we created as mentioned in section 2.4 . We tried to group taxis using cosine similarity. We add the first taxi directly to the list of similar taxis.Now we traverse the taxi particular row in the cosine similarity matrix and add the taxi which is most similar to the given taxi and satisfies the minimum similarity coefficient value with all other taxis present in the similar taxi list.The above process is repeated untill the list of similar taxis do not change. This process of creating list of similar taxis is carried out untill all the taxis are put in one group or another. We found around 61 groups. About 40 taxis were unable to fit in any group.

3.2 Results

In this section, we first discuss the primitive predictive models and their results. Later the results obtained after applying different improvements as discussed in section 2.2.4 is present in the tabular form. Also, the result of the forest model is presented. The evaluation metric for the proposed model is the **Mean Haversine Distance**. The Haversine Distance is commonly used in navigation. It measures distances between two points on a sphere based on their latitude and longitude.

The Harvesine Distance between the two locations can be computed as follows

$$a = \sin^2(\frac{\theta_2 - \theta_1}{2}) + \cos(\theta_1)\cos(\theta_2)\sin^2(\frac{\phi_2 - \phi_1}{2}) \tag{3.1}$$

$$d = 2.r.a.\tan(\sqrt{\frac{a}{1-a}}) \tag{3.2}$$

where θ is the latitude, ϕ is the longitude, d is the distance between two points, and r is the sphere's radius (Earth's radius).

Training and Testing

Data of 60 days has been taken to train the model. The model was then tested on the next 7 days of data. The above process was repeated 10 times by moving both the training and testing data forward by a window size of 7 days. We assume data of 60 days is sufficient to test immediate 7 days.

3.2.1 Primitive Model

In this subsection, we show our results for the primitive models for different clustering algorithms.

A. Predicting destinations using the starting coordinate

Using training data, we created a matrix to store the probabilities of reaching a destination given a starting point.

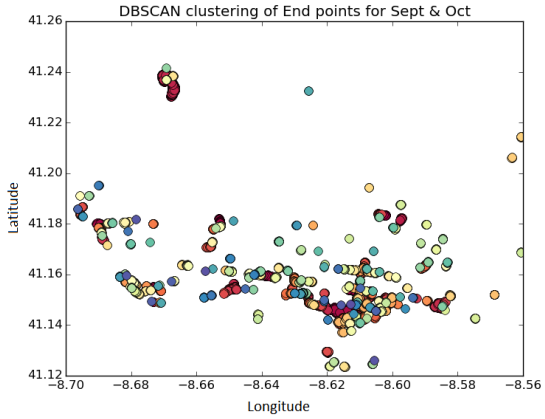

Figure 3.1: DBSCAN clustering of destinations

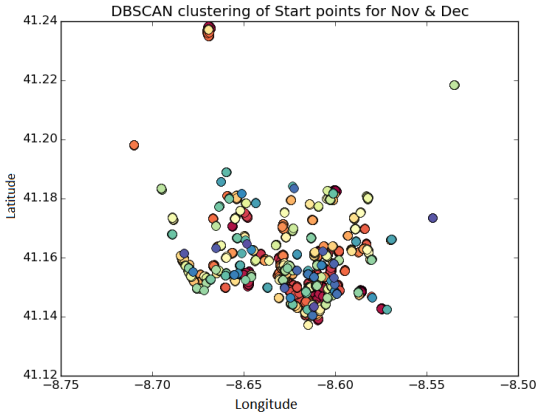

Figure 3.2: DBSCAN clustering of start points

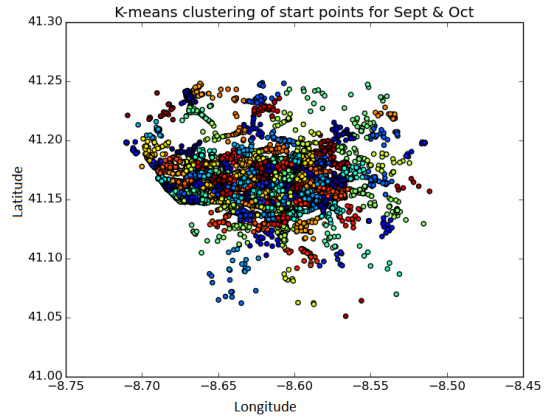

Figure 3.3: *k-means* clustering of start points

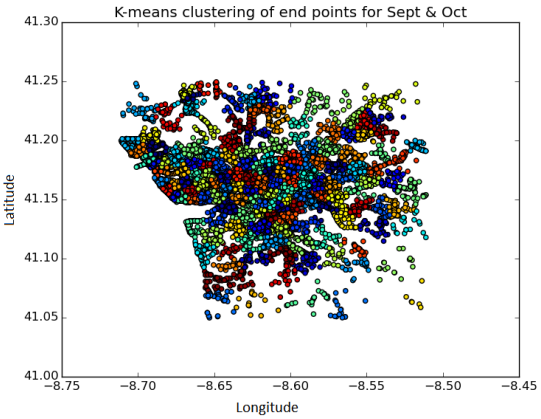

Figure 3.4: *k-means* clustering of destinations

1. **Using k-means**
   k-means clustering algorithm was used to find start points (Figure 3.3) and destination points labels (Figure 3.4) and then matrix is created. The result obtained after running the model was **4.95 kms**.

2. **Using DBSCAN**
   DBSCAN clustering algorithm was used to find start points (Figure 3.2) and destination points labels (Figure 3.1) and then matrix is created. To evaluate our model, we predicted destinations and found the variation from the actual end coordinate to be **3.13 kms** on average.

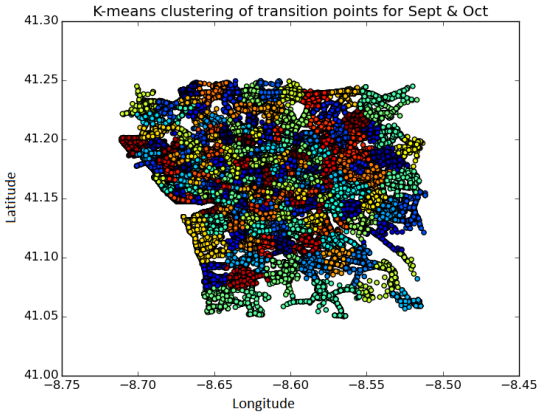

Figure 3.5: *k-means* clustering of transition points

B. Predicting destination using starting point and weighted destination approach

This model is similar to above one except that weighted destination approach is considered rather than maximum probable

1. **Using k-means** The result obtained by *k-means* approach was **3.14 kms**

2. **Using DBSCAN** The result obtained after running the above model with DBSCAN (Figure 3.1 & Figure 3.2) was **3.04 kms**

C. Predicting destination using starting point and 'n' intermediate support points

In this model, a transition probability matrix was created with all possible combination of starting, transition and end cluster.

1. **Using k-means** To evaluate our model, the testing data was used and found the variation from the actual end coordinate to be **2.84 kms** on average.

2. **Using DBSCAN** Here DBSCAN clustering algorithm was used to cluster just the starting (Figure 3.2) and end (Figure 3.1) coordinates .Transition points were clustered like before using *k-means* (Figure 3.5) because in DBSCAN high frequent routes are being clustered as one big cluster and also it cannot handle the large number of transition points. The result obtained after running the model was **3.24 kms**

D. Predicting destination using starting point, 'n' intermediate support points and weighted destination

In this model, instead of checking a single highest probability, the product of each probability and the respective coordinates was taken to obtain the weighted destination co-

ordinate for each trip.

1. **Using k-means** The result obtained by *k-means* (Figure 3.3, Figure 3.4 and Figure 3.5) approach was **2.61 kms**

2. **Using DBSCAN** The result obtained after running the model with DBSCAN for start and destination clusters,(Figure 3.1 and Figure 3.2), and *k-means* transition cluster was **3.16 kms**

Since the best result was found to be **2.61 kms** using the model which used k-means to cluster all the three type of points, n intermediate support point and weighted destination, we considered this model for further improvements.

The **GOOD-AVG-BAD** column in the table 4.1 and table 4.2 represents the percentage of the testing data where -

- **GOOD**: Distance between predicted destination and the actual destination is less than 2 kms.
- **AVG**: Distance between predicted destination and the actual destination is between 2 kms and 5 kms.
- **BAD**: Distance between predicted destination and the actual destination is greater than 5 kms.

3.2.2 Predictive Model Matrix

After applying the improvements as mentioned in section 2.2.4, the model was trained and table 3.1 shows the results obtained.

Type	Maximum Probable	GOOD-AVG-BAD (in %)	Weighted Probable	GOOD-AVG-BAD (in %)
Call_Type A	2.274	64.3 - 21.4 - 14.2	1.954	65.7 - 27.3 - 7.0
Call_Type B	2.426	64.2 - 20.9 - 14.9	1.938	68.3 - 24.4 - 7.3
Call_Type C	3.582	52.0 - 24.4 - 23.6	2.920	53.4 - 32.7 - 13.9
Call_Type C & Day_Type A	3.727	51.6 - 23.0 - 21.3	3.142	51.1 - 33.0 - 15.9
Call_Type C & Day_Type D	3.358	52.5 - 26.0 - 21.6	2.782	54.1 - 32.5 - 13.4

Table 3.1: Results after data segregation

3.2.3 Predictive Model Forest

As mentioned in section 2.2.5, the tree model was trained and the table 3.2 shows obtained results.

Type	Maximum Probable	GOOD-AVG-BAD (in %)
Call_Type A	2.274	64.3 - 21.4 - 14.2
Call_Type B	2.426	64.2 - 20.9 - 14.9
Call_Type C	3.582	52.0 - 24.4 - 23.6
Call_Type C & Day_Type A	3.727	51.6 - 23.0 - 21.3
Call_Type C & Day_Type D	3.358	52.5 - 26.0 - 21.6

Table 3.2: Results for Model forest

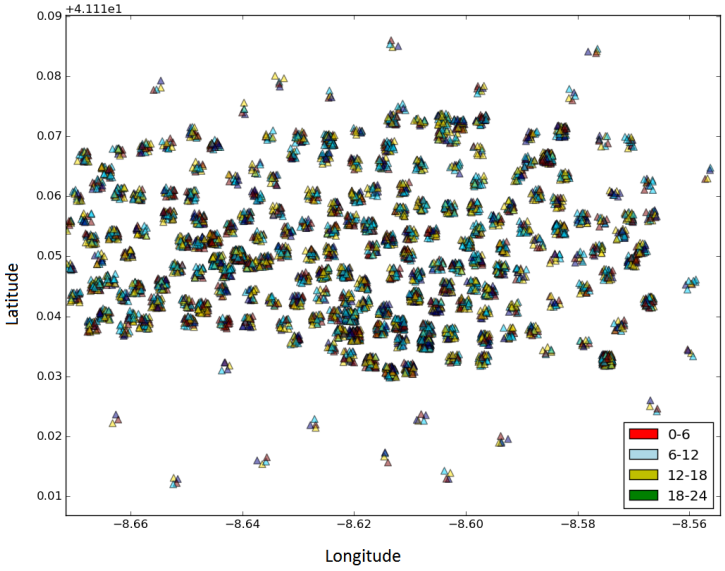

Figure 3.6: Clusters based on time ranges

3.2.4 Discussion

To summarize our observations, we found out that for the taxi trips which were either booked by calling the dispatcher or by going to the taxi stand and hiring a taxi, the results were found to be good with approximate error rate of 1.9 kms. These trips accounted for 66% of the total trips. Whereas for the situation in which the taxi was booked from the street, the error rate (2.9 kms) turned out to be average. This was expected as the average distance of trips under this call type was 7 kms which is more than that of the other call types. So there was a higher chance of deviating from the correct path.

Each day was further divided into four time zones and start support point of all trips in each of the time zones are shown in Figure 3.6. What we found out was that in certain time zones there was a high probability of trips starting from certain support points. Only in such cases are predictions improved.

We tried to group taxis which are similar to one another on the basis of their starting taxi stand. The threshold similarity between taxis was taken to be 0.6 and groups

of similar taxis were formed. On training and testing these group of taxis individually, a significant improvement was observed in a very few groups.

Chapter 4

Conclusion

In day to day scenario, most of the taxi drivers operating with an electronic dispatch system do not indicate the destination of their current ride which makes it extremely difficult for the dispatchers to know which taxi to contact. We tried to solve this problem by predicting the final destination coordinate given the coordinates of the initial partial trip for such cases. Instead of predicting the exact final coordinate, we tried to predict the representative or the support point of the final coordinate. This model would work on any city because we didn't use any aspect of the city. It was also observed that some features like which day of the week the ride was taking place and how the taxi was booked by the customer had a major role to play in the final destination of the customer. Other features like individual taxis and individual customers may have a particular pattern during a particular week of the month or during a week of the day but such parameters were not used in our model to predict the destination. The result obtained by testing the model on a month of data which was not used for training gave a result of 2.26 kms. Also, the result obtained on submitting the model on kaggle was 2.75 kms. The best score on kaggle was 2.14 kms.

4.1 Future work

The model could be further improved by using an ensemble method of the two models discussed above to the two classifiers. Secondly, the time factor could also be used as one of the features for classification if it is clubbed with other features like taxis or particular individual to give a better result. A more specific predictive model too could be built for each city using some features like tourist destinations, transport hubs, shopping centres for a particular city.

Bibliography

[1] Alvarez-Garcia, Juan Antonio, et al. *Trip destination prediction based on past GPS log using a hidden markov model*, Expert Systems with Applications 37.12 (2010): 8166-8171.

[2] Mathew, Wesley, Ruben Raposo, and Bruno Martins. *Predicting future locations with hidden Markov models*, Proceedings of the 2012 ACM conference on ubiquitous computing. ACM, 2012.

[3] Gambs, Sbastien, Marc-Olivier Killijian, and Miguel Nez del Prado Cortez. *Next place prediction using mobility markov chains*, Proceedings of the First Workshop on Measurement, Privacy, and Mobility. ACM, 2012.

[4] Thada, Vikas, and Vivek Jaglan. *Comparison of Jaccard, Dice, Cosine Similarity Coefficient To Find Best Fitness Value for Web Retrieved Documents Using Genetic Algorithm*, International Journal of Innovations in Engineering and Technology (2013).

[5] Data Set from Kaggle `https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i/data`

[6] *Making a Bayesian Model to Infer Uber Rider Destinations* by Uber `https://newsroom.uber.com/inferring-uber-rider-destinations/`