CAB FARE PREDICTION USING HAVERSINE FORMULA

VinodkumarChintala (L00157096), Balasubramaniyam Muthuramalingam (L00157060), LYIT.

I. Introduction

Artificial Intelligence (AI) is one of the emerging fields shaping the transportation sector. It is widely applied in public transportation, for example in cab services, companies use AI for calculating the distance from starting point to the end point and determine the price for the trip. Due to increase in demand of cabs, AI has shown significant development in this area. The distance is calculated by Haversine formula, given the latitude and longitude. The distance is directly proportional to the price, i.e., the value of the price increases with distance. By applying Decision Tree and Random forest tree and Linear Regression algorithms, fare value for a trip is predicted. Further, to reduce the prediction error and to increase the accuracy speed Gradient Boosting technique is implemented. Setting up a target outcome for the future model will minimize the error. In order to setup a target outcome , for each iteration in the data depends on the amount of changes that affects the overall prediction error. If the change in prediction is small for a case it causes a large error drop so the predictions from the next model will produce values closer to target outcome which will reduce the error. Finally, if the small change of prediction causes no change in error, then the next outcome of the case is zero but it doesn't mean it decreases the error. At the end the model is optimized again using parameter tuning, hyperparameter optimization is implemented to customize the model to fit the dataset for a specific task. Hyperparameters are set manually by the user to guide the learning process. Two optimization algorithms are used to during hyperparameter tuning, Random Search and Grid Search. Random Search selects a domain which is also called a search space and randomly samples the points present in the domain. In Grid search the search spaces is defined by a grid and every position in the grid is evaluated. The performance is measured using an error whether it is minimized with zero error which represents it's a perfect a model. Good performance are small negative values which is close to zero and perfect model is zero. Random search is preferred as it provides new hyperparameter values or new combination but grid search is quick to implement and appropriate for smaller datasets.

II. LITERATURE REVIEW:

There are reports related to the models predicting the price for cabs and other means of transportation systems. Anastasios Noulas's group posted a paper in taxi price comparison exploiting their insights and challenges between Uber and Black cabs in mobile app. It showed that price charged by Uber were more stable than Black cabs as there were more deviations in Black cabs. It turns out Uber was stable when compared to Black cabs i.e., it provided accurate price for a trip without any surcharges most of the time. These prices are predicted with and without reduction co-efficient. There are four different factors involved in calculating the reduction co-efficient, availability, degree of necessity, time horizon and proportion of budget. Since Black cabs prove to be more deviated so the price gain overall is calculated by taking the difference between Uber price and Black Cab price dividing by Black cab price.

Another paper posted by Yash Indulkar, Analysis of Uber and Ola cabs using Deep Learning. Type of analysis is Sentiment Analysis; it is defined as the basic emotion of a human and trying to understand its problems. It focuses on understanding the customer's complaint registered in Uber and Ola cab services and tweets, it is categorized as positive input or negative input. Two algorithms are used to determine the type of emotion/sentiment, Deep Feed forest neural network and Convolution neural network. Using the training dataset, the model is trained to and then tuned to work with testing dataset.

A paper published by State Islamic University, shows the use of Haversine formula applied to count the visitor in an event. Haversine formula is used to calculate the distance between the user and the events that exist. The existing events are influenced by number of visitors registered in that geographic location . Using Location based service (LBS) user's locations are saved, it's a new technology used in Android applications. The result of Haversine is calculated with the radius of the area of event thus it can provide presence of visitors.

III. Datasets

Datasets are contents of information stored in a tabular form. It is represented in .csv format. In this model two datasets are used; one is used for training the model and other is used to test the model. The training dataset is used to check if it is compatible with the respective algorithms, Decision tree, Random forest and Linear Regression algorithm. In the end it is optimized and processed using gradient boosting and parameter tuning.

A. Training Dataset

In this dataset, the model is trained. The raw file is uploaded and analysed to understand the contents, total size of the data is 16074 entries and further categorized into rows and columns (16067 rows and 7 columns).
There are 7 classes of different datatypes. 6 Float 64 types and 1 Object types before pre-processing.

```
fare_amount          float64
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count      float64
dtype: object
```

The below image is a sample set of information from the training dataset which is recorded from the year 2009 to 2015.

fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.84161	40.712278	1
16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.782004	1
5.7	2011-08-18 00:35:00 UTC	-73.982738	40.76127	-73.991242	40.750562	2
7.7	2012-04-21 04:30:42 UTC	-73.98713	40.733143	-73.991567	40.758092	1
5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.783762	1

B. Test Dataset:

After training the model using the training dataset, the fare is predicted for the test dataset. This dataset for testing purpose and to check the accuracy of the model. The fare is determined using the test dataset, it is tuned further for accuracy. There are 9914 rows and 6 columns. Each attribute is considered as classes with their own datatype (Object, Float64).

pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
2015-01-27 13:08:24 UTC	-73.97332001	40.76380539	-73.98143005	40.74383545	1
2015-01-27 13:08:24 UTC	-73.98686218	40.71938324	-73.99888611	40.73920059	1
2011-10-08 11:53:44 UTC	-73.982524	40.75126	-73.979654	40.746139	1
2012-12-01 21:12:12 UTC	-73.98116	40.767807	-73.990448	40.751635	1

Sample entries of the dataset before processing.

```
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count        int64
dtype: object
```

Attributes:-
pickup date time: Starting time of cab ride.
pickup longitude: Co-ordinate of pickup with respect to east and west
pickup latitude: Co-ordinate of pickup with respect to north and south
fare amount: Price for the trip
dropoff longitude: Co-ordinate of drop point with respect to east and west
dropoff latitude: Co-ordinate of drop point with respect to north and south
passenger count: number of passengers in a cab during the trip

IV. Methodology

A. Overview:

Cab price is determined by calculating the distance from starting point to ending point. It is a regression model, supervised learning. The datasets contain the latitude and longitude of the trips, using the co-ordinates we can determine the distance in kilometres(km). Using Haversine formula distance is calculated and stored in a new memory variable. Decision Tree algorithm, Linear Regression and Random forest algorithm is applied to train the model. This process is iterative until the model is familiar with the dataset, this helps the model to understand the task of the problem. Once the algorithm is applied to the model, mean squared error and R value are determined to predict the accuracy of the model and the deviation value or error value. Finally, the model is skimmed and optimized by tuning it. Gradient boosting and Parameter tuning are applied for the processed data to reduce the error rate and increase the speed of the accuracy i.e., the computation time required to provide the output. Under hyperparameter tuning two different optimization algorithms are applied, Grid Search and Random Search, this is done to measure the performance of the model using both the algorithm and after each iteration it records error value until it reaches zero which is considered as perfect model.

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	16067.000000	16067.000000	16067.000000	16067.000000	16012.000000
mean	-72.462787	39.914725	-72.462328	39.897906	2.625070
std	10.578384	6.826587	10.575062	6.187087	60.844122
min	-74.438233	-74.006893	-74.429332	-74.006377	0.000000
25%	-73.992156	40.734927	-73.991182	40.734651	1.000000
50%	-73.981698	40.752603	-73.980172	40.753567	1.000000
75%	-73.966838	40.767381	-73.963643	40.768013	2.000000
max	40.766125	401.083332	40.802437	41.366138	5345.000000

B. Latitude and Longitude:

Latitude is geographic co-ordinates of north and south position on earth's surface. The angle of latitude ranges between 0° and 90°. The circles parallel to the equator are constant lines of parallel, latitude is used together with longitude to find the accurate location. Longitude is like latitude, but it specifies the co-ordinates of east and west on earth's surface. The angle of longitude ranges between 0° and 180°. The lines are represented in vertical with slight curvature.

C. Pre-processing of Data:

In this stage the datasets are observed and compared with the starting variable and target variable. If the data consists of any NaN values, it Is removed and aligned to provide a better presentation. While exploring the data, the following steps are involved, data cleaning [dropping invalid values], data sorting, eliminating missing values and outlier analysis.

D. Data Splitting:

The training and testing dataset is split further for modelling,

```
# train test split for further modelling

x_train, x_test, y_train, y_test = train_test_split( dataset_train.iloc[:, dataset_train.columns != 'fare_amount'],
                                                     dataset_train.iloc[:, 0], test_size = 0.20, random_state = 1)

print(x_train.shape)
print(x_test.shape)

(12339, 7)
(3085, 7)
```

E. Outlier Analysis:

Values that exceeds the defined range are dropped along with the missing values. This phase is called the missing value analysis, where the dataset is observed to find entries that does not enclose within the defined range. All the classes are analysed to check for NaN values and out of range values, the missing value percentage will increase if all the out of range values are defined as NaN which will lower the standard of dataset so to maintain the percentage of the missing values, out of range values are marked as outliers and removed using is.null() method.

Training Dataset

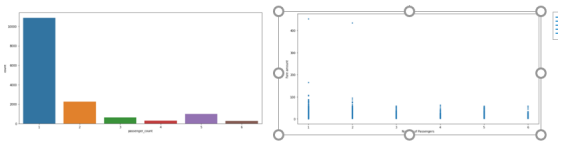
```
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
Year                 0
Month                0
Date                 0
Day                  0
Hour                 0
Minute               0
dtype: int64
```

Testing Dataset

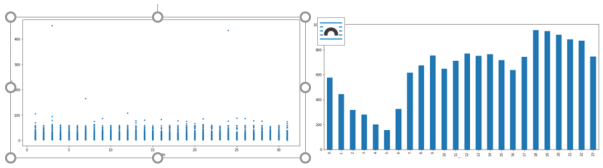
```
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
Year                 0
Month                0
Date                 0
Day                  0
Hour                 0
Minute               0
dtype: int64
```

F. Data Visualization:

Data is represented in graph format, this is used to analytical and comparing purposes. Firstly, passenger count is represented, this is used to further to find the relation between the fare for a passenger with respect to the passenger count.



The following observations are made:
Passengers travelling alone or group of two are more frequent than others, i.e., passenger count for most of the entries is lesser than 3 but greater than 0. Fare price are higher when passengers count is single or double. The data is now further observed to check if the fare price is also dependent on date and time.



V. ALGORITHMS USED

A. Decision Tree Algorithm

Decision tree (DT) is a supervised learning, and it is used to solve classification and regression problems. It is a tree representation where each node is a class label. It has a root node and branch node. DT can represent any Boolean function on discrete values. After generating each set of branch nordes, DT struggles to choose a root node for respective branches.
There are two ways to determine the root node for branch nodes, Information Gain and Gini Index. Information gain, when a node is used in DT to split the training instances into smaller subsets, the randomness of the information changes i.e., entropy. It is defined as change in entropy, lower the change better to find the suitable root node.
The data is split into two axis, x test and y test, similarly for training dataset x train and y train. Calling Decision tree regressor to predict the data and stored in new variable.

```
deci_tree_model = DecisionTreeRegressor(max_depth = 2).fit(x_train,y_train

# Prediction on train data

prediction_train_dt = deci_tree_model.predict(x_train)


# Prediction on test data

prediction_test_dt = deci_tree_model.predict(x_test)
```

B. Random Forest Algorithm

Random forest (RF) is made of trees and increased trees results in robust forest. Similarly, it creates decision trees on datasets and then gets the prediction from each node and finally selects the suited node by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=300, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

```
prediction_train_rf = rand_fore.predict(x_train)

# Prediction on test data

prediction_test_rf = rand_fore.predict(x_test)
```

C. Linear Regression

Linear regression is a statistical model. It analyzes the linear relationship between two variables with a given set of dependent variables and independent variables. Linear relationship between variables is defined as the value of one or more independent variables will increase or decrease, the value of dependent variable will also increase or decrease accordingly. It can also be represented in mathematical terms : Y = mX+ b. The linear relationship between two variables can be positive or negative. Here, two attributes are assigned to X and Y coordinates, X as Years and Y as Male. A function is defined where the slope and intercept for the model is calculated.

```
[ ] # Building model on top of training dataset
    model = LinearRegression().fit(x_train , y_train)

[ ] # Prediction on train data
    pred_train_LR = model.predict(x_train)

[ ] # Prediction on test data
    pred_test_LR= model.predict(x_test)
```

D. Gradient Boosting

It is a technique creating attention for the speed of accuracy and prediction and often used to optimize large and complex

datasets. In gradient boosting the optimal solution is obtained by iterative process, the next best possible model and combined with the older model to reduce the prediction error. Target outcomes are set and calculated for each depending on how much the data prediction has changed and it impacts the overall prediction errors. If a small change causes large load on error, the target value is set to high. Otherwise, the model shows a small changes that causes no error, then the outcome for the next case is considered as zero. Changing the prediction does not decrement the error rate. Creating a Gradient model for training data and testing data and assigning to a new variable which is considered as a object to call the gradient model. Further we calculate the R value and Mean squared value to determine the error rate and accuracy of the model.

```
[ ]  # Prediction on train data

     pred_train_GB = GB_MODEL.predict(x_train)


     # Prediction on test data

     pred_test_GB = GB_MODEL.predict(x_test)
```

```
[ ]  # RMSE for train data

     RMSE_train_GB = np.sqrt(mean_squared_error(y_train, pred_train_GB))


     # RMSE for test data

     RMSE_test_GB = np.sqrt(mean_squared_error(y_test, pred_test_GB))
```

E. Parameter Tuning

Using hyperparameter tuning the model is customized for specific dataset. In hyperparameters the values are set manually to guide the learning process. In a model, it has wide range of hyperparameters and interact in a non-linear pattern. So it is required to search for perfect set of hyperparameters to provide an optimized solution for the model. It is done by applying optimizing algorithms Random Search and Grid Search.

```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state = 42)
from pprint import pprint

# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(rf.get_params())
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

F. Random Search

It randomly samples points in the defined search space , It is applied by importing SickIt library , the model is evaluated for given set of hyperparameter vector using cross validation (CV). Two arguments are required to optimize the model, one is values of hyperparameter set and second is the search space which is called the domain. Applying random search CV in Random Forest

```
rand_grid = {'n_estimators': n_estimator,
             'max_depth': depth}

randomcv_rf = RandomizedSearchCV(RRF, param_distributions = rand_grid, n_iter = 5, cv = 5, random_state=0)

randomcv_rf = randomcv_rf.fit(x_train,y_train)
prediction_RRF = randomcv_rf.predict(x_test)

view_best_params_RRF = randomcv_rf.best_params_

best_model = randomcv_rf.best_estimator_

predictions_RRF = best_model.predict(x_test)
```

G. Grid Search

It evaluates every hyperparameter values in the grid within the search space. Grid search is efficient if the dataset isn't complex and small. It is great for spot checking combination within the domain. Using grid search is like random forest. The only difference is the search space should be discrete grid to be searched. Importing grid search CV and assigning an object to it.

```
[ ]  from sklearn.model_selection import GridSearchCV

     # Grid Search CV for random Forest model

     regr = RandomForestRegressor(random_state = 0)
     n_estimator = list(range(11,20,1))
     depth = list(range(5,15,2))
```

VI. EVALUATION METRICS

Implementation of Haversine Formula:
Haversine formula is an important equation to find the navigation, it gives shortest distance between two points on a sphere based on latitude and longitude. Importing math library under which sin, cos, sqrt, asin, radian methods are imported. After importing function is defined under which, co-ordinates of pickup latitude, pickup longitude and dropoff latitude, dropoff longitude, are stored in different memory indices. Converting units of latitude and longitude from degrees to radians.

```
from math import radians, cos, sin, asin, sqrt
def haversine(a):
    longitute1=a[0]
    latitude1=a[1]
    longitute2=a[2]
    latitude2=a[3]
```

```
# Convert decimal degrees to radians
longitute1, latitude1, longitute2, latitude2 = map(radians, [longitute1, latitude1, longitute2, latitude2])
```

Applying the haversine formula by finding the distance of longitude and latitudes. This is done by taking the difference between the end point and start point of longitude, similarly

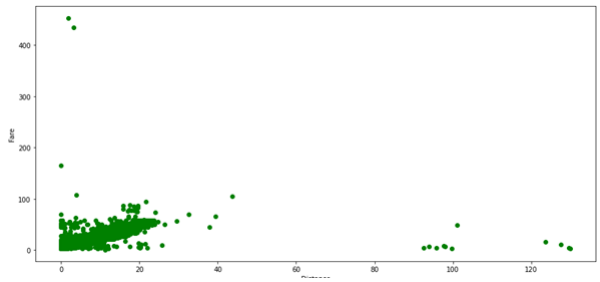
for latitude.
The expression for haversine formula is d = 2r sin-1 $\sin^2$(2-1/2) + cos 1 cos 2 $\sin^2$(2-1/2)

```
# haversine formula
dlon = longitute2 - longitute1
dlat = latitude2 - latitude1
a = sin(dlat/2)**2 + cos(latitude1) * cos(latitude2) * sin(dlon/2)**2
c =  2 * asin(sqrt(a))
```

In the end, unit is converted to kilometre (km).

VII. Result and Conclusion

Graph representation to define the relationship between the fare and distance which is calculated using haversine formuale.



The dataset is compatible with the applied model, random forest, linear regression and decision tree, using the entries, R squared values were determined for both test and training dataset. The resulted value for both the datasets are below 1 for random forest, linear regression, decision tree. Mean squared error and Mean absolute error are calculated to evaluate the performance on the dataset before cleaned and processed.
Mean Squared Error MSE
Mean Absolute Error MAE
R square for both test and train dataset
R squared value is calculated after finding the mean squared error and mean absolute error for both the datasets.

Random Forest:

```
Mean Squared Error for Train data = 7.871095528292946
Mean Absolute Error for Train data = 0.8993859415944033
Mean Squared Error for Test data = 15.45519172929083
Mean Absolute Error for Test data = 2.1504509130199883
```

R value for train data:

```
0.9363303958773824
```

R value for test data:

```
0.7939766236273516
```

Decision Tree:

```
Mean Squared Error for Train data = 57.78346570031226
Mean Absolute Error for Train data = 3.0971106392705208
Mean Squared Error for Test data = 20.288729313175867
Mean Absolute Error for Test data = 2.875461603855385
```

R value for train data:

```
0.5325872525943499
```

R value for test data:

```
0.7295437941743983
```

Linear Regression:

```
Root Mean Squared Error for Train data = 8.99100049041213
Root Mean Squared Error for Test data = 5.332146690728364
```

R value for train Data:

```
0.34609748309146127
```

R value for test Data:

```
0.6209938297138393
```

Gradient Boosting Regressor:
After using gradient boosting and finding the Mean Squared Value and R value for the train and test dataset.

```
Root Mean Squared Error for Train data = 5.885370347600376
Root Mean Squared Error for Test data = 3.6548238255428775
```

R value for train Data:

```
0.7198152072614653
```

R value for test Data:

```
0.8219364621305532
```

Hyperparameter Tuning:

Random Forest using Cross Validation:
After implementing hyperparameter tuning and determining the parameter to be used, Random Search hyperparameter Grid is created to calculate the R value and Mean squared Value.
R value for train Data:

```
# R_square

RRF_r2 = r2_score(y_test, predictions_RRF)
```

```
# Calculating RMSE

RRF_rmse = np.sqrt(mean_squared_error(y_test,predictions_RRF))
```

R value for test Data:

```
Random Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
R-squared = 0.73.
RMSE =  4.469299222774914
```

Applying the optimizing algorithm Random Search CV on Gradient boosting model and calculating the R value and Mean Squared value.

R value for test Data:

```
Random Search CV Gradient Boosting Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
R-squared = 0.78.
RMSE =  4.069712487003473
```

Grid Search using Cross Validation with 5-Fold CV:
R value for test Data:

```
gridcv_rf = GridSearchCV(regr, param_grid = grid_search, cv = 5)

gridcv_rf = gridcv_rf.fit(x_train,y_train)

view_best_params_GRF = gridcv_rf.best_params_
```

R value for test Data:

```
Grid Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'max_depth': 5, 'n_estimators': 12}
R-squared = 0.8.
RMSE =  3.8294149900194943
```

Grid Search CV for Gradient Boosting:
The R value and Mean Squared Value were determined using Random Forest CV on gradient boosting, now applying it using Grid Search CV with 5-Fold CV to determine the error.

```
gridcv_gb = GridSearchCV(gb, param_grid = grid_search, cv = 5)

gridcv_gb = gridcv_gb.fit(x_train,y_train)
view_best_params_Ggb = gridcv_gb.best_params_
```

```
Grid Search CV Gradient Boosting regression Model Performance:
Best Parameters =  {'max_depth': 7, 'n_estimators': 19}
R-squared = 0.81.
RMSE =  3.7449268310103383
```

Fare Price using Grid Search CV.
Comparing all the value shows that Grid Search is more opted as the value shows minimal error compared to Random Search CV on gradient boosting, hence choosing Grid Search optimizing on Random Forest Model using the test dataset to find the fare price for the calculated distance.

```
# Grid Search CV for random Forest model

regr = RandomForestRegressor(random_state = 0)
n_estimator = list(range(11,20,1))
depth = list(range(5,15,2))
```

The data is already been cleaned and processed, hence applying grid search cv on the random forest model.

```
#Apply model on test data

predictions_GRF_test = gridcv_rf.predict(dataset_test)

predictions_GRF_test
```

```
array([ 8.96532204,  9.20613259,  5.4861336 , ..., 52.92558951,
       25.75879193,  5.82456515])
```

Final output for the test data after calculating the fare price using Grid Search

	passenger_count	Year	Month	Date	Day	Hour	distance	Predicted_fare
0	1	2015	1	27	1	13	2.323259	8.965322
1	1	2015	1	27	1	13	2.425353	9.206133
2	1	2011	10	8	5	11	0.618628	5.486134
3	1	2012	12	1	5	21	1.961033	9.180767
4	1	2012	12	1	5	21	5.387301	14.064193

VIII. Future work

Finding optimal path for the given latitude and longitude can be done using Ant Colony optimization, it is specialized in solving the optimization problems , after each iteration the density of pheromone is vaporized. It represents the behavioural pattern of ants searching for food. This approach is carried out by a single ant wandering out for food and once it finds the source it traces back to its colony by leaving pheromone as a marker, using the marker the colony reaches the source. This marker is the path from the source to destination. It is an optimization technique for path finding algorithms, it provides increased accuracy i.e., it locates the exact position of the destination and calculates the total distance, which is the shortest path, hence it is more effective than haversine formulae.

IX. Important Links

Data set link https://www.kaggle.com/pankajkumar90/cab-fare-dataset
Github link https://github.com/TheAlgorithms

References

[1] “Cecep Nurul Alam, Khaerul Manaf, Aldy Rialdy Atmadja, Digital Khrisna Aurum. Implementation of Haversine Formula for counting event visitor in radius based on Android Application. Informatics Engineering Dept Faculty of Science and Technology State Islamic University Sunan Gunung Jati Bandung].

[2] “2. Chopde, P.N.R., and Nichat, M.K. 2013. Landmark Based Shortest Path Detection by Using A* and Haversine Formula. Department of Computer Science and Engineering, G.H. Raisoni College of Engineering and Management, Volume 1 Issue 2.].

[3] “3. Yash Indulkar, Abhijit Patil. Sentiment Analysis of Uber and Ola using Deep Learning. Information Technology Thakur College of Science Commerce.

[4] “4. Anastasios Noulas, Vsevolod Salnikov, Desislava Hristova, Cecilia Mascolo and Renaud Lambiotte. 2018. Developing and Deploying a Taxi Price Comparison Mobile App in the Wild: Insights and Challenges. Center for Data Science, New York University.

[5] [5] A. Marazzi, “5. E. Alpaydin, Introduction to Machine Learning, 2nd ed. The MIT Press, 2010.

[6] [6] “6. Kai Zhao1, Denis Khryashchev, Juliana Freire,Cl´audio Silva and Huy. Predicting Taxi Demand at High Spatial Resolution: Approaching the Limit of Predictability. Center for Urban Science and Progress, New York University 2Department of Computer Science and Engineering, New York University.

[7] https://towardsdatascience.com/introduction-to-machine-learningalgorithms-linear-regression-14c4e325882a

[8] Alper Sarikaya, Michael Correll, Lyn Bartram, Melanie Tory, and Danyel Fisher. (2018). What Do We Talk About When We Talk About Dashboards?. IEEE InfoVis 2018 (Berlin, Oct 21-26, 2018)

[9] https://www.decisionanalyst.com/media/downloads/ConsumerDecisionMaking.pdf

[10] Sengottuvelan P Gopalakrishnan T (2016). A hybrid PSO with Naïve Bayes classifier for disengagement detection in online learning, Program, ISSN: 0033-0337, vol. 50, no. 2, pp. 215-224

[11] https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/