Improving Taxi Revenue using Reinforcement Learning

Shahil Subham
Dept. of ECE
R. N. S. Institute of Technology
Bangalore, India

Saurabh Singh
Dept. of ECE
R. N. S Institute of Technology
Bangalore, India

Anusha Sunil Kumar
Dept. of ECE
R. N. S. Institute of Technology
Bangalore, India

Farheen Fatima
Dept. of ECE
R. N. Shetty Institute of Technology
Bangalore, India

Geetha G
Assistant Professor
Dept. of ECE
R. N. Shetty Institute of Technology
Bangalore, India

***Abstract*—Several advancements has been made recently in the field of transportation system and taxis being a major part of urban transportation has seen a tremendous growth in the past few years through the use of online cab services such as Uber, Ola etc. Relying on these advances these aggregation systems such as Uber, Ola etc. was able to activate more cabs and thereby improved customer experience by increasing availability and by reducing wait times. A lot of studies were made in the customer's perspective to give improved customer experience. But in this paper, the only focus is on improving performance from a driver's perspective by using current and past movement trajectories and trips and thereby maximizing the long-term revenue earned by the driver.**

Keywords—Reinforcement Learning, Taxi Revenue, XG boost, Regression, Random forest, K-neighbor, Gradient boosting.

I. INTRODUCTION

Generally, when there is no customer on board taxis roam around with no clear destination and this is referred to as "cruising". Such a cruising taxi may find customers either on streets or due to being in proximity to customers who put in a request to taxi aggregation systems such as Uber, Ola etc. In such cases, it is necessary for the taxi to be in the right location at the right time to increase revenue by reducing cruising time. In this paper we focus on developing a Reinforcement Learning approach with the aim of maximizing long-term revenue by providing guidance to cruising taxi drivers to be on the right locations to be at different times of the day on different days of the week. Reinforcement learning is an ideal approach for this problem due to the following reasons: (a) In order to find customers during cruising we need to make a sequence of decisions, say for example, the driver can wait in current zone for 5 minutes, and if no customer found, the driver can wait for 5 more minutes and if the driver still fails to find a customer then move to another zone; (b) Reinforcements are well-defined, i.e., cost from traveling between locations and the sum of revenue earned from a customer; (c) Customer demand is uncertain and RL approaches can capture such uncertainty quite well; and finally (d) Because of its learning focus, RL can adapt to any changes in demand patterns.

II. DATA

A. *Data Collection and Exploration*

Data collection is the process of gathering and measuring information on targeted variables. The dataset used in this paper consist of 50000 rows and seven column attributes containing information as longitude and latitude of pickup and drop off locations, passenger count ,pickup date and time etc. Training data has one more attribute for fare amount. Data exploring being the first step in data analysis refers to view the dataset in an organized way.

B. *Data Pre-processing*

This is the most important. part in the machine learning workflow. It is the process of preparing the raw data and transforming the raw data into an understandable format. The algorithm is completely dependent on how the data is fed into it. Therefore feature engineering being an integrated step in data pre-processing should be given top priority for every project. The advantages of the pre-processing of the data is that it reduces Over-fitting i.e. redundancy of the data can be reduced by which opportunity to make decisions based on noise is minimized and fewer misleading data results in improving modeling accuracy. Another advantage of data pre-processing is that it reduces training time i.e. fewer data points reduce algorithm complexity and algorithms train faster. In this paper the dataset is divided into training set used for training the model and test set used to test the trained model. 70% of the dataset is given as training set and the remaining 30% of the dataset forms the test set. The flowchart in Fig 1 gives the basic steps followed in the paper.

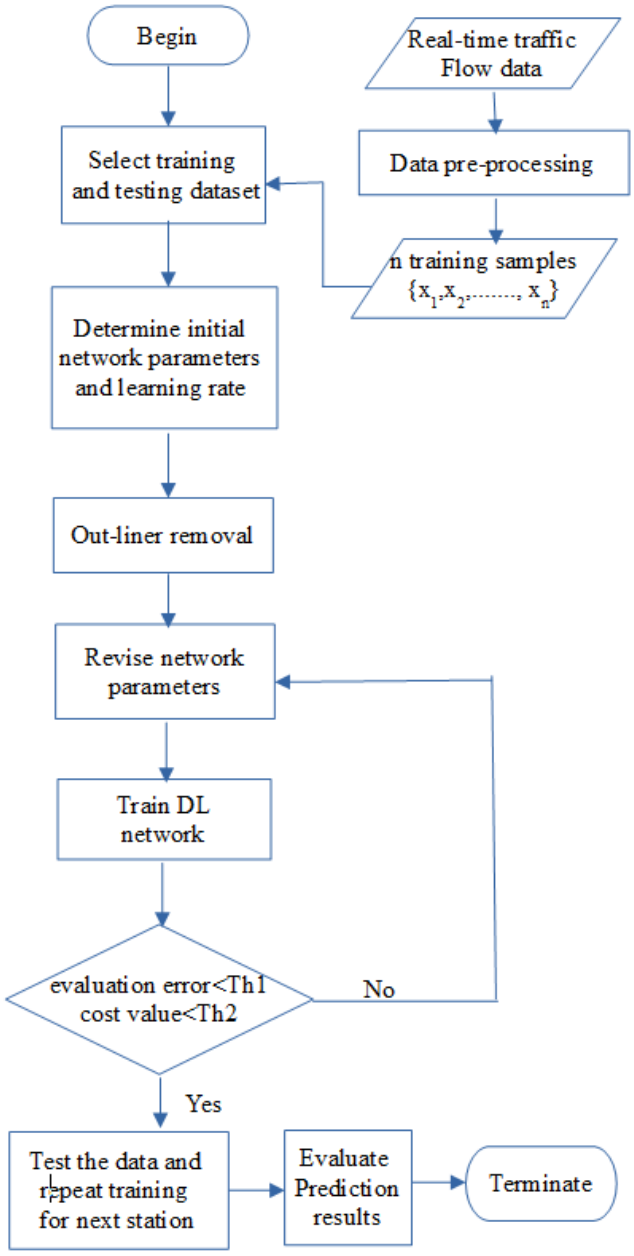

Fig 1: Flowchart showing the basic steps

III. TAXI DATASET

Datasets are an integral part in the field of machine learning. The dataset used in this paper is a New-York taxi dataset which is obtained from Kaggle. The taxi dataset used in this paper has seven attributes namely key, pickup_date, pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude and passenger_count. The training set also have a dependent attribute named fare_amount.

A. *Defining boundaries*

Since we need to work in a finite environment it is important to define a boundary. For this a boundary box is defined with the help of the minimum longitude and latitude, and the maximum longitude and latitude. Now any location data point with the corresponding longitude and latitude values below the minimum value or above the maximum value can be removed from the dataset since it is out of scope. We will be considering only those points that lie inside the defined boundary box.

B. *Preparing dataset for model training*

- In the dataset, there can be some rows where some of the attribute values can be missing. Say for example, the dropoff_longitude value is absent in some row. In that case the model would not be able to give accurate results. Hence it is important to remove all the missing data from the dataset in order to give accurate predictions.
- In the dataset there are longitude and latitudes that corresponds to locations in water bodies. These data points are considered as noise as it make no sense if the drop off or pickup locations are in water because taxi services cant be made available in water bodies. Hence it is necessary to remove all the data points that lie in water bodies.

C. *Data analysis*

- To further analyze the dataset we tried to visualize traffic density by the hour (and year). Counting the number of pickups in an area will give us some impression of the traffic density. If the traffic is more then there are chances that it would take a lot of time for the driver to make a ride. Visualization of the traffic density by the hour (and year) for two days (Monday and Friday) for the year 2014 is given in the Fig 2 and Fig 3.

- Before building any model it is important to test some basic intuition. Visualization of distance time relation and distance fare relation has to be done. It can be observed that if distance between pickup and drop-off location is longer, the fare amount will also be high. It is to be noted that fare at night is different from fare at day time. Also for some trips, like to/from an airport, have fixed fee regardless of the pickup or drop off location.

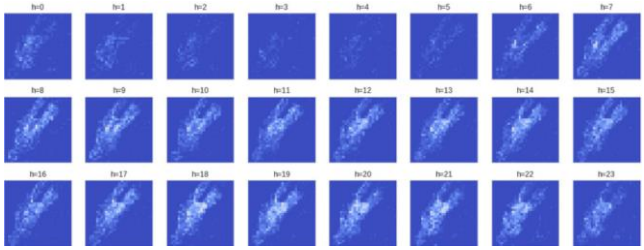

Fig 2: Pick up density, year=2014,day= Monday

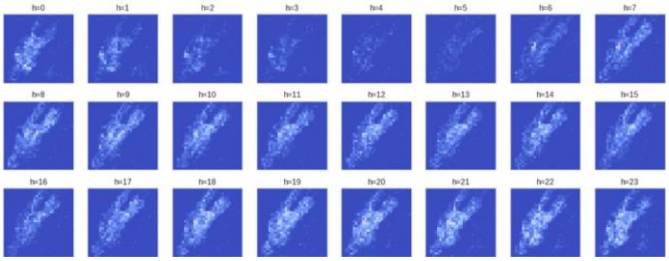

Fig 3: Pick up density, year=2014,day= Friday

IV. EVALUATING VARIOUS MODELS

Different models will be generated by using different techniques-Linear Regression, Polynomial Regression, K-neighbor regression, Random Forest, Gradient Boosting and Extreme Gradient (XG) boost to generate the fare amount. The generated models will be then evaluated and analyzed on the basis of its RMSR values and the best among them will be selected for the resourceful implementation of this project. Lesser the RMSR value better will be the efficiency of our model.

A. *Linear Regression*

This is a machine learning algorithm which is based on supervised learning. This regression performs the task to predict a dependent variable value based on given independent variable. So this regression technique finds out a linear relationship between input and output [9]. The following steps have been used for generating linear regression model:

1) Import package and class: The first step is to import the package numpy and class Pipeline from sklearn.pipeline, class, LinearRegression from sklearn.linear_model and the class StandardScaler from sklearn.preprocessing. Now we have all the functionalities that we need to implement linear regression model.

2) Provide data and create a model: The data is then provided and eventually the appropriate transformations are done. Then the regression model is created by using existing data.

3) Checking the results and predicting the response: The model is then fitted. After this we will get the results to check whether the model works satisfactorily. Once there is a satisfactory model, it will be then used for predictions with either existing or new data.

B. *Polynomial Regression*

This is a form of regression analysis in which relationship between the independent variable x and the dependent variable y is modeled as a n^{th} degree polynomial. Some polynomial terms will be added to the Multiple Linear Regression equation to convert it into Polynomial regression.

This is a linear model with some modification in order to increase the accuracy. The dataset used in Polynomial regression for training is of non-linear in nature. It makes use of Linear Regression model to fit the complicated and non-linear functions and dataset. The original features are converted into polynomial features of degree 2 and is then modeled using a linear model. The main steps involved in polynomial regression are given below:

1) Data pre-processing and building a Polynomial regression model: In data pre-processing the data gets encoded so that it can be brought to such a state that now the machine can easily parse it. The features of data can now be easily interpreted by the algorithm. Then the linear regression model is built and fitted to the dataset. In building polynomial regression model, linear regression model is taken as reference. Once the polynomial regression model is built, it will be different from the simple linear model. Here we are using PolynomialFeatures class of pre-processing libraries.

2) Visualizing the result and predicting the output: The result of polynomial regression model is then visualized and then the final result with the polynomial regression model is then predicted and compared with Linear Model.

C. *K- Neighbor Regression*

K Nearest Neighbor is an algorithm that stores all available cases and predict the numerical target based on a similarity measures. The main steps involved in K-Neighbor Regression are given below:

1) Importing and splitting the data: First the class KNeighbourRegressor is imported from sklearn.neighbours. Then the dataset is imported and split into training and testing set.

2) Calculating the Euclidean distance: After finding the k parameter which is the number of Neighbors we calculate the distance between the query instances and all the training samples. Based on the k-th minimum distance the nearest neighbours are determined.

3) Prediction of the query instance: This is done by simply taking the simple majority of the category of the nearest neighbours.

D. *Random Forest*

This is a tree based algorithm which involves building several decision trees, then their output is combined to improve generalization ability of the model. Here a combination of weak learners(individual decision trees) produces a strong learner. In simple words you can say a random forest is a collection of several decision trees. A random forest works the following way:

1) First it uses the bagging algorithm to create random samples. Given a dataset (n rows and p columns),it creates new dataset by sampling cases at random with replacement from the original data. About one third of the rows of the given dataset are left out, known as Out Of Bag samples (OOB).

2) Then the model trains on the new dataset. OOB samples are used to determine unbiased estimate of the error. Out of p columns, P<<p columns are selected at each node in the dataset. The P columns are selected at random. The default choice of P is p/3 for the regression tree.

3) Here unlike a decision tree, no pruning takes place in random forest i.e. each tree is fully grown. In pruning a subtree is selected that leads to the lowest test error rate. Cross validation is used to determine the test error rate of a subtree.

4) Several trees are grown and the final prediction is obtained by averaging or voting.

Here each tree is grown on a different sample of original data. The main advantage of random forest is that it is robust to correlated predictors and takes care of missing data internally in an effective manner.

E. *Gradient Boosting*

This is a machine learning technique for regression and classification, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It is a boosting technique. Boosting is an ensemble technique of converting weak learners into strong learners. Similar to random forest, gradient boosting is a set of

decision trees. By using gradient descent and updating our predictions based on a learning rate, we can find the values where MSE(Mean Square Error) is minimum. Gradient boosting trains the model gradual, additive and sequential manner. It basically involves three elements which are discussed below:

1) A loss function to be optimized: The loss function depends on the type of problem being solved. A benefit of the gradient boosting framework is that a new boosting algorithm does not have to be derived for each loss function, instead it is generic enough framework that any differentiable loss function can be used.

2) A weak learner to make prediction: Decision trees are used as the weak learner in gradient boosting. We can say the gradient boosting technique reduces error sequentially. Trees are constructed in a greedy manner in such a way that the loss will be as minimum as possible. The weak learner is constrained in specific ways, such as maximum number of layers, nodes, splits or leaf nodes to ensure that the learners remain weak, but can still be constructed in a greedy manner.

3) An additive model to add weak learners to minimize the loss function: Trees are added one at a time, and existing trees in a model are not changed. A gradient descent procedure is used to minimize the loss when adding trees.

F. XG Boost

Extreme gradient boosting is similar to gradient boosting framework but more efficient. It has both linear model solver and tree learning algorithm. Its capacity to do parallel computation on a single machine makes it very fast. It is 10 times faster than existing gradient boosting implementations. It supports various objective functions, including regression, classification and ranking.

XG boost only works with numeric vectors. Therefore all other forms of data are first converted into numeric vectors. One Hot Encoding is a method which is used to convert categorical variable into numeric vector. This step will make a sparse matrix using flags on every possible value of that variable. Sparse Matrix is a Matrix where most of its values are zeroes. Conversely dense matrix is a matrix where most of the values are non zeroes.

Fig 4 shows the different RMSR (Root Mean Square Rate) values of the models generated by different techniques. RMSR value shows how much percentage factual output which is predicted by the model varies with the ideal output.

```
rmsr - model (nfolds=10)
========================================
4.2531 - xgboost100
4.3046 - random_forest_regressor_n100
4.3595 - random_forest_regressor_n10
4.4630 - gradient_boosting_n100
4.7004 - kneighbors
4.8814 - gradient_boosting_n10
5.1222 - polynomial
5.7785 - linear_model
6.8268 - xgboost10
```

Fig 4: Different RMSR values of the models.

Lower values of RMSR indicates better fit. So, among all the models generated by different techniques, XG boost is best suited for the resourceful implementation of this project.

Fig 5 shows the accuracy of the model when XG boost is implemented. At the x axis we have the actual output and at the y axis we have the obtained output. The red line is the output of our agent and the blue dots indicates the actual output values. The Root Mean Square Error(RMSE) value obtained when XG boost is applied to our model is 3.75. This value gives the difference between actual output and the predicted output.

The Fig 6 is a histogram that gives an idea about the accuracy of the model when XG boost is applied. The x axis is the difference between actual output and the predicted output and y axis shows the performance of the model. It is understood from the histogram when the difference between the actual and the predicted output is zero, the system performs with maximum accuracy. This is the ideal condition. As the difference between the actual and predicted output become non-zero values then we can say the agent is operating in a more practical condition.

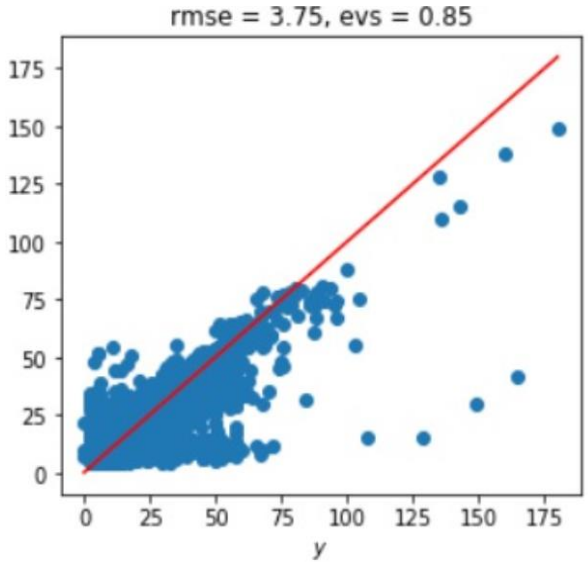

Fig 5: Accuracy of the model when XG boost is used

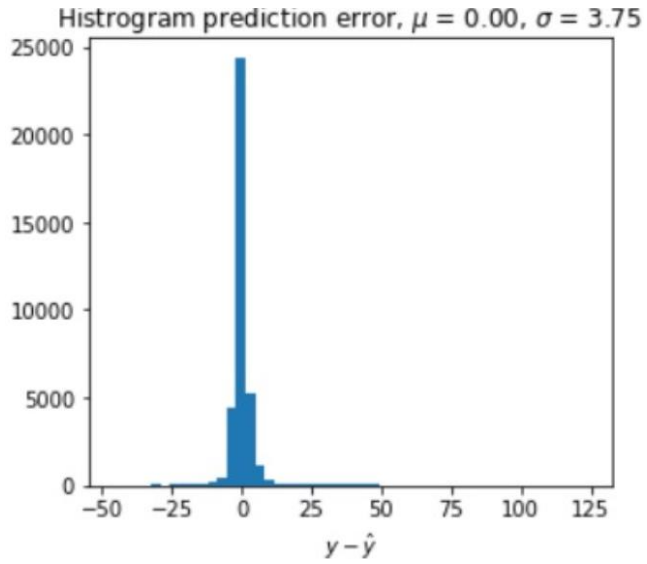

Fig 6: Histogram prediction error

Type	Linear	Polynomial	KNN	Gradient	RF	XGB
Obtained RMSR	5.77	5.12	4.70	4.45	4.29	4.25

Table 1:RMSR values for various techniques

Model with the least Root Mean Square Rate (RMSR) value will be used to implement the agent discussed in the paper. On comparing it is observed that Extreme Gradient (XG) boost which is the more efficient version of Gradient boosting is best suited for this model since it has the least RMSR value. Table 1 shows the various RMSR values obtained for different techniques.

V. Introducing RL To The Model

Reinforcement learning is defined as the study of decision-making over time, that trains the algorithms using a system of reward and punishment. A reinforcement learning algorithm, or agent, interacts with its environment and the agent receives rewards if correct actions are performed and receive punishments if incorrect actions are performed. This way the agent learn on its own without any human intervention and tries to maximize the reward and reduce the penalty. In this paper we assign various rewards for different actions. Say for eg the agent receives a reward every time the revenue earned by the driver is higher than the amount spent by the driver during the time of the cruising, and receives a punishment every time the revenue earned by the driver is lesser than amount spent by the driver during the time of cruising, also if the revenue is more than 1000 our agent receives a reward, for every correct turn it gets a reward of 20 points and for every incorrect decision it gets a negative reward of -1 points etc.

Q learning is a model-free off policy reinforcement algorithm that comes up with a policy that tells the agent to take proper actions under various circumstances. Every action that an agent takes has a q value which is the expected discounted future reward. Now it is important for the agent to have a memory. This is where the Bellman equation comes to play. We use the Bellman equation to enable our agent with a memory.

$$V(s)=\max(R(s, a)+\gamma V(s')) \quad (1)$$

The equation (1) shown above is the original bellman equation. In order to join the probabilities of the actions that are taken in the above equation we need to associate a probability with each of the terms to quantify our agent, if it has any chance of taking a particular action. Then the above equation has to be modified as:

$$V(s)=\max(R(s, a) + \gamma \Sigma s' P(s, a, s')V(s')) \quad (2)$$

The equation (2) is the Bellman equation with Markov decision process. Now we have to transition to Q learning . We need to get an equation to quantify the quality of particular action.

$$Q(s, a)=(R(s, a) + \gamma \Sigma (P(s, a, s') \max Q(s',a')) \quad (3)$$

The equation (3) shown above is the Bellman equation with q values.

A. *Working of RL based model*

Let us consider a scenario where the taxi drivers cruising trajectory started from a point A and ended at a point E.

In the beginning, assume that the driver made the decision to go to a destination E at the point A itself. Without any conscious reasoning, if the taxi driver had made a decision to go to E at A, then the driver would have chosen a path such that the chosen path is close to the shortest path distance between the points A and E. In this case simply assume that, E is not close to the shortest path distance. So now it is important to identify the point on the cruising path which is close to the shortest path distance to E. Say this point is D. Then it is evaluated if the driver could make the decision to go to point D at A itself. If this decision cannot be made, then the point where the driver decided to go to point D is identified. Assume that point C is that point. This computation is repeated further until the final trajectory A, B, C, D, E is obtained as shown in Fig. 7 [4]

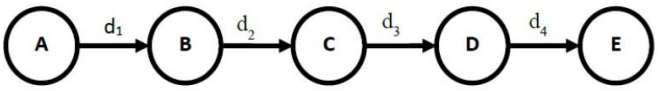

Fig 7: Activity diagram for cruising path

B. *Framework*

When a taxi driver drops off a customer, and is looking for a new customer, he can either choose to stay in the area and wait for a new passenger there, or he can travel to a new location. Traveling to a new location comes with an associated cost, but if the new location is chosen correctly it will impact future trips to improve the revenue for the day. We can define the states, reward and value functions as:

.

- States: (z, t) a drop off zone at a corresponding time
- Reward: R(z1, t1) average trip fair of one trip for a pickup zone z1 at time t1
- Value function: V (z, t) gives the expected total revenue to the end of the day starting from state (s, t).

In a traditional reinforcement learning model the intention of the driver would be an important feature. In practice, although a driver may intend to drive to a particular location, there are chances that he may find a customer along the way. However, it is not possible to know a driver's intention from historical data because we only know where he ends up picking the next customer. So in order to reduce the difficulty in estimating the model parameters from historical data, we consider a simplified policy:

Policy: $\pi(z, t)=z_1$ *the next pickup zip-code from state (z,t).The driver goes to* z_1 *directly and will keep searching inside* z_1 *until he picks up the next customer at* z_1.

Then, we can write down the value function as:

$$V^{\pi}(z, t) = E[f(\pi(z, t), Z', T_{pick}) + V^{\pi}(Z', T_{next})] = E[\sum(P(z_1, z', T_{pick}) f(z_1, z', T_{pick}) + V^{\pi}(z', T_{next}))] = E[R(z_1, T_{pick}) + \sum P(z_1, z', T_{pick}) V^{\pi}(z', T_{next})] \quad (4)$$

The equation (4) shows the value function where Z', T_{pick}, T_{next} represents the random zip-code that the customer at z_1 and the random drop-off time at Z' respectively. $f(z_1, z', T_{pick})$ is the trip fare from z_1 to z' at the time t and $P(z_1, z', T_{pick})$ is a customers transition probability from z_1 to z'.The above equation hold after assuming that all the random variables are independent. To further quantify T_{pick} and T_{next}, we have

$$T_{pick} = t+\Delta_{travel}(z,z_1)+\Delta_{search}(z_1) \quad (5)$$

$$T_{next}=T_{pick}+\Delta_{trip}(z_1,z') \quad (6)$$

From equation (5) and equation (6) it can be seen that T_{pick} and T_{next} are further quantified. Δ_{travel}, Δ_{search} and Δ_{trip} are the random time intervals for the time cost traveling from z to z_1 without a passenger, searching for the next customer at z_1, and driving passenger from z_1 to z' respectively. Finally, we replace all the random time with their expectations. As the value function is almost linear in t and the estimates of $R(z_1,t_1)$ and $P(z_1,,z',t)$ will be piece wise constant in hour, this approximation should be accurate enough most of the time. The optimal value function can be defined as shown below in equation (7):

$$V^*(z, t)=\max V^{\pi}(z, t) \quad (7)$$

Then, it will satisfy the equation (8) shown below:

$$V^*(z, t)=\max\{R(z_1, t_{pick})+\sum P(z_1,z', t_{pick})V^*(z', t_{next})\} \quad (8)$$

Where

$$t_{pick}=t+\delta_{travel}(z, z_1, t)+\delta_{search}(z_1,t+\delta_{travel}(z, z_1,t))$$

$$t_{next}=t_{pick}+\delta_{trip}(z_1,z',t_{pick})$$

C. Agent class

If this framework is used, then it is needed to fill the following:

can define the states, reward and value functions as:

1) State and Action Size.

2) Hyper-parameters.

3) Create a neural-network model in function 'build_model()'.

4) Define epsilon-greedy strategy in function 'get_action()'.

5) Complete the function 'append_sample()'. This function appends the recent experience tuple <state, action, reward, new-state> to the memory.

6) Complete the 'train_model()' function with following logic:

- If the memory size is greater than mini-batch size, then randomly sample experiences from memory as per the mini-batch size and do the following:
- Initialize the input and output batch for training the model.
- Calculate the target Q value for each sample: reward + gamma*max(Q(s', a,)).
- Get Q(s', a) values from the last trained model.
- Update the input batch as the encoded state and output batch as the Q-values.
- Then fit the DQN model using the updated input and output batch.

Q-network Architecture:

- Input: encoded state i.e. each vector is a combination of locations + hours in a day + days in a week.
- output: q-values for all actions including (0, 0).

Hyper-parameters

- state_size: vector length of encoded states (number of neurons in input layer): (36 -> 5 zones + 24 hours + 7 days).
- action_size: vector length of predicted q_values for all actions.
- learning_rate: amount that the weights are updated during training.
- discount_factor: affects how much weight is given to the future rewards in the value function.
- batch_size: batch size used in neural network for training
- memory_length: replay memory buffer size
- nn_epochs: number of epochs for neural network

D. DQN block

Episodes: For implementing reinforcement learning we need to convert activity graphs into episodes. Each node in the activity graph shown in Fig. 7 represents a state and the subsequent node represents the action taken. Activity graphs are converted into a series of state-action pairs with the help of zone structure of the map and spatio-temporal information present in each node. The last node of the activity graph is considered as terminal state of the episode. **B**y applying a fixed cost per km to the weight of the edge the cost of travel between nodes is determined. If the cruising trajectory ends with finding a passenger, a positive reward is awarded.

The equation (9) given below[4] represents an episode for activity graph shown in Fig. 7. Let S_x be the state and Z_x be the zone of node X, S_{term} is the terminal state.

$(S_a, Z_b) \rightarrow (S_b, Z_c) \rightarrow (S_c, Z_d) \rightarrow (S_d, Z_e) \rightarrow S_{term}$ (9)

The agent learns the Q values of state-action pairs from the episodes. Once the Q values are obtained, the Q table is updated with the new Q values and by referring to this Q table the action with the highest Q value for a given state is selected as the best action.

VI. RESULTS

On performing all these operation on a daily basis, the model trains itself and makes itself more efficient with every decision it makes. The model learns to predict the better place and route to be on at any given instant. This data can further be used by other taxi drivers or taxi company as reference so that they can implement it in their daily lives and thus a better functioning system is established for the cab drivers.

The Fig 8 and Fig 9 are the snapshot of the simulation obtained. Here R, G, B, Y are four different locations that can be either pickup or drop off locations and A represents the obstacles. The rectangular slab represents the location of the taxi at any given time. In Fig 8 our agent gets a negative reward of -1 points for taking a wrong action. In Fig 9 our agent was able to successfully drop the passenger to the right location and hence a positive reward of 1000 points was assigned to our agent.

Fig 8: When negative reward is received

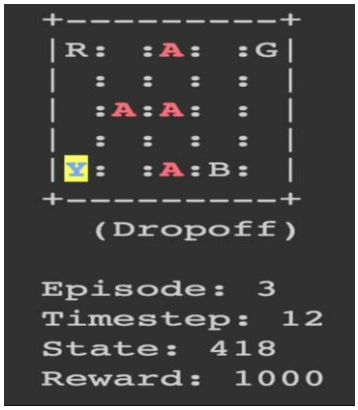

Fig 9: When positive reward is received

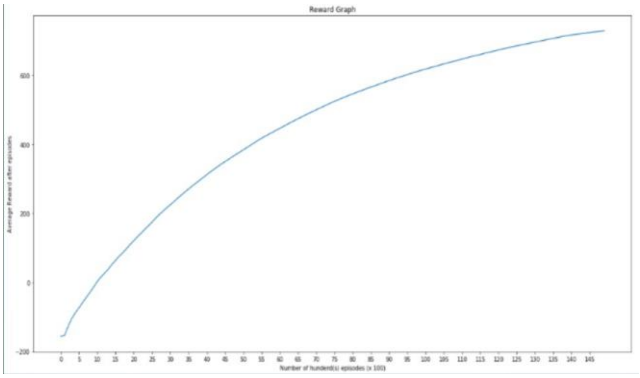

Fig 10: Reward graph

The graph shown in Fig 10 shows the exponential growth of the revenue of the driver as the data value and number of decisions made by the model increases. The x axis gives the number of hundreds of episodes and the y axis gives the average reward after episodes. From the graph we can see that initially when the number of episodes is zero or in other words when we first start to train our agent, the agent gets negative rewards. But as the episodes increases the agent starts to learn about its environment and as a result the average reward assigned to the agent increases. After training the agent for approximately around fifteen thousand episodes the average reward value attains a maximum value and remains constant after that. The accuracy of the agent discussed in this paper is 83.36%. The driver will have to download the app to keep the track of the customers. Developing an app suitable for this work is kept as future work.

VII. CONCLUSION

The paper talks about the enhancements that can be made in the field of taxi services so that it is user friendly and make more accurate predictions. In this paper, an RL agent, with no knowledge of the environment or taxi demand scenario, is capable of obtaining revenue which is comparable to (and in some cases higher than) revenue earned by top 10 percentile of drivers.

REFERENCES

[1] Samuel Daulton, Sethu Rmamn, Tijl Kindt, "NYC taxi data prediction
[2] Aishwarya Ramachandran "Machine Learning to predict taxi fare -Part one: Exporatory Analysis" 18 Aug 2018.
[3] Aishwarya Ramachandran "Machine Learning to predict taxi fare -Part two: Predictive modeling" 22 Sept 2018.
[4] Tanvi Verma, Pradeep Varakantham, Sarit Kraus, Hoong Chuin Lau "Augmenting Decision of Taxi Drivers through reinforcement Learning For Improving Revenue" 2017.
[5] Whong, Chris. "FOILing NYCs Taxi Trip Data. N.p., 18 Mar. 2014. Web. 11 Dec. 2014..
[6] Andre, D., and Russell, S. J. 2002. State abstraction for programmable reinforcement learning agents. In AAAI/IAAI, 119–125.
[7] Reuters. 2016. Uber debuts self-driving vehicles in landmark pittsburgh trial. Reuters, 14 September 2016. Available: http://www.reuters.com/article/us-uberautonomous- idUSKCN11K12Y [Last accessed: November 2016].
[8] Straitstimes. 2016. World's first driverless taxi trial kicks off in singapore. The Straits Times, 26 August 2016. Available: http://www.straitstimes.com/singapore/transport/worldsfirst- driverless-taxi-trial-kicks-off-in-singapore [Last accessed: November 2016].
[9] Parameshachari B D et. al "Big Data Analytics on Weather Data: Predictive Analysis Using Multi Node Cluster Architecture", International Journal of Computer Applications (0975 – 8887) proceedings of National Conference on Electronics, Signals and Communication – 2017, pp 12-17,2017