Price analysis, Profitability and Affordability of Taxi trips in New York City

Name: Venkata Krishna Bharadwaj Boinepally

UB ID: 50419396

UB Email: vboinepa@buffalo.edu

Course: CSE 587: Data Intensive Computing

Term: Fall 2022

Project Name: [Phase 2] Prediction of Taxi Trip Fare for given distance and time

Data Processing

(Same as Phase 1. For more details, refer Phase 1)

Data Processing steps are being repeated here. These are the same steps as Phase 1. Since I am downloading the data again, I am including the Data Processing that is required. Scroll down to the "Machine Learning models to predict fare" section for the actual Phase 2 code.

Phase 2 begins

Machine Learning models to predict fare

(June 2022 Data, For Hire High Volume Vehicles only. Ex: Uber, Lyft)

```
def get_model_results(model_name, model):
    y_pred = model.predict(X_test)
    loss = mean_squared_error(y_test, y_pred)
    score = r2_score(y_test, y_pred)
    print(f'Test results for {model_name}')
    print(f'Test Loss: {loss}')
    print(f'Test R Square: {score}')
    return loss, score
```

```
X_train, X_test, y_train, y_test = train_test_split(df[['trip_miles', 'trip_time']],
df[['base_passenger_fare']], test_size=0.3)
```

The fare depends on the demand for the taxi at that particular day of the week and time. So deducing the day of the week from the date and passing it to the model can make it predict better

```
df['trip_day'] = df.request_datetime.dt.day_name()
df['trip_day'] = pd.factorize(df['trip_day'])[0]
```

Splitting the given data in the ratio 70:30 for training and testing

```
X_train, X_test, y_train, y_test = train_test_split(
    df[['trip_miles', 'trip_time', 'request_hour', 'pickup_hour', 'dropoff_hour',
'PULocation', 'DOLocation', 'taxi_company', 'trip_day']],
    df[['base_passenger_fare']],
    test_size=0.3
)
```

Model 1 - Linear Regression

- Linear Regression generally fits most of the data with good results (i.e., less loss).
- This will be a good starting point for doing price prediction.
- Based on the results we get for Linear Regression, we can improve the model for the next models.
- Normalizing the data before doing linear regression and enforcing positive coefficients (because fare is positive) gave better results than leaving everything as default
- We can now further improve this by using other advanced machine learning models like Decision Tree, Deep Learning

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression(normalize=True, positive=True, n_jobs=-1)
lr.fit(X_train, y_train)
lr_loss, lr_score = get_model_results("Linear Regression", lr)
```

Model 2 - K Nearest Neighbours

- K Nearest Neighbors takes the points closer to the given point and predicts fare based on the K closest points to the given input
- This should make the predicted price to be similar to other trips with similar trip time and trip fare
- KNN is also not effected by data which is not close to the given input which should give this model a better chance to be not effected by far away points which can result in drastic changes because of single outlier point
- Using k=10 and weighting points closer to the input gives good results
- This is an improvement over Linear Regression

```
from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor(n_neighbors=10, weights='distance', n_jobs=-1)
knn.fit(X_train, y_train)
knn_loss, knn_score = get_model_results("k Nearest Neighbours", knn)
```

Model 3 - Decision Tree Regression

- Decision Trees are used when we can group the data that are close to each other through similar feature set.
- Using Decision Tree will make sure the predicted price is consistent with the similar data points already present in training set

- For the purpose of this taxi fare data, I am choosing 500 max leaf nodes so that there will be 500 leafs for all price points.
- This configuration gives the best possible loss for the Decision Tree Regression
- In the 3 models we have seen so far, this Decision Tree model gave best results
- This gave better results than KNN

```
from sklearn.tree import DecisionTreeRegressor
dtree = DecisionTreeRegressor(max_leaf_nodes=500)
dtree.fit(X_train, y_train)
dtree_loss, dtree_score = get_model_results("Decision Tree Regressor", dtree)
```

Model 4 - LightGBM Regression

- LightGBM is a lighter Gradient boosting framework based on Decision Trees and is known for being fast and efficient.
- Instead of growing horizontally, this algorithm grows vertically and chooses leaf with minimum error which drastically improves prediction but this is prone to overfitting for smaller data sets. Since we are using 17M records, chances are very less for this to happen.
- We are choosing "gbdt" boosting type for the algorithm as it is based on ensemble model
- Since we have lots of data, we can choose a smaller learning rate so that the tree learns accurately.
- This is the best result we achieved so far.

```
from lightgbm import LGBMRegressor
lgbm = LGBMRegressor(learning_rate=0.1, n_jobs=-1)
lgbm.fit(X_train, y_train)
lgbm_loss, lgbm_score = get_model_results("LightGBM Regressor", lgbm)
```

Note: SVM Regression is taking too long to train on 17M records so that is not included in this phase

Model 5 - Deep Learning Model

For training the Deep Learning Model, we are using all 17 Million records.

After trying various Neural Network Architectures, different hyperparameters, I have finalized the following Neural Network with best results. The other Neural Networks I have tried are not shown here.

`!pip install tensorflow-addons`

- Since we have lots of data, we need to have sufficiently large number of parameters to train before we can start to overfit.
- Hence choosing more deeper and more number of neurons per layer.
- I have tried 7 other architectures of Neural Networks and this configuration performed best. Other Neural Networks are not shown in this notebook.

```
dl = Sequential(
  [
      Input(shape=X_train.shape[1:]),
```

```
      BatchNormalization(),
      Dense(64, activation='relu', name='layer1'),
      Dense(128, activation='relu', name='layer2'),
      Dense(128, activation='relu', name='layer3'),
      Dense(512, activation='relu', name='layer4'),
      Dense(1024, activation='relu', name='layer5'),
      Dense(1024, activation='relu', name='layer6'),
      Dense(256, activation='relu', name='layer7'),
      Dense(128, activation='relu', name='layer8'),
      Dense(64, activation='relu', name='layer9'),
      Dense(16, activation='relu', name='layer10'),
      Dense(1, activation='relu', name='output')
  ]
)
```

- We have 1.9M parameters to train. Since we have 17M data points. We should be able to train them.
- 20% of the training data is used for validation

```
dl.compile(optimizer='adam', loss='mse', metrics=[RSquare()])
history = dl.fit(X_train, y_train, validation_split=0.2, epochs=10,
batch_size=2048)
```

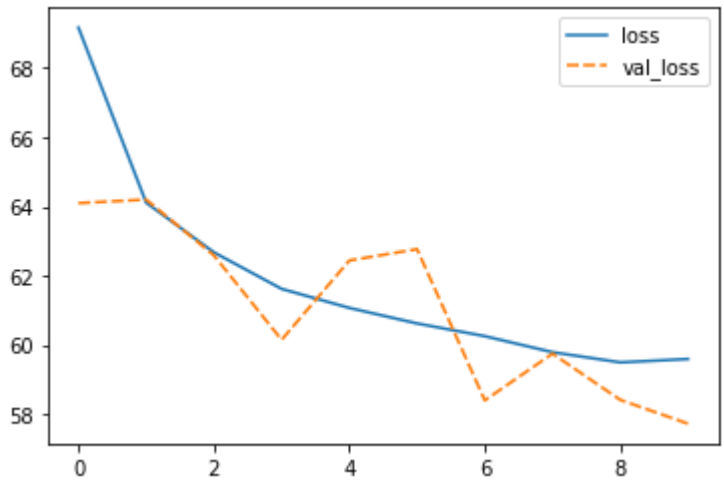
We can keep this training for some more epochs and go on till we see over fitting. With the addition of two new features "trip_day" and "request_hour", the model performed better with significantly improved results.

Results

Visualization - 1

We can plot the loss function over epochs for the Deep Learning Model and see it decreasing over epoch

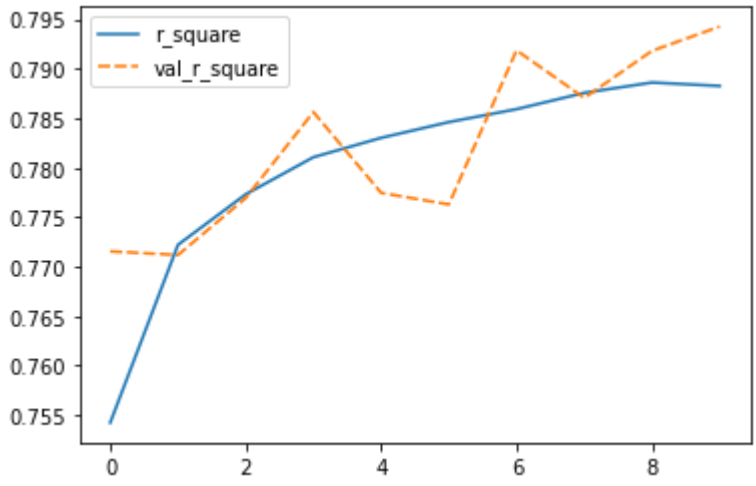
```
ax = sns.lineplot(data={k: v for k, v in history.history.items() if 'loss' in k})
ax.set_title('training and validation loss value over epochs')
```



Visualization - 2

We can plot the "R Square" of the train and validation over epochs and see it improving over time

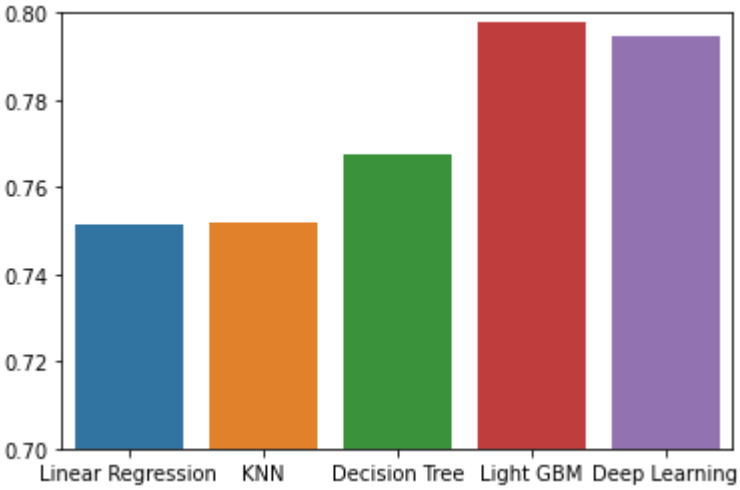
```
ax = sns.lineplot(data={k: v for k, v in history.history.items() if 'r_square' in k})
ax.set_title('training and validation R square value over epochs')
```



Visualization - 3

Comparing the R Square values of all the Machine Learning models and deep learning model

```
ax = sns.barplot(x=list(all_models_score.keys()), y=list(all_models_score.values()))
ax.set_ybound(0.7, 0.8)
ax.set_title('R square for various ML models')
```

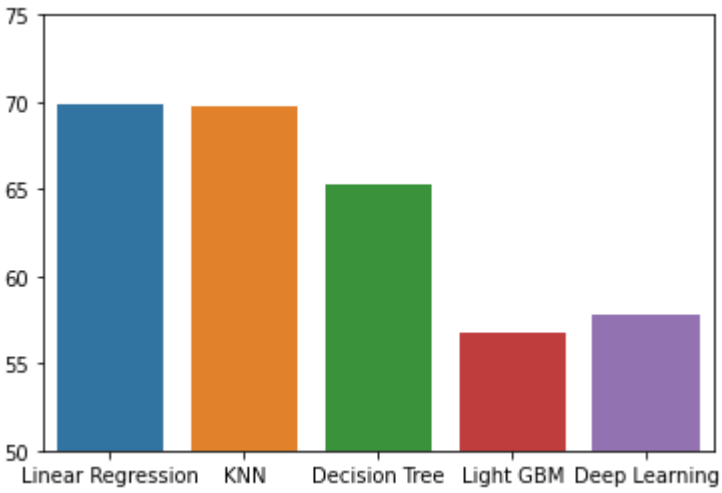


Top 2 performing models are Light GBM and Deep Learning model. These will be used in the Phase 3 for building data product.

Visualization - 4

Comparing the loss values of all the Machine Learning models and deep learning model

```
ax = sns.barplot(x=list(all_models_loss.keys()), y=list(all_models_loss.values()))
ax.set_ybound(50, 75)
ax.set_title('Loss for various ML models')
```



Again here too, Light GBM and Deep Learning models have least losses

Conclusion

- In this notebook, Machine Learning algorithms are applied to predict the pricing
- Light GBM Regression gave best results among the various Machine Learning algorithms

- Seven deep learning models are applied to the problem resulting in even better results.
- Only one best performing deep learning model is shown in this notebook among all the ones that are tried.
- We will be using the LightGBM Regression model and the Deep Learning model for inference in Phase 3
- We can potentially extend this model by adding historical trip data from different months. But this is not the scope of this project.

References

- https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page
- https://registry.opendata.aws/nyc-tlc-trip-records-pds/
- Seaborn, Pandas, Numpy documentation
- https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
- https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw
- http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-9.html
- https://hagan.okstate.edu/NNDesign.pdf#page=469
- https://datamahadev.com/understanding-light-gradient-boosting-machine/
- https://neptune.ai/blog/lightgbm-parameters-guide
- LightGBM is inspired from the Kaggle competition - https://www.kaggle.com/competitions/new-york-city-taxi-fare-prediction/code