Forecasting Ride-Share Driver Demand Using Machine Learning

Nikhil Mylarusetty

Department of Applied Data Science, San Jose State University

Data 270: Data Analyst Process

Professor Dr. Eduardo Chan

May 15, 2023

MODEL DEVELOPMENT

4.1 Model Proposal

The proposed project is the NYC Ride-Share driver demand project, with the primary motivation of finding the driver demand using the input features day, month, year, day of the week, and pickup locations. For this Project, the data has been collected from the official NYC TLC website, which is the most reliable source where real-time data on NYC yellow taxis are provided. The initial data collected is for six months for overall 263 locations in New York City. Raw Dataset consisted of 19 columns and 2463931 rows, which is minutely data. Pre-processed the data by removing NULL values, eliminating outliers, data transformation, and feature engineering. After performing pre-processing, the final dataset for modeling consisted of 26456 rows and 8 columns. Here the data has been aggregated into daily data by grouping the data from minute to daily. During the pre-processing step, the target variable 'Number of pickups' has been created. These are the final data which are prepared for modeling.

Extreme Gradient Boosting(XGBoost)

The significant reason for choosing XGBoost algorithm is that in a paper by Chen and Guestrin (2016), it is mentioned that XGBoost is known for its scalability as it uses automatic parallel and distributed computation, where multiple processors are used for computation by dividing a large problem into smaller ones. This process makes the execution faster and speeds up the time during model training. This paper also mentions the ability of XGBoost in handling Complex and large datasets. This paper clearly explains the overview, and the architecture of gradient boosting algorithms and mentions their key features and the math behind them.

In a paper by Noorunnahar et al. (2023) it is stated that XGBoost is a robust gradient-boosting algorithm that is ideally adapted for time-series tasks as it can handle missing values and feature interactions and prevent overfitting with its built-in Regularization. Therefore, the use of the XGBoost algorithm for the NYC driver demand project is an ideal choice as it consists of time series data where the key component of forecasting the target is dependent on it.

In a paper by Preniqi et al. (2020) they compared ARIMA, LSTM, SVM and XGBoost algorithms for electricity price forecasting and stated that XGBoost is accepting multiple external factors very efficiently at a time as input during modeling. This is one of the justifications for selecting XGBoost, as the current Project is a multivariate time-series project requiring algorithms capable of predicting output from multiple inputs, such as XGBoost.

XGBoost also gives the feasibility to customize various parameters, which will help to enhance the model performance by giving accurate results. Furthermore, while training the model, there is a high chance of overfitting, so to overcome such situations choosing an algorithm that can handle overfitting will be beneficial. In a paper by Anggraeni et al. (2021), the author mentioned that XGBoost uses regularized model formalization, which helps it in controlling the overfitting and making the model better in performance. This paper also discusses the architecture, functionalities and the advantages of using XGBoost for time series data. All of these advantages of using XGBoost are a great support for the NYC taxi project as it consists of a vast dataset of 19 crores.

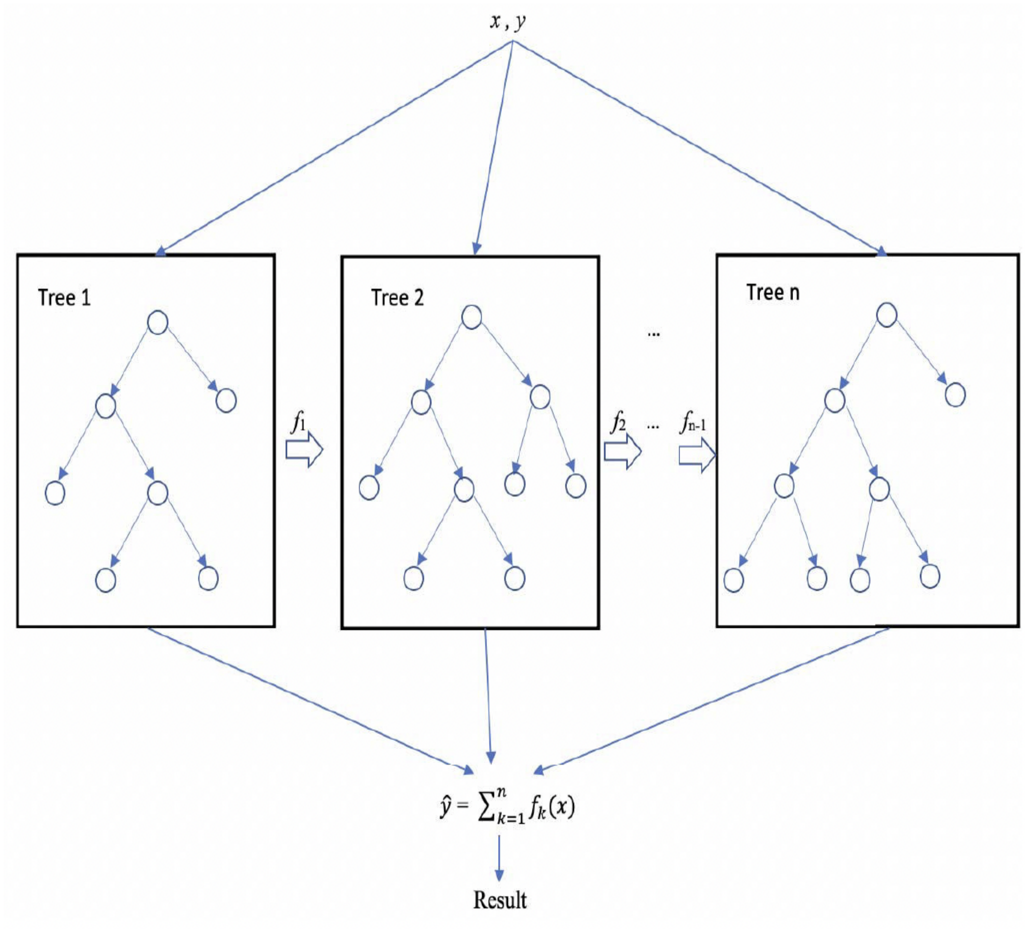
The XGBoost algorithm is a gradient-boosting method applicable to classification and regression problems. XGBoost is a machine learning algorithm based on trees that is extensively used due to its effectiveness Gupta et al. (2022). This Project uses the XGBoost regression model because the target variable that is to be forecasted is the daily pickups, which is a continuous numerical variable. The XGBoost regressor generates an ensemble of decision trees using the boosting method, where these trees have a binary structure and are added sequentially. A decision tree is a representation of decision outcomes that resemble trees. It is made up of nodes, which represent the features, and branches, which represent the outcomes of decisions. The leaves, or lowest nodes, in a decision tree that represents the predicted outcome.

The XGBoost algorithm considers each decision tree as a weak learner. Then combines all the weak learners together to make a stronger model by adjusting and improving fellow learners. This adjustment and improvement is made by training the model in a sequential way where weak learners try to correct the errors of the previous one.

The below figure 1 shows the architecture of XGBoost taken from the paper by Anggraeni et al. (2021).

Figure 1

XG-Boost Model Architecture



Note. Architecture of XGBoost taken from Anggraeni et al. (2021)

From the above XGBoost architecture it can be understood that the final prediction score is the aggregate of all the constructed decision trees (improved weak learners). Below is the equation 1 from the paper by Chen and Guestrin (2016) tree ensemble model

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F} \tag{1}$$

F is the set of regression trees, while f_k is an independent tree from this set, and $f_k(x_i)$ is the output value of this tree for the input instance xi. Here, $\hat{y}_i$ is the predicted value of i^{th} instance.

The XGBoost algorithm first creates an initial decision tree using the data, and the loss will be calculated using the objective function. The model is trained by minimization of the regularized objective function. An objective function is the combination of loss function and a regularization term. The Objective function helps to guide the model during the training process. Below is the equation 2 to calculate the loss function taken from the paper by Chen and Guestrin (2016).

$$\mathcal{L}(\phi) = \sum_{i} l(\hat{y}_i, y_i) + \sum_{k} \Omega(f_k) \tag{2}$$

Here, *l* represents the difference between the predicted output and the actual objective value. There are several loss functions like MSE, RMSE, and R-square to calculate the errors of regression tasks. Out of Which, MSE is the most popular metric for the XGBoost algorithm. The algorithm's primary objective is to reduce these errors by improving the model's parameters. This loss function also helps in finding the best-split point of a tree. If a split point has the lowest loss function, it will be chosen as the optimal split point in the tree. The MSE formula used by XGBoost is shown below in equation 3.

$$\ell = 1/n\ \Sigma\ (y_i - \hat{y}_i)^2 \tag{3}$$

Here, $\hat{y}_i$ is the predicted value and y_i is the actual true value.

The second term Regularization is used to prevent overfitting during the training of decision trees. Regularization is the addition of a penalty term to the loss function to rescue the model from overfitting the training set and from becoming overly complex.

Regularizations include lasso regularization and ridge regularization, also known as L1 and L2 Regularization. L1 regularization adds the penalty proportional to the absolute values of weights, whereas L2 adds their penalty to the square of the weights. Here, the employed weights are the regularization parameters that regulate the severity of the penalty term. Below equation 4 is the formula for Regularization, taken from the paper by Chen and Guestri (2016).

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda||w||^2 \tag{4}$$

Here, λ is the regularization parameter of both L1 and L2 Regularization. Where w is the weight of the point.

Besides the regularized method mentioned above, XGBoost also uses a shrinkage strategy, known as the learning rate, to handle the overfitting during the training of decision trees. Shrinkage involves reducing the weights of each individual tree to reduce its influence on the outcome of the prediction. This is basically where the shrinkage value is low, which means the learner rate is slow, which reduces the probability of the model overfitting. If the shrinkage value is high, the learner rate is rapid and there is a possibility of overfitting. The shrinkage values can be adjusted accordingly, as XGBoost gives this as a parameter where enhancement can be done. This shrinkage parameter is supposed to be consistently less, i.e., 0.01 to 0.1, to avoid overfitting.

Both the regularization and shrinkage techniques help to avoid and prevent the model from overfitting. In the absence of these techniques, a model may be susceptible to overfitting, resulting in subpar performance on new and unseen data.

In this manner, the XGBoost algorithm iteratively constructs new trees that prioritize data points that the previous tree incorrectly classified. New trees are added to the ensemble, and their predictions are combined with those of the previous ones. The tree is grown up until a stopping requirement, such as the maximum tree depth or the minimum sample size required to divide a node. This process is continued till a model performance is improved, which can result in a powerful performance of an ensemble model that predicts the values of a target feature. All this process enables XGBoost to build strong decision trees that adapt well to new data and predict with reasonable scores.

4.2 Model Supports

Environment, Platform, Tools

The hardware used for this project is an Apple MacBook Air M1 which is of eight-core CPU, eight-core GPU, and eight gigabytes of RAM. And the solid-state of storage is 512 gigabytes. And the software used are the GCP Terminal, Google Colab, Google docs and Grammarly

XGBoost can be run on various operating systems like Windows, MacOS, and Linux. XGBoost can be run in different hardware configurations, which means it can be run in a single machine or a group of machines. As mentioned previously, XGBoost supports parallel processing power where it can also use GPUs to make the computations faster. For this Project, MacOS has been used, but there is no specific reason for choosing it, as the performance and functionality of XGBoost are not dependent on the operating system.

XGBoost can be used in Python, R, and C++ programming languages. Python is used for the NYC driver demand project as it is the most used programming language for implementing XGBoost. There are many advantages of using Python as the programming language, as it provides classes and functions that help build and tune the XGboost model.

The environment used for implementing XGboost is Google Colab, a cloud-based interactive computing environment that makes the coding easier and visualizations clear and effective. One of the vital reasons for choosing Google Colab is that it gives access to powerful hardware like GPUs and TPUs, which speeds up the computation time, and large-scale Machine learning tasks can be executed efficiently. Google Colab is widely used for data science projects because it allows data experimentation and supports data exploration, preprocessing, and modeling of the various machine learning algorithms. In addition, it is simple to use and share, and the project pipeline is very apparent step-by-step because multiple processes are performed in a single notebook. Working with Google Colab simplifies documentation because explanations and visualizations can be displayed directly in the notebook. As there are numerous benefits to using Google Colab and the project necessitates the processes mentioned above, Google Colab was selected.

The commonly used libraries with the XGBoost to implement the algorithm are NumPy, pandas, Scikit-learn, matplotlib, and seaborn. NumPy is used for scientific computing purposes in Python. For data analysis and modeling purposes, NumPy is widely used in data science using Python. It is a better fit for XGBoost when the data has a high matrix orientation. Moreover, Pandas library is the most useful and used one in this project as it helps clean the data. Pandas help in manipulating large datasets by providing data structures. Pandas is helpful in transforming data into various formats and helps in the data preparation process. This data prepared as a data frame can be converted into input formats which can then be used by XGBoost for its modeling purpose.

Furthermore, coming to Scikit-learn is a well-known machine-learning library that can be used by a wide range of supervised and unsupervised learning algorithms. This is an essential library for this project as this helps in training and evaluating the models. Scikit-learn also helps in the preprocessing phase, regarding feature scaling by normalizing the data.

Normalization is essential for this project as the target variable consists of a wide range of values. Normalization scales the input features to a normalized range between 0 to 1. This process ensures that all the columns are equally contributed to the model and avoids models dominating the other. Matplotlib and Seaborn libraries are used for XGBoost to show the graphs, statistics, create charts, and visualize the model. Matplotlib and Seaborn are particularly helpful in the data analysis phase. So, in this way, all these libraries show their proficiency in different phases by helping in data cleaning, analyses, feature creation, and transformation, and the data is ready for modeling. The table 1 shown below to show all the methods used for this project.

Table 1

Libraries Used for XGBoost Model

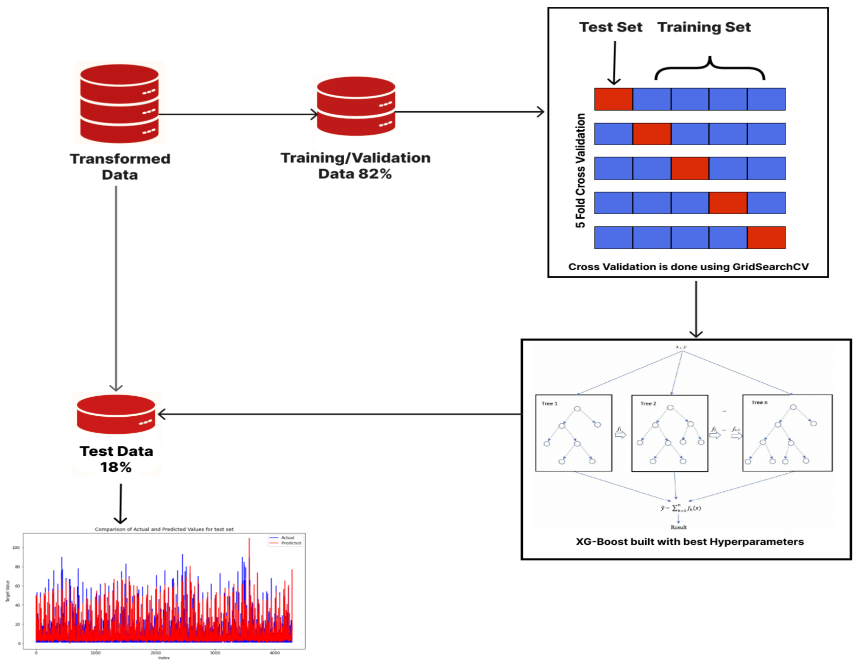
Libarary	Method	Usage
Scikit-Learn	sklearn.preprocessing	Normalizing the data
	train_test_split	splitting the training, validation and test data
	sklearn.model_selection	cross validation and parameter tuning
	sklearn.metrics	provides metrics for evaluation
	fit	Training the model
	predict	To predict output
	score	Evaluate the performance
Pandas	read_csv	Read the data from csv
	head	Top n rows
	describe	Statistical summary of data
	groupby	Aggregating the data
	merge	Merging the data frames with a common column
Numpy	array	Create a array
Matplotlib	plot	Creates a Line chart
	Bar	Creates a bar plot
	scatter	Creates a scatter plot
Seaborn	Scatterplot	Scatterplot with a regression line
	boxplot	To visualize the distribution of data

Model Architecture and Data Flow

As mentioned in the proposal, the data has been preprocessed from its raw form with the help of all these libraries. After preprocessing, the dataset consists of 26456 rows and 8 columns, including the target column ‘Number of Pickups.’ With the help of sklearn.model_selection library’s train_test_split function, the data is separated into datasets for training, validation, and testing. And then the modeling process is done on this split data, the flow of this is shown in the below figure 2.

Figure 2

Datflow Architecture for XG-Boost Model


Transformed Data
Training/Validation Data 82%
Test Set
Training Set
5 Fold Cross Validation
Cross Validation is done using GridSearchCV
Tree 1
Tree 2
Tree n
Result
XG-Boost built with best Hyperparameters
Test Data 18%
Comparison of Actual and Predicted Values for test set
Actual
Predicted
Target Value
Index

After splitting the data, the best hyperparameter is found to train the model. There are various hyperparameters for the XGBoost model, which are a number of estimators, learning rate, maximum depth, gamma, and many more. We can enhance the model by trying a combination of hyperparameters. To evaluate each set of hyperparameters, the cross-validation function is used. This evaluation is done for this project with the help of GridSearchCV. GridSearchCV fits the XGBoost with the training data with all the hyperparameters and calculates the validation score for each hyperparameter.

So based on this evaluation score of the cross-validation function, the best hyperparameters are selected. In this way, XGBoost is built on the best hyperparameters. And finally, this model is used to make predictions on the validation and test sets which will then be compared with the actual value to check the accuracy of the model.

4.3 Model Comparison and Justification

In recent times there are many machine learning algorithms that have been used to predict driver demand. The models provided in chapter1 are LSTM, Random Forest, XGBoost and Facebook Prophet. Though any machine learning algorithm that can handle multivariate time series data is a good choice for the NYC driver prediction, there are a few strengths and weaknesses for each of them; where the algorithm that is chosen should be based on the data which is to be analyzed or forecasted. A few of the algorithms which have been tested and compared with XGBoost in the case of time series data are LSTM, Random Forest, ARIMA, SARIMA, and Prophet.

In the paper by Anggraeni et al. (2021) the computation power of XGBoost is mentioned, which states that XGBoost has high execution speed than any other machine learning algorithm. This is one of the main reasons for choosing XGBoost , as the driver demand project consisted of hug dataset, the computation speed plays a crucial role. As the dataset is huge there is a chance of overfitting, but XGBoost is robust to overfitting as it has built-in functions like shrinkage and regularization which can tackle with such data. Apart from this XGBoost offers with various hyperparameters which helps modelling more efficiently as the predictions can be made better by trying various combinations of parameters

LSTM is a type of RNN and the basic idea behind Long short term memory algorithm is to use its memory cells which have the capacity to store data for an extended period of time. These cells are connected to a set of gates where the information is regulated in and out of the cells. This gating mechanism helps the model learn when to store the information and when to delete it. In this way, an LSTM is able to handle the lengths of the input features; this can be an advantage to time series data as the lengths may vary in it.

For LSTM, the order of the input is crucial as it is designed explicitly for sequential data. That is the reason LSTM can process only time series and natural language text data, whereas XGBoost can process all structured forms of numerical and categorical data. LSTM uses multiple sequential layers when there are multiple inputs to be trained for modeling. This makes the model complex and time-consuming. But when it comes to XGBoost. Zhou et al. (2022) mentioned that it performs exceptionally well when there are more input variables and can also extract the hidden layers. In this case, this Project has various input features like date, day of the week, location ID, and location details. Since XGBoost can efficiently predict the target variable and manage multiple input features, its performance will be dominant.

As discussed earlier, XGBoost is a tree-based approach that uses multiple CPU cores, which makes the computation time less, but in the case of LSTM, it is a neural network, so its computation is typically high when compared to XGBoost. Moreover, XGBoost uses less memory space, and the training time is much faster than LSTM. Furthermore, though both algorithms work better with large datasets, LSTM needs more sparse data. So based on these observations, it is believed that XGBoost is performing better than LSTM for this Project and is well suited for this dataset. A paper by Vanichrujee et al. (2018), also gives more evidence to this, where XGBoost outperformed LSTM in driver predictions in high taxi-required areas.

And coming to the most powerful statistical model for time series analysis and forecasting is the Facebook prophet algorithm. Prophet is an additive/multiplicative regression model with its main components being trend, seasonality, and holidays. These are used to find the underlying patterns and trends in the time series data and use those findings and analysis for future forecasting. In prophet algorithms, they fit the linear regression for modeling the time series data and predict the target variables of the model using the maximum likelihood estimation method. Though the algorithm is widely used for time series forecasting, its modeling process is done by assuming data to be in a linear relationship. However, in the case of the NYC taxi prediction project, the data used is in a non-linear pattern and complex relationship with the data, which makes Prophet a wrong choice where XGBoost will be better in such cases as they can handle both linear and non-linear relationships in the data. Moreover, XGBoost works better with large and sparse datasets, but Prophet falls flat when there are irregularities in the data. Prophet is also less prone to overfitting, whereas the XGBoost algorithm in-built has techniques to reduce the risk of overfitting.

A paper by Preniqi et al. (2020) gives evidence that the XGBoost is performing more excellent results with multivariate data than Prophet and ARIMA. This paper also mentioned that XGBoost has the capability of finding the hidden features of the multivariate data.
In the present project, the data is abundant, and there are many input features that may affect the patterns and trends in the data. So the algorithms that are efficient only for linear dependencies between the variables will be a wrong choice. One such algorithm is the Prophet. This algorithm might work well for linearly dependent data, but in the case of the present Project, they could not be effective.

Random forest is like XGBoost, which uses multiple decision trees for training the data, and based on the average of individual trees, the final model is made. It creates each tree based on random sample sets of data and features. This makes the random forest model more robust to overfitting. Like XGBoost, the random forest model can also handle complex and high-volume datasets. The major difference is that XGBoost builds the trees sequentially, whereas, in random forests, trees are independent and build parallelly. In the research by B. Gupta et al. (2018) it is mentioned that XGBoost performed better than random forest but also stated that when the data provided is enormous, the random forest will improve its performance rapidly and outperform XGBoost. The author of the paper also mentioned that random forest works faster than XGBoost, which means that the computation speed is high for the random forest. So, in comparison, both random forest and XGBoost have strengths and weaknesses, so the better performer is entirely dependent on the data used. In the case of the present Project, both the algorithms performed well by predicting the values accurately with the random forest being slightly better than XGBoost. Below is the table 2 showing comparison between the models

Table 2

Model Comparison

Characteristic	**XGBoost**	**LSTM**	**Facebook Prophet**	**Random Forest**
Architecture	Ensemble Model	Neural Network(Type of RNN)	Additive/Multiplicative Regression model	Ensemble Model
Data Type	Numerical, categorical values and text	Time series data and Natural language texts	Numerical, categorical values and timestamps	Numerical, categorical values and text
Size	Handles small, medium and large datasets	Works better with vast data	Works better on time interval data and can handle small and medium datasets	Cannot handle small datasets
Issues	Not prone to overfitting	Chances of overfitting	Prone to overfitting	Not prone to overfitting
Complexity	High in Scalability, Faster computation, less training time	Less computation speed, training take time	Computation is slow, training takes time	High computation power, training speed is very quick
Strengths	High Speed and good accuracy for structured data	Ability of capturing long term dependencies	Results best accuracy when data is in linear relationship	Can Handle non-linear relationships in the data
Limitations	Sensitive to hyperparameter tuning	Struggles when the data is too small	Struggles with irregular data	Struggle with Noisy data

4.4 Model Evaluation Methods

Evaluation metrics are crucial for time-series data to check the model's ability to make accurate predictions. Based on the results of the metrics used, the forecasted values are assessed. As the project predicts the number of pickups in a location that is a real value, the metrics used to evaluate the XGBoost model are RMSE, MSE, and R-square. As mentioned earlier, these metrics are the most used metrics which can evaluate the performance of the XGBoost regressor model. Any of these metrics is sufficient to evaluate the model but used these three metrics to understand the model's performance better.

Mean Square Error (MSE)

According to Géron (2019 MSE can be defined as the mean squared difference between the predicted outcome and actual true value. When the MSE score is low, it indicates that the model is performing well, but when the score is high that means the model is predicting wrong values compared to the actual. Outliers and errors are something which should be handled before performing evaluations, but there is a chance of small and large errors even after pre-processing is performed. MSE is sensitive to such large errors, though this can help in identifying cases where the model is performing poorly, but it's not the same every time; if there are cases where there are very few large errors, then the score is high, but the performance might be excellent. So this can be an advantage and disadvantage based on the data used. The MSE is calculated from the formula mentioned below as equation 5.

$$\mathrm{MSE} = (1/n) * \Sigma(\hat{y} - y)^2 \tag{5}$$

Here $\hat{y}_I$ is the predicted value and the y is actual value.

Root Mean Square Error (RMSE)

RMSE is very similar to MSE, which is the square root of the MSE. If the RMSE is low, the error difference between the predicted output and the actual true value is less, indicating that predicted values are accurate. But when the RMSE is high, then there is incorrectness in the predicted values. As mentioned by Hyndman and Athanasopoulos (2018) one of the disadvantages of RMSE is that it is susceptible to large errors and outliers in the data. This is directly reflected in the scores of the RMSE, which helps the model concentrate on the data when there are errors or outliers in the data. Below is the formula to calculate the RMSE, which helps to measure the predicted deviation from the actual. Below equation 6 is the formula to calculate RMSE

$$\text{RMSE} = \sqrt{(1/n * \Sigma(\hat{y} - y)^2)} \tag{6}$$

Since RMSE relies on root operations, it is more susceptible to significant errors than MSE. In situations where the objective is to detect large errors and the data are normally distributed, RMSE is preferred, while MSE is effective in scenarios where the data are evenly distributed. In this way the evaluation metrics are chose between MSE and RMSE based on the type of data used.

R-Square

R-square is a measure that is in the range between 0 to 1. Hastie et al. (2013) states that the R-square calculates the proportion of the objective variable's variance. When the R-square value is 1, the model perfectly fits the data and explains more of the variance in the data. But when the value is 0, the model explains nothing about the variance. So, when the score is near 1, the model better fits the data. Though other metrics help identify areas where the model is underperforming, R-square helps to find the overperformance of the model. The formula to calculate R-square is given by equation 9

$$\text{SSR} = \Sigma(y_i - \bar{y})^2 \tag{7}$$

$$\text{SST} = \Sigma(y_i - \bar{\bar{y}})^2 \tag{8}$$

$$\text{R-square} = 1 - (\text{SSR/SST}) \tag{9}$$

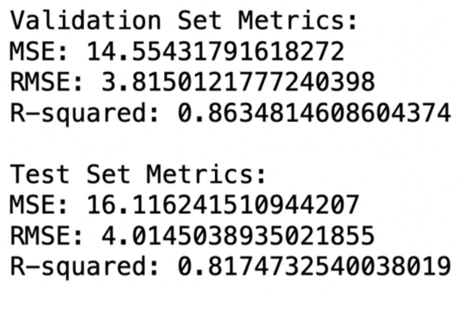
Here SSR in equation 7 is the sum of squares regressions which is used to measure the variability and SST in equation 8 is the total sum of squares which represents the dispersion of the data.

4.5 Model Validation and Evaluation

For this driver demand project, the data is divided into three sets: train data, validation, and test data. The training set consists of 64% of the overall data, which is used for the training purpose of the model. The validation set consists of 18% data used for hyperparameter tuning of the model and preventing it from overfitting. And the testing data is 18% which will be used to examine the model performance and test their accuracy by comparing the predicted and actual. The reason to use the ratio 64:18:18 is that the data used for this project is time-series data, so the splitting is done in a way that the training set consists of 4 months of data, the validation set consists of 1 month of data, and the test set consists of 1 month of data.

Initially, the model is trained without using the hyperparameter tuning, where the baseline model is fit with the training data and uses default parameters for making predictions on the data. Basically, the predictions might not be accurate as the model is not fine-tuned, but this helps the model in finding the real patterns by not getting sensitive to any hyperparameters. This can act as the starting point of the model assessment. The scores obtained after executing the baseline model are shown in Figure 3:

Figure 3



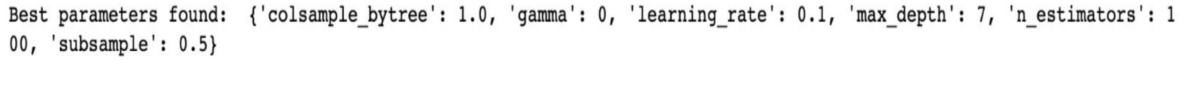
```
Validation Set Metrics:
MSE: 14.55431791618272
RMSE: 3.8150121777240398
R-squared: 0.8634814608604374

Test Set Metrics:
MSE: 16.116241510944207
RMSE: 4.0145038935021855
R-squared: 0.8174732540038019
```

The error difference is very high so, the hyperparameter tuning must be applied to reduce it. For this project, a param_grid dictionary is used, which holds various hyperparameters that are n_estimators, learning_rate, max_depth, gamma, subsample, and colsample_bytree. n_estimators help to fit the model with several trees. When the model chooses a more significant number of trees, the performance might be better, but there is a chance of overfitting, so choosing the number based on the data and problem statement is essential. Even if it is the same with max_depth, increasing the depth of the tree is good for better performance, but there is a chance of overfitting if the number is too high. Learning_rate is used to prevent overfitting. The values of the learning rate are generally smaller values; for this project, the shrinkage step value or the learning rate value is 0.01. Both subsample and colsample_bytree parameters are also used to prevent overfitting, which is used to randomly sample fractions of observation and features for each tree. Moreover, the gamma parameter is used to make the model more conservative by increasing its value.

These parameters are tuned with the help of a grid search. Grid search tries out various combinations of hyperparameters and finds the best set of them. So, this grid search uses cross-validations with CV= 5 to evaluate the performance of each of the hyperparameter combinations. Moreover, the other parameter used in this project is n_jobs, which is set to -1, which specifies that all CPU cores are to be used, which can speed up the search process. As the fitting of the training set to the grid-search object is done, the best set of hyperparameters is obtained with the grid_search.best_estimator parameter. Below figure 4 shows the best hyperparameter found for this model.

Figure 4

Best Hyperparameters for the model



```
Best parameters found:  {'colsample_bytree': 1.0, 'gamma': 0, 'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 1
00, 'subsample': 0.5}
```

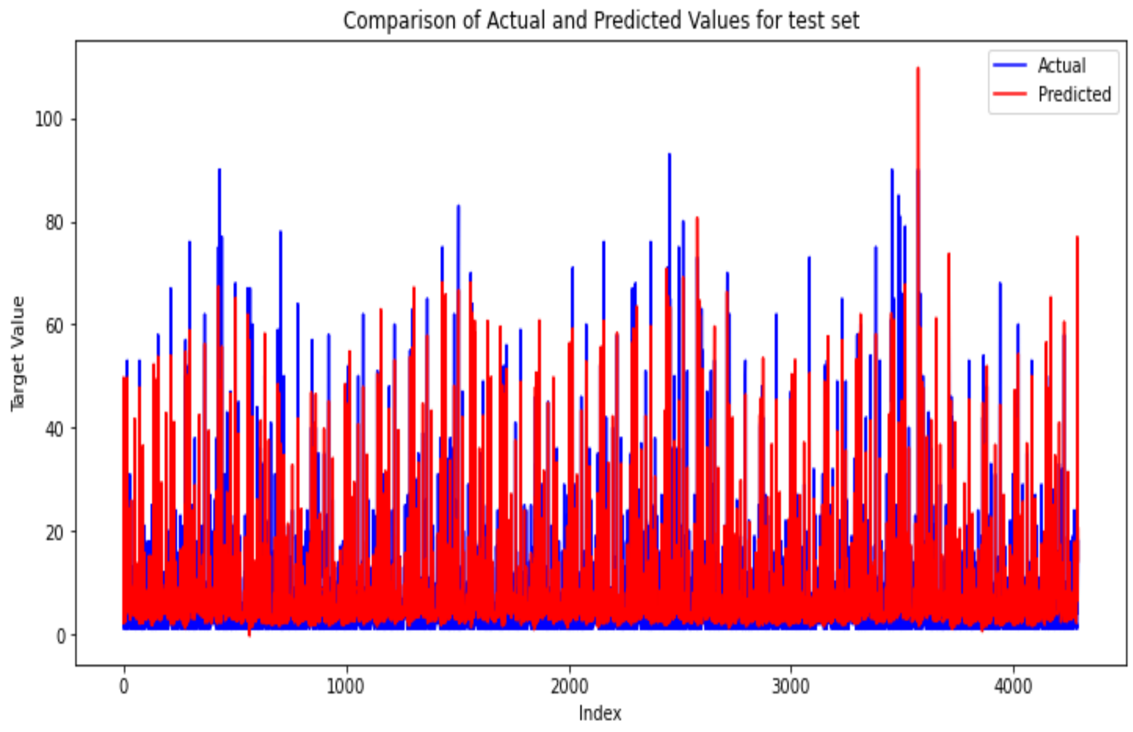
So now, as the model is trained for months, the model is evaluated on the validation set by predicting the number of pickups for that set of dates. Here the model's performance evaluation is done, i.e., the comparison between the actual and predicted values with the help of evaluation metrics MSE, RMSE, and R-square scores. The exact process is repeated for the test set, but here the model's performance is evaluated with the new and unseen data. Validation sets are used to check whether the model is doing good or if there is any overfitting; validation sets also help to find the best hyperparameters for the model.

Both the validation set, and training set have good scores with all the evaluation metrics. Both sets are predicting accurate results, and it is a good fit at the end.

Below figure 5 show the visual representation on comparison between predicted and the actual values.

Figure 5

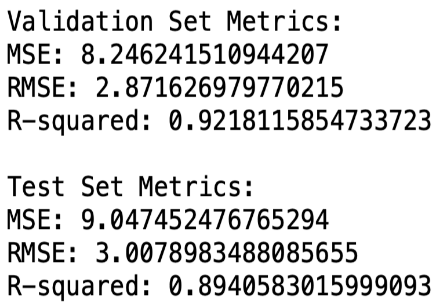
Actual vs predicted results comparison



Below figure 6 shows the results of all the metrics used to evaluate the performance of validation and test sets.

Figure 6

Validation and Test Results for XGBoost Model

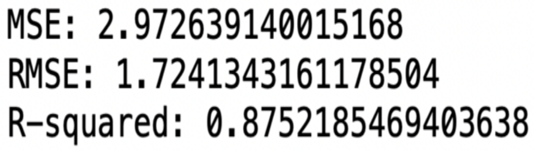


```
Validation Set Metrics:
MSE: 8.246241510944207
RMSE: 2.871626979770215
R-squared: 0.9218115854733723

Test Set Metrics:
MSE: 9.047452476765294
RMSE: 3.0078983488085655
R-squared: 0.8940583015999093
```

To reduce the error scores, the data is smoothed using exponential smoothing. This helped bring down the MSE and RMSE scores near zero and made it even more accurate. The below figure 7 is provided to show the test results after smoothing is performed on the train data

Figure 7

Result After Smoothing The Data

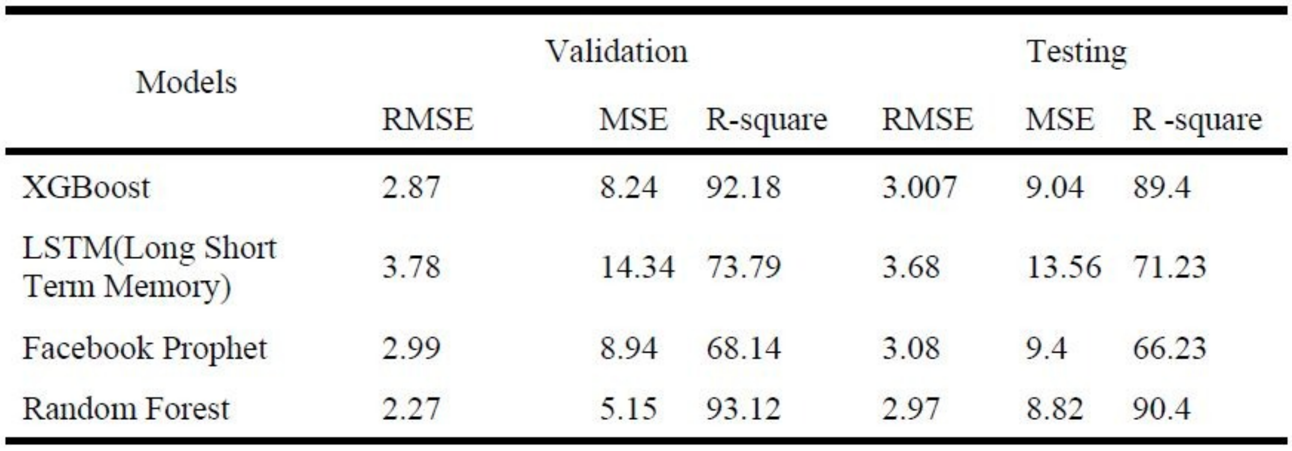


```
MSE: 2.972639140015168
RMSE: 1.7241343161178504
R-squared: 0.8752185469403638
```

The below figure 8 compares the results between all the models used for the project

Figure 8

Model Comparison Results



Models	Validation			Testing		
	RMSE	MSE	R-square	RMSE	MSE	R -square
XGBoost	2.87	8.24	92.18	3.007	9.04	89.4
LSTM(Long Short Term Memory)	3.78	14.34	73.79	3.68	13.56	71.23
Facebook Prophet	2.99	8.94	68.14	3.08	9.4	66.23
Random Forest	2.27	5.15	93.12	2.97	8.82	90.4

Conclusions

After comparing all the results from the proposed model for the project, it is evident that Random Forest is predicting slightly better than XGBoost, but these two are outperforming the other two algorithms Prophet and LSTM. Smoothing is performed on the train data which helped in better performance of XGBoost by reducing the error rate. It is also evident that XGBoost handled the non-linear relationship between the data and made accurate predictions on them. Comparing the evaluation metric results of the ensemble learning models Random Forest and XGBoost to the other algorithms reveals that they both easily handle multivariate data. By reviewing the validation scores and the differences between the validation score and the test score, it is evident that both ensemble models are resistant to overfitting.

Limitations

The MSE and RMSE scores are high without smoothing the data, which can be because of the size of the dataset or the inability of XGBoost in handling outliers in the data. Finding the optimal hyperparameters is time consuming and the sensitivity to hyperparameters might be the reason for the error difference. Though all these are handled after the data is smoothed, but the model is expected to work better with the original data itself.

Future Work

In the future, this model can be trained with a vast dataset to make the model better by training with more information. Many other factors like weather, holidays, and events can be added to the data to make the dataset more informative. There are many other hyperparameter tuning parameters that are to be explored for XGBoost. Can try them to make the model accurate. For this project, a new model is created by combining the ensemble models random forest and XGBoost predicted average values and performed on validation and test sets. Did not explore much on their combination further, as the scores were similar to how the separate models predicted. So, in the future, there is lots of scope for combining XGBoost with other models, which can make predictions even better as a combination of algorithms consistently leverages the potential strength of the respective algorithm

References

Anggraeni, F., Adytia, D., & Ramadhan, A. W. (2021). *Forecasting of Wave Height Time Series Using AdaBoost and XGBoost, Case Study in Pangandaran, Indonesia*. https://doi.org/10.1109/icodsa53588.2021.9617524

Bentéjac, C., Csörgő, A., & Martínez-Muñoz, G. (2021). A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, *54*(3), 1937–1967. https://doi.org/10.1007/s10462-020-09896-5

Chen, T., & Guestrin, C. (2016). *XGBoost*. https://doi.org/10.1145/2939672.2939785

Cherif, I. L., & Kortebi, A. (2019). *On using eXtreme Gradient Boosting (XGBoost) Machine Learning algorithm for Home Network Traffic Classification*. https://doi.org/10.1109/wd.2019.8734193

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. "O'Reilly Media, Inc."

Gupta, B., Awasthi, S., Gupta, R., Ram, L., Kumar, P., Prasad, B. R., & Agarwal, S. (2018). Taxi Travel Time Prediction Using Ensemble-Based Random Forest and Gradient Boosting Model. In *Advances in intelligent systems and computing* (pp. 63–78). Springer Nature. https://doi.org/10.1007/978-981-10-7200-0_6

Gupta, R., Yadav, A. K., Jha, S., & Pathak, P. (2022). Time Series Forecasting of Solar Power Generation Using Facebook Prophet and XG Boost. In *2022 IEEE Delhi Section Conference (DELCON)*. https://doi.org/10.1109/delcon54057.2022.9752916

Hastie, T., Tibshirani, R., & Friedman, J. (2013). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science & Business Media.

Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: principles and practice*. OTexts.

Noorunnahar, M., Chowdhury, A. H., & Mila, F. A. (2023). A tree based eXtreme Gradient Boosting (XGBoost) machine learning model to forecast the annual rice production in

Bangladesh. *PLOS ONE*, *18*(3), e0283452. https://doi.org/10.1371/journal.pone.0283452

Preniqi, V., Mishra, B. K., Thakker, D., Feigl, E., Mokryani, G., Abdullatif, A., & Konur, S. (2020). *Comparative Study of Shortterm Electricity Price Forecasting Models to Optimise Battery Consumption*. https://doi.org/10.1109/ithings-greencom-cpscom-smartdata-cybermatics50389.2020.00069

Vanichrujee, U., Horanont, T., Pattara-Atikom, W., Theeramunkong, T., & Shinozaki, T. (2018). Taxi Demand Prediction using Ensemble Model Based on RNNs and XGBOOST. https://doi.org/10.1109/icesit-icictes.2018.8442063

Zhai, N., Yao, P., & Zhou, X. (2020). *Multivariate Time Series Forecast in Industrial Process Based on XGBoost and GRU*. https://doi.org/10.1109/itaic49862.2020.9338878

Zhou, X., Zhai, N., Li, S., & Shi, H. (2022). Time Series Prediction Method of Industrial Process with Limited Data Based on Transfer Learning. *IEEE Transactions on Industrial Informatics*, *19*(5), 6872–6882. https://doi.org/10.1109/tii.2022.3191980