Food Delivery Time Prediction

Machine Learning Final Project Report

Meet Patel, Mihir Chitre
Khoury College of Computer Sciences
Northeastern University
Boston, USA
patel.meetd@northeastern.edu, chitre.m@northeastern.edu

Abstract

This report discusses the motivation, implementation, and analysis of our final project completed for CS 6140 Machine Learning. The main objective is to present the problem statement, elaborate on the project process and outcomes, and identify potential areas for future investigation. The project focuses on utilizing machine learning algorithms to estimate the delivery time for food orders. The predictive models incorporate various factors, including delivery person information, order details, weather conditions, traffic density, and other pertinent variables. The report provides an overview of the chosen task, a comprehensive explanation of the implementation and application of the machine learning models, and an in-depth discussion of the obtained results.

Keywords: Machine Learning, Regression, Delivery time.

INTRODUCTION

The driving force behind our project stems from the desire to delve into the realm of machine learning classifier models and their practical application to a dataset of appropriate complexity. After thorough consideration, we identified the captivating domain of food delivery time prediction as an ideal candidate for investigation. In this context, our focus revolves around developing predictive models that estimate the time taken for food deliveries based on various crucial factors. These factors encompass delivery person information, order details, weather conditions, traffic density, and other relevant variables. The ability to accurately predict delivery times holds great potential and presents its unique set of challenges. Variability in delivery routes, weather conditions, and other dynamic elements add complexity to the prediction task. However, the successful application of such predictive models can lead to improved operational efficiency in the food delivery industry and enhance customer satisfaction.

DATA

For our food delivery time prediction project, we obtained the dataset from Kaggle[1]. The dataset initially contained 19 features, each contributing valuable information for predicting delivery times. Before utilizing the dataset for model training, we conducted a comprehensive Exploratory Data Analysis (EDA) to gain insights and a deeper understanding of the data.

During the EDA phase, we unearthed valuable patterns and correlations within the dataset, enabling us to make informed decisions for feature engineering and selection. As the data exhibited missing and incorrect values, extensive data cleaning and processing were necessary to ensure the data's reliability and accuracy for model training.

To enhance the predictive power of our models, we engaged in feature engineering, creating additional relevant features to better represent the underlying patterns and relationships in the data. Subsequently, we conducted feature selection to identify the most significant attributes that contribute to accurate delivery time predictions.

Our rigorous data preprocessing and feature engineering efforts laid a strong foundation for training our predictive models. The processed dataset with the engineered features was then used to fit various machine learning models, ensuring high accuracy and reliable predictions.

Data Cleaning

During the data cleaning process, we ensured the dataset's quality and suitability for analysis. Firstly, we converted features like "Time taken" into a numeric format by extracting the relevant numerical values and removing unnecessary text data. Additionally, the "Weather conditions" feature had redundant prefixes to the values, which were removed to achieve uniformity. Some rows in the dataset contained incorrect values for restaurant longitudes and latitudes, necessitating their removal to maintain data integrity.

Next, to prepare the dataset for regression models, we converted all the numerical features to either float or integer data types. Furthermore, we encountered several columns with missing values, such as "Delivery person's age," "Rating," "Weather conditions," "Traffic density," "Festival," "Multiple deliveries," and "City." To handle these missing values appropriately, we leveraged insights gained from our exploratory data analysis (EDA) phase.

Throughout the data cleaning process, our focus was on preserving the data's accuracy and consistency. By addressing these data quality issues, we prepared a cleaned and preprocessed dataset, ready for further analysis and modeling using our machine learning regression models.

Feature Engineering

In the feature engineering phase, we focused on extracting valuable information from existing features and creating new ones to enhance the predictive power of our model for food delivery time.

Firstly, we extracted the city code from the "Delivery person id" feature, considering the vast number of delivery persons. This additional information provided insights into the cities from which the delivery persons operated, which could potentially influence the delivery time.

Next, we calculated the distance between the latitude and longitude coordinates of the restaurant and the delivery location. This new feature, "Distance," captured the spatial relationship between these locations and was more informative than the original latitude and longitude features. Consequently, we removed the redundant "Restaurant latitude" and "Restaurant longitude" as well as "Delivery latitude" and "Delivery longitude" features.

Furthermore, we calculated the time difference between "Time ordered" and "Time order picked" to obtain the duration it took to prepare the order. This newly derived feature, "Order preparation time," provided valuable insights into the order processing efficiency, allowing us to remove the original "Time ordered" and "Time order picked" features.

In addition to the above, we engineered several new features from the "Order date" to capture temporal aspects that could impact delivery time. These features included "Order day," "Order month," "Order year," "Day of the week," "Month start/end indicator," and "Quarter start/end indicator." By engineering these new features, we transformed the raw "Order date" into more meaningful representations that were suitable for regression models.

Through feature engineering, we enriched the dataset with relevant information, optimizing it for regression modeling. The derived features provided a comprehensive understanding of the factors influencing food delivery time, making our predictive model more robust and effective.

Data Preprocessing

As part of Data Preprocessing, we focused on preparing the dataset for predicting food delivery time. To achieve this, we employed the Label Encoding technique to handle categorical features in the dataset. Specifically, we applied Label Encoding to relevant columns such as "Weather conditions," "Type of order," and "Type of vehicle" etc. This process assigned unique numerical labels to each category within these features, allowing us to represent categorical data in a suitable numerical format.

By performing Label Encoding on the categorical features, we successfully transformed the dataset, making it compatible with regression models.

Feature Selection

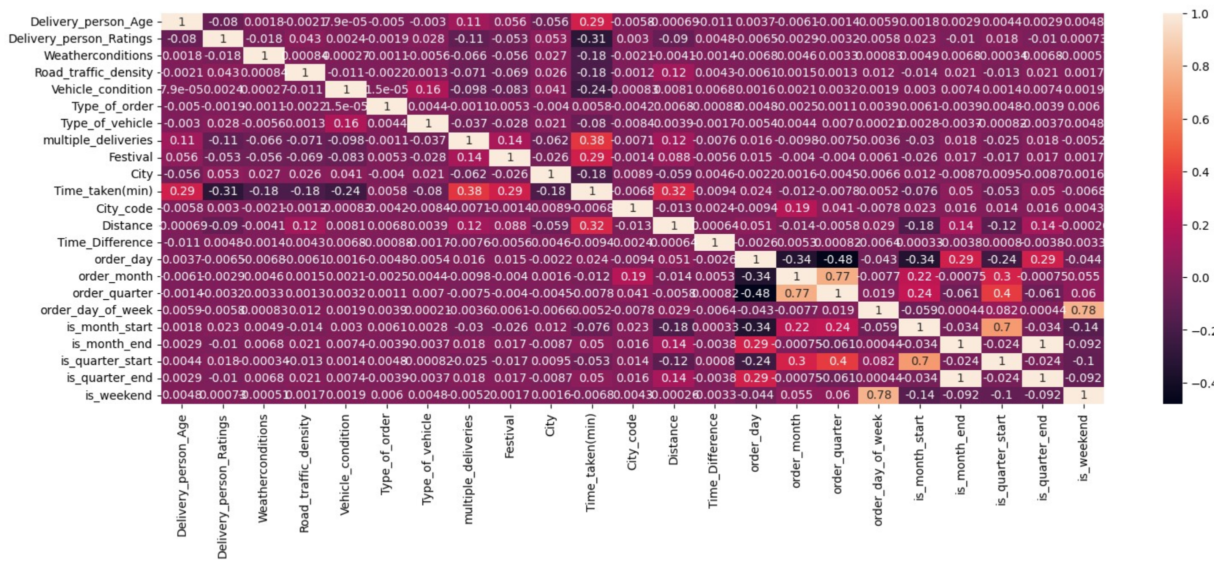
In our food delivery time prediction project, selecting the most influential features plays a crucial role in achieving accurate predictions. To identify the key attributes that contribute significantly to the delivery time prediction, we started by creating a covariance matrix. Figure 1 illustrates the covariance matrix obtained after performing all the necessary data processing steps.

Analyzing the covariance matrix provided valuable insights into the interdependencies between different features. Based on this analysis, we made informed decisions to eliminate features that we deemed to have little impact on the delivery time prediction. Additionally, we removed features that exhibited high correlation with each other to avoid redundancy and multicollinearity, which could potentially introduce bias in our predictive models.

By conducting this rigorous feature selection process, we ensured that the final set of features used for model training captures the most relevant and distinct information, leading to enhanced prediction accuracy and better generalization on unseen data.

IMPLEMENTATION

After conducting EDA and data processing, we obtained a dataset with 19 relevant features for food delivery time prediction. Our implementation utilized standard machine learning libraries like NumPy and Pandas for efficient array and dataset processing. Some models were implemented using popular



Correlation matrix

Python libraries like scikit-learn, while others were custom-built from scratch. We evaluated model performance using the R2 score, a metric indicating the proportion of variance in the target variable explained by the features. This approach allowed us to develop a robust prediction system, offering opportunities for process optimization and enhanced customer satisfaction in food delivery.

Models implemented using libraries

Artificial Neural Net In this implementation, we used TensorFlow and Keras libraries to develop an Artificial Neural Network (ANN) for predicting food delivery time based on a dataset. The core function, "build and train model," allowed us to create and train the ANN with various configurations. Two input parameters, "num layers" and "units per layer," controlled the neural network's architecture, enabling exploration of different layer configurations.

The ANN model was constructed using the Sequential API from Keras, comprising multiple dense layers with ReLU activation functions. The number of units per layer was determined by the "units per layer" parameter, and the output layer had a single unit for predicting the delivery time.

To minimize prediction error, we compiled the model using the Adam optimizer and mean squared error (MSE) loss function. To prevent overfitting, we implemented early stopping with the EarlyStopping callback, monitoring validation loss and halting training if it didn't improve for a specified number of epochs. During training, we used a batch size of 32 and a validation split of 0.2 to assess the model's performance on unseen data. After training, we evaluated the model on the validation set, calculating performance metrics such as MSE, RMSE, MAE, and R-squared (R2) to gauge accuracy and generalization capability. By collecting and storing results from different configurations, including one to five hidden layers and various units per layer, we gained valuable insights into the performance of different ANN architectures for predicting food delivery time. This facilitated the selection of the best-performing model configuration for the given task.

Linear Regression We implemented Linear Regression using scikit-learn's LinearRegression model. The algorithm fits a linear equation to the training data, with coefficients representing the slope and intercept of the line. No hyperparameters were involved in this model, as the algorithm automatically estimates the best fit line during training. Linear Regression assumes a linear relationship between the input features and the target variable. It is computationally efficient and easy to interpret, making it a suitable choice for our initial food delivery time prediction.

Lasso Regression and Ridge Regression Lasso and Ridge Regression are both variations of linear regression with added regularization to prevent overfitting and improve model generalization. We utilized scikit-learn's Lasso and Ridge models, respectively. Lasso Regression introduces L1 regularization, which adds the absolute values of the coefficients to the cost function. This leads to some coefficients being exactly zero, effectively performing feature selection and producing a more interpretable model. Ridge Regression, on the other hand, introduces L2 regularization, adding the squared values of the coefficients to the cost function, which shrinks the coefficients towards zero but does not set them to exactly zero. We performed grid search cross-validation to find the best hyperparameter 'alpha' (regularization strength) with values [0.01, 0.1, 1.0] for Lasso Regression.

Decision Tree Regression Decision Tree Regression is a non-linear algorithm that creates a tree-like structure by recursively splitting the dataset based on feature thresholds. We implemented Decision Tree Regression using scikit-learn's DecisionTreeRegressor. Decision trees are powerful and interpretable, capturing complex relationships within the data. However, they can overfit the training data. To mitigate overfitting, we introduced pruning parameters, such as 'maxdepth', to control the maximum depth of the tree. This allowed us to limit the tree complexity and improve the generalization capability of the model.

Random Forest Regression Random Forest Regression is an ensemble method that combines multiple decision trees to make predictions. We utilized scikit-learn's RandomForestRegressor for this purpose. Each tree in the forest is trained on a random subset of the data and features, and the predictions from all trees are averaged to produce the final prediction. Random Forests effectively reduce overfitting and improve predictive accuracy compared to individual decision trees. We tuned the number of trees ('n-estimators') with values [25, 50, 100] through grid search to optimize the model's performance.

Support Vector Regression (SVR) Support Vector Regression (SVR) is a powerful non-linear regression algorithm that uses support vectors to identify a hyperplane that best fits the data points in a higher-dimensional space. We employed scikit-learn's SVR model with two kernel functions: 'linear' and 'rbf' (Radial Basis Function). The 'linear' kernel works well for linearly separable data, while the 'rbf' kernel handles non-linear relationships effectively. We performed grid search to tune the regularization parameter 'C' with values [1, 10, 100] and the kernel parameter 'gamma' with values ['scale', 'auto'] to control model complexity and non-linearity, respectively.

XGBoost Regression XGBoost (Extreme Gradient Boosting) is a popular gradient boosting library known for its high predictive accuracy and efficiency. We used the xgb.XGBRegressor model from the xgboost library. Gradient boosting involves sequentially training weak learners and combining their predictions to create a strong learner. We tuned the number of boosting rounds ('n-estimators') and the learning rate ('learning-rate') with values [50, 100, 200] and [0.01, 0.1, 0.2], respectively, to optimize the model's performance.

AdaBoost Regression AdaBoost (Adaptive Boosting) is an ensemble method that builds multiple weak learners sequentially, with each learner focusing on instances misclassified by its predecessors. We implemented AdaBoost using scikit-learn's AdaBoostRegressor model. We tuned the number of boosting rounds ('n-estimators') and the learning rate ('learning-rate') with values [50, 100, 200] and [0.01, 0.1], respectively, to improve the model's accuracy. AdaBoost is effective at handling complex datasets and providing accurate predictions by combining multiple weak learners into a strong predictive model.

Models implemented from scratch

In addition to the models we implemented using libraries, we also developed custom implementations of Linear Regression, Ridge Regression, and Decision Tree Regressor. These custom classes were carefully optimized to ensure that they exhibit comparable performance to the library implementations. We took special care to streamline the code and enhance efficiency, resulting in our custom implementations demonstrating good performance on par with the built-in library counterparts.

Linear Regression In our custom Linear Regression implementation, we have created a class LinearRegression to perform simple linear regression. This class includes methods for fitting the model to the training data (fit), making predictions on new data (predict), and extracting the learned coefficients (get-coefficients). The fitting process involves estimating weight coefficients and an intercept term using the Normal Equation. The class provides flexibility by adding a bias term to the feature matrix, which allows for better model generalization. After training the model, we calculate evaluation metrics such as Mean Squared Error, Root Mean Squared Error, Mean Absolute Error, and R-squared to assess its performance on the validation data. Additionally, we retrieve the learned weight coefficients for further analysis, excluding the bias term. This custom implementation enables us to have more control over the model's internals and optimize its performance according to our specific needs.[2]

Ridge Regression In our custom Ridge Regression implementation, we created a class RidgeRegression to perform ridge regression, which was a variant of linear regression with regularization. The class included methods for fitting the model to the training data (fit), making predictions on new data (predict), and retrieving the learned coefficients (get-coefficient).

The regularization parameter alpha was set to a default value of 1.0 and could be adjusted to control the amount of regularization applied to the model. We incorporated the regularization term in the coefficient matrix calculation to handle multicollinearity and prevent overfitting.

To tune the regularization strength, we used Grid Search CV with hyperparameter alpha set to [0.01, 0.1, 1.0], which allowed us to find the optimal alpha value that led to the best model performance.

Using this custom Ridge Regression class, we were able to apply ridge regression to our dataset and better handle potential issues related to correlated features and overfitting.[3]

Decision Tree Regression In our custom Decision Tree Regression implementation, we created a class DecisionTreeRegressor to perform regression using a decision tree. The class included methods for fitting the model to the training data (fit), making predictions on new data (predict), and finding the best split for a node during the tree construction (find-best-split). We used mean squared error (MSE) as the impurity measure for finding the best split.[4]

To prevent overfitting, we included a hyperparameter max-depth, which controlled the maximum depth of the tree. If the depth reached the specified max-depth, or if all target values in a node were the same, we stopped splitting and returned the mean target value of the node as the prediction. For finding the best split at each node, we iterated over all features and thresholds, calculating the MSE for each split. We selected the feature and threshold that resulted in the lowest MSE as the best split for that node.

We used Grid Search CV with hyperparameter max-depth set to [None, 5, 10] to explore different tree depths and find the optimal value that led to the best model performance. Using this custom Decision Tree Regressor class, we were able to construct a decision tree and make predictions on new data. The performance of the Decision Tree Regression model was evaluated using metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R2).

Results

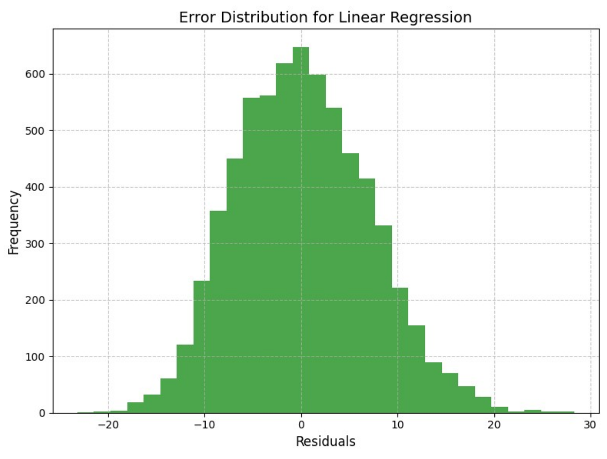
In this section, we present the results of our experiments with different machine learning models for predicting food delivery time. To optimize the performance of each model, we conducted hyperparameter tuning using Grid Search CV, exploring various combinations of hyperparameters for each model.[5] Additionally, we employed K-fold cross-validation to assess the generalization performance of the models on different subsets of the dataset.

For each model, we trained and evaluated multiple instances with different hyperparameter settings to identify the most effective configurations for our problem. The following sections present a comprehensive analysis of the performance of each model on our dataset. We compare the models based on various evaluation metrics, including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R2). Through these analyses, we aim to identify the best-performing model and understand the strengths and limitations of each approach.

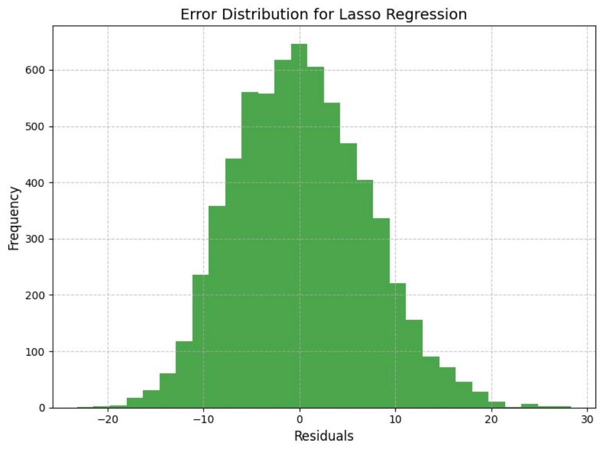
To gain a deeper understanding of the models' predictive performance, we analyzed the instances with the highest prediction errors (extreme errors). For each model, we identified the top 10 instances with the largest absolute errors between predicted and true delivery times.

Models implemented using libraries

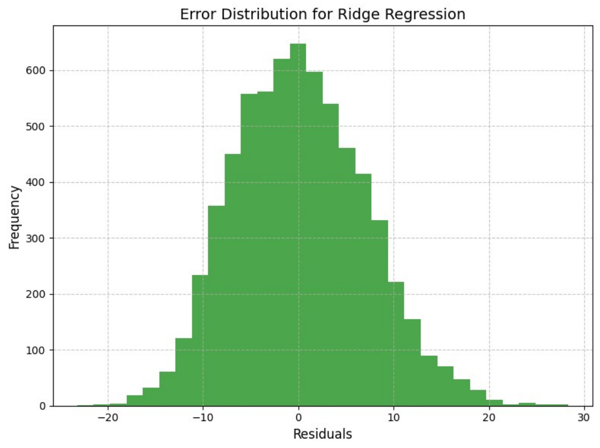
Linear Regression Linear Regression, a simple yet interpretable model, demonstrated moderate performance in predicting food delivery time. The model's mean squared error (MSE) on the validation set was 47.59, indicating that it tends to have larger prediction errors. However, it achieved an R-squared (R2) value of 0.47, signifying that approximately 47 percent of the variance in the target variable could be explained by the model. The model's average performance metrics through K-fold cross-validation improved slightly, with an MSE of 45.01 and an R2 of 0.48.



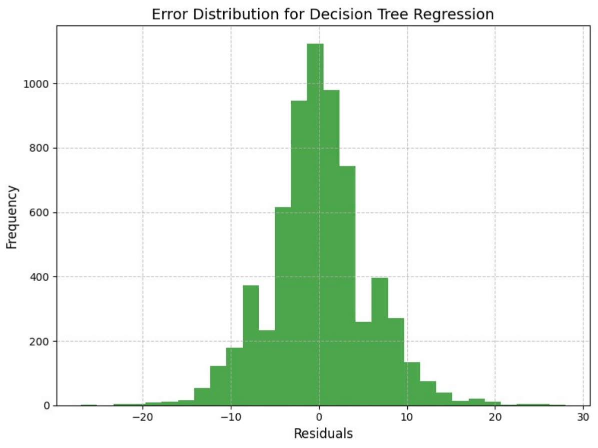
Lasso Regression Lasso Regression, which incorporates L1 regularization to induce sparsity in the model, yielded similar performance to Linear Regression. The model's MSE on the validation set was 47.58, and the R2 value was 0.47, aligning with Linear Regression's results. K-fold cross-validation resulted in slightly improved metrics, with an MSE of 45.01 and an R2 of 0.48. The hyperparameter tuning process with Grid Search CV identified an optimal alpha value of 0.01, leading to a similar performance to Linear Regression.



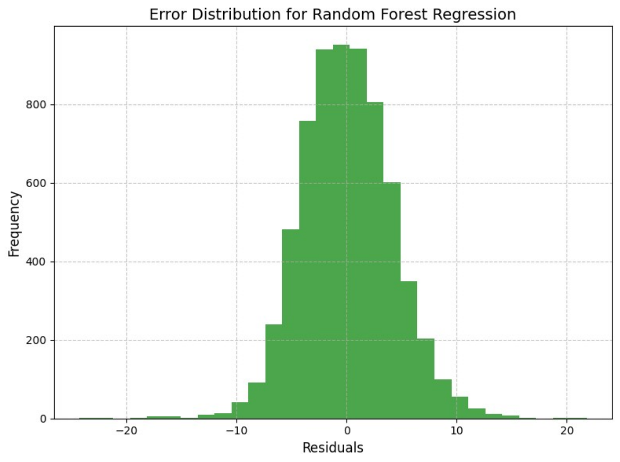
Ridge Regression Ridge Regression, utilizing L2 regularization to prevent multicollinearity, also demonstrated comparable performance to the previous models. The model's MSE on the validation set was 47.59, and the R2 value was 0.47, consistent with the other regression models. The averaged performance metrics through K-fold cross-validation were similar, with an MSE of 45.01 and an R2 of 0.48. The best hyperparameter found with Grid Search CV was an alpha value of 1.0, which slightly regularized the model and improved generalization.



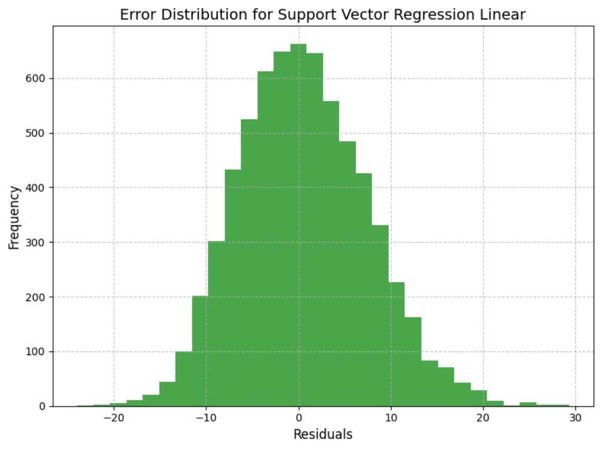
Decision Tree Regression Decision Tree Regression offered a notable improvement in performance compared to the linear models. The model's MSE on the validation set was 31.91, and the R2 value reached 0.65, indicating that the model captured a more significant portion of the target variable's variance. K-fold cross-validation confirmed the model's promising performance with an averaged MSE of 30.94 and an R2 of 0.64. Grid Search CV revealed that a maximum depth of 10 resulted in the best model configuration, preventing the model from overfitting.

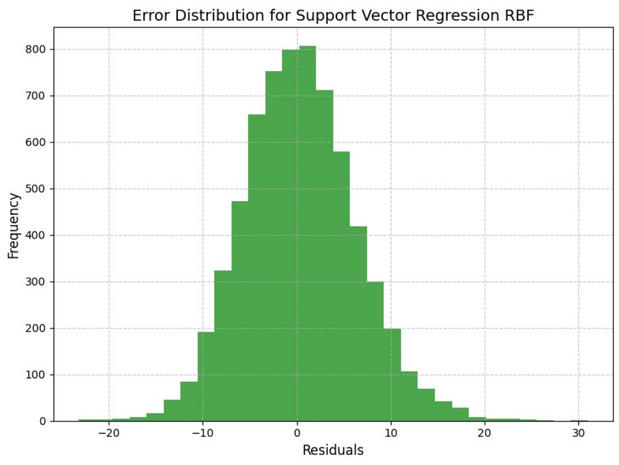


Random Forest Regression Random Forest Regression, an ensemble technique based on decision trees, exhibited significant improvement over individual decision trees. The model's MSE on the validation set was 17.09, and the R2 value reached an impressive 0.81. K-fold cross-validation consistently showed strong performance, with an averaged MSE of 16.54 and an R2 of 0.81. Grid Search CV indicated that utilizing 200 estimators (trees) led to the best-performing Random Forest model.

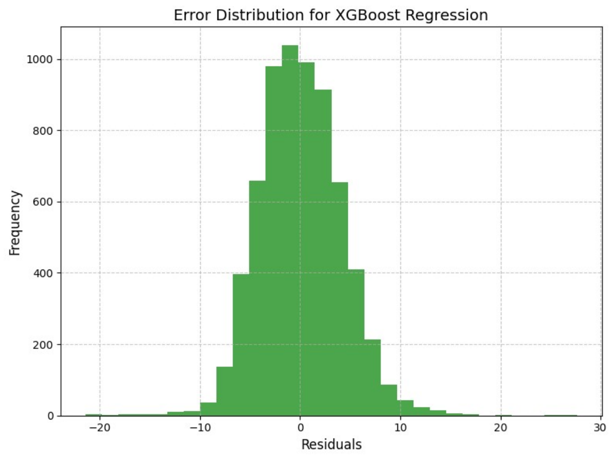


Support Vector Regression (Linear and RBF Kernels) Support Vector Regression (SVR) with both linear and radial basis function (RBF) kernels achieved moderate performance. The linear kernel SVR achieved an MSE of 48.20 and an R2 of 0.47 on the validation set. The RBF kernel SVR performed slightly better, with an MSE of 35.90 and an R2 of 0.60. K-fold cross-validation results showed similar performance trends. Grid Search CV identified an optimal regularization parameter (C) of 10 and an automatic gamma value for the RBF kernel.

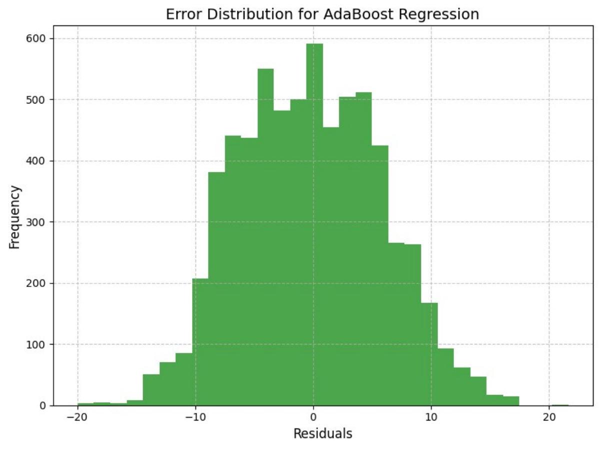


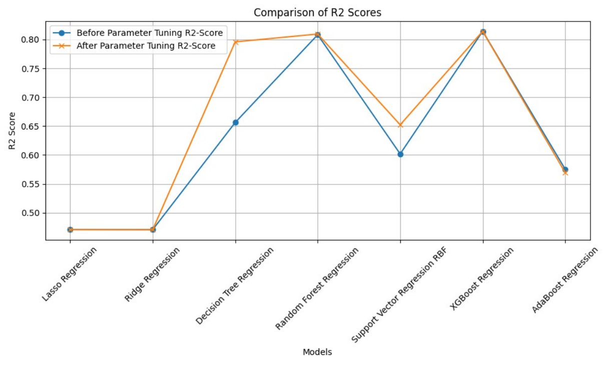


XGBoost Regression XGBoost Regression, a powerful boosting algorithm, delivered outstanding results. The model's MSE on the validation set was 16.73, and the R2 value reached an impressive 0.81. K-fold cross-validation consistently demonstrated the model's robust performance, with an averaged MSE of 16.12 and an R2 of 0.81. Grid Search CV revealed that using 100 estimators (trees) and a learning rate of 0.1 resulted in the best-performing XGBoost model.



AdaBoost Regression AdaBoost Regression, another boosting algorithm, exhibited satisfactory performance. The model's MSE on the validation set was 36.43, with an R2 of 0.60. K-fold cross-validation confirmed the model's performance with an averaged MSE of 37.35 and an R2 of 0.57. Grid Search CV identified a learning rate of 0.2 and 50 estimators as the optimal configuration for the AdaBoost model.





Artificial Neural Net The ANN performance was influenced by the architecture of the network, specifically the number of hidden layers and the number of neurons per layer. As the number of hidden layers increased, the model gained more capacity to learn complex patterns in the data. However, too many hidden layers led to overfitting, causing the model to perform poorly on unseen data.

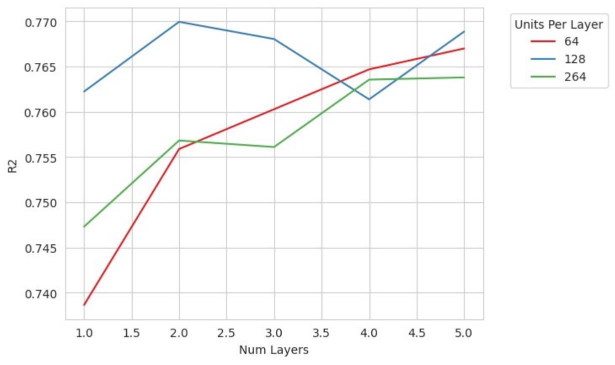
The ANN exhibited promising performance in predicting food delivery time. As shown in table below, the models with 2 hidden layers and 128 neurons per layer achieved the lowest MSE of 20.77 and the highest R-squared (R2) value of 0.77 on the validation set. The models with 1 hidden layer and 64 neurons per layer also performed well, with an MSE of 23.60 and an R2 of 0.74. Overall, the ANN demonstrated competitive performance comparable to other machine learning models implemented using libraries.

The ANN demonstrated competitive performance compared to traditional machine learning models,

Num Layers	Units Per Layer	MSE	RMSE	MAE	R2
1	64	23.60	4.86	3.81	0.74
1	128	21.47	4.63	3.65	0.76
1	264	22.81	4.78	3.76	0.75
2	64	22.04	4.69	3.68	0.76
2	128	20.77	4.56	3.58	0.77
2	264	21.96	4.69	3.68	0.76
3	64	21.64	4.65	3.64	0.76
3	128	20.94	4.58	3.59	0.77
3	264	22.02	4.69	3.65	0.76
4	64	21.25	4.61	3.61	0.76
4	128	21.54	4.64	3.66	0.76
4	264	21.35	4.62	3.62	0.76
5	64	21.04	4.59	3.60	0.77
5	128	20.87	4.57	3.57	0.77
5	264	21.33	4.62	3.59	0.76

Table 1: *Performance of the Artificial Neural Network (ANN) for different configurations*

such as Linear Regression and ensemble models like Random Forest and AdaBoost. The ability of the ANN to capture non-linear relationships between features and the target variable made it an effective approach for this food delivery time prediction task.



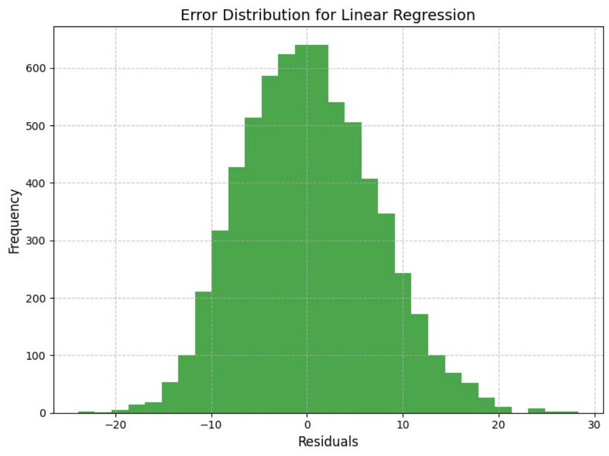
ANN performance graph

Models implemented from scratch

Linear Regression The Linear Regression model was implemented from scratch using Python and NumPy. It fits a linear equation to the training data to predict the food delivery time based on the features. The model's performance on the validation set was evaluated, and it achieved a Mean Squared Error (MSE) of 47.84, a Root Mean Squared Error (RMSE) of 6.92, a Mean Absolute Error (MAE) of 5.56, and an R-squared (R^2) value of 0.47.

The coefficients of the model, representing the contributions of each feature to the prediction of the food delivery time, were also computed. The intercept (b) was found to be 26.22, and the coefficients (w) for the features ranged from -2.02 to 2.48. Positive coefficients indicate a positive relationship with the target variable, while negative coefficients suggest a negative relationship. The model's R^2 value indicates that approximately 47 percent of the variance in the target variable is explained by the model, suggesting that there is room for improvement.

To further evaluate the model's performance and assess its generalization capability, K-Fold Cross Validation with $k = 5$ was performed. The average performance metrics over the five folds were calculated, resulting in an average MSE of 45.01, an average RMSE of 6.71, an average MAE of 5.40, and an average R^2 value of 0.48. K-Fold Cross Validation helps to gauge the model's stability and robustness by training and testing it on different subsets of the data. While the Linear Regression model showed moderate performance, there may be potential for enhancing its accuracy through feature engineering and exploring more complex models that can capture non-linear relationships in the data.

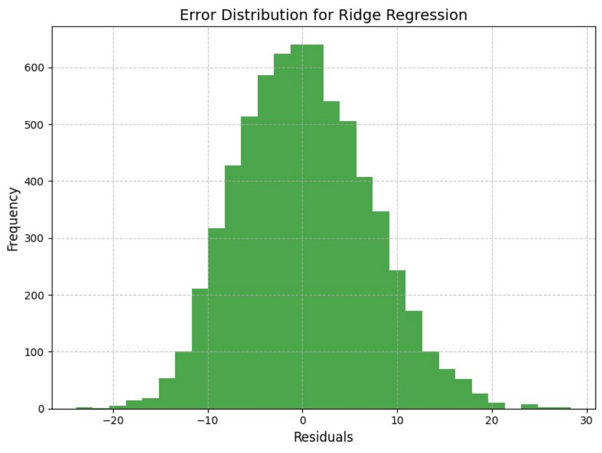


Ridge Regression The Ridge Regression model, a variant of Linear Regression, was implemented from scratch using Python and NumPy. Ridge Regression is a regularized linear regression model that includes an additional regularization term to prevent overfitting. The regularization parameter α was set to 0.01. Similar to the Linear Regression model, Ridge Regression aims to predict the food delivery time based on the provided features.

The model's performance on the validation set was evaluated, and it achieved a Mean Squared Error (MSE) of 47.84, a Root Mean Squared Error (RMSE) of 6.92, a Mean Absolute Error (MAE) of 5.56, and an R-squared (R^2) value of 0.47. The Ridge Regression model's coefficients were computed, representing the contributions of each feature to the food delivery time prediction. The intercept term was found to be 26.22, and the coefficients for the features ranged from -2.02 to 2.48.

To further assess the model's generalization capability and stability, K-Fold Cross Validation with $k = 5$ was performed. The average performance metrics over the five folds were calculated, resulting in an average MSE of 45.01, an average RMSE of 6.71, an average MAE of 5.40, and an average R^2 value of 0.48. The Ridge Regression model demonstrates a moderate performance similar to the Linear Regression model. However, the regularization term in Ridge Regression helps control the model's complexity and may provide better generalization on unseen data compared to the standard Linear Regression model.

Although both the Linear Regression and Ridge Regression models show similar performance on the given dataset, further experimentation with different regularization parameters and feature engineering might lead to an improved predictive performance for the Ridge Regression model.

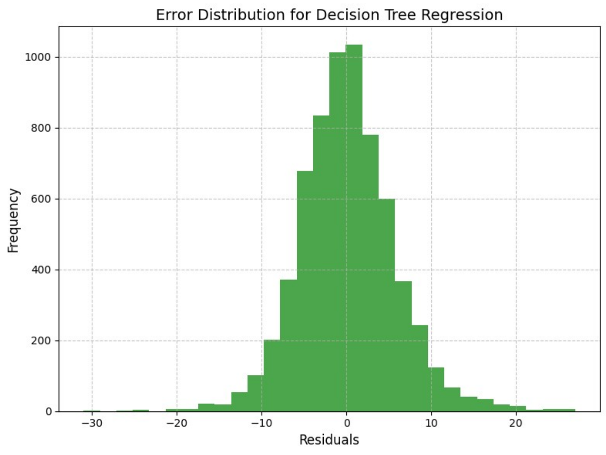


Decision Tree Regression The Decision Tree Regressor was implemented from scratch in Python, a non-linear regression algorithm that partitions the feature space into segments and predicts the target variable based on the average of the target values in each segment.

To assess the model's generalization capability, K-Fold Cross Validation with $k = 5$ was performed. The model was trained and evaluated on five different folds of the dataset, and the average performance metrics were calculated. The Decision Tree Regressor achieved an average MSE of 40.87, an average RMSE of 6.39, an average MAE of 4.85, and an average R^2 value of 0.53. The K-Fold Cross Validation results suggest that the model's performance is relatively consistent across different subsets of the data.

To optimize the model, we conducted an experiment with different maximum depths ranging from 20 to 30. We found that setting the maximum depth to 27 yielded the best results. Above this depth, the model was prone to overfitting, while below this depth, the model was not capturing the underlying patterns properly.

Thus, the custom-implemented Decision Tree Regressor shows promise in predicting the food delivery time, but further optimization and tuning of hyperparameters may enhance its performance. The model's effectiveness in handling non-linear relationships makes it a valuable addition to the set of regression models evaluated in this study. Additionally, as it was implemented from scratch, it offers a deeper understanding of the underlying principles of decision tree-based regression algorithms.



Conclusion

In this food delivery time prediction project, we aimed to develop and evaluate various regression models to predict the time taken for food delivery based on several relevant features. We began by exploring the dataset, preprocessing the data, and performing feature engineering to ensure the data was suitable for training the models.

We implemented multiple regression models using libraries such as scikit-learn and TensorFlow/Keras, as well as from scratch, to comprehensively assess their performance. The models included Linear Regression, Lasso Regression, Ridge Regression, Decision Tree Regressor, Random Forest Regressor, Support Vector Regression, XGBoost Regressor, AdaBoost Regressor, and Artificial Neural Network (ANN) with varying architectures.

By evaluating the models on the validation set, we observed that the Random Forest Regressor and the XGBoost Regressor demonstrated superior performance, consistently outperforming other models. Both models exhibited lower Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE), as well as higher R-squared (R^2) values, indicating a better fit to the data. Furthermore, the Random Forest Regressor and XGBoost Regressor showcased robustness when evaluated through K-Fold Cross Validation, confirming their ability to generalize well to different subsets of the data.

The ANN, with its deep learning architecture, also performed competitively, displaying strong predictive capabilities. However, the optimization of ANN hyperparameters may further enhance its performance. Conversely, traditional linear regression models, while interpretable, exhibited limited predictive power, suggesting the need for more complex models to capture non-linear relationships effectively.

Additionally, we identified the optimal maximum depth of the Decision Tree Regressor as 27, as it offered the best balance between avoiding overfitting and ensuring proper fit to the data. Regularization techniques like this can be crucial when utilizing decision trees.

In conclusion, this project demonstrated the significance of model selection and hyperparameter tuning in regression tasks. The Random Forest Regressor and XGBoost Regressor emerged as the top-performing models for predicting food delivery time. However, each model contributed valuable insights into the complexities of the data. The results provide a solid foundation for future improvements and optimizations, opening avenues to enhance the food delivery experience and optimize delivery processes for the food industry.

Future work

Looking into the future of food delivery time prediction, there's an exciting path ahead that involves refining our models for more accurate predictions. One avenue to explore is the enhancement of predictive accuracy and practicality through feature engineering. Incorporating variables like weather conditions, traffic dynamics, and local events could provide our models with a more comprehensive understanding of the delivery process.

Another promising direction is the exploration of hybrid and ensemble models. By combining our custom-built models with established algorithms such as Random Forests and Gradient Boosting, we have the potential to elevate prediction performance by leveraging the strengths of different approaches.

Deep learning techniques, including Long Short-Term Memory (LSTM) networks and transfer learning, represent a captivating frontier in predictive modeling. These methods can unveil intricate temporal patterns and harness the knowledge from pre-trained neural networks, potentially leading to significant improvements in prediction accuracy.

An essential aspect of our future work involves ensuring the interpretability of our models. Developing techniques to unravel the inner workings of custom models, such as conducting feature importance analyses and utilizing SHAP values, is crucial for building trust and facilitating the real-world deployment of our predictive systems.

Moreover, a key step towards practicality lies in end-to-end implementation and deployment. Transitioning from model development to a functional system involves integrating prediction models into the existing food delivery infrastructure, enabling seamless real-time predictions and improving the overall efficiency of the delivery process.

In a nutshell, our journey ahead involves adding more useful information, trying out different model combinations, exploring advanced techniques, making our models understandable, and putting everything into action for practical use. By doing these things, we're not only making food delivery predictions better, but also contributing to the bigger world of using data to predict things in many different ways.

References

[1] Gaurav Malik. "Food Delivery Dataset". https://www.kaggle.com/datasets/gauravmalik26/food-delivery-dataset?select=train.csv.

[2] Khushbu Kumari and Suniti Yadav. "Linear regression analysis study". In: *Journal of the Practice of Cardiovascular Sciences* 4 (Jan. 2018), p. 33. DOI: `10.4103/jpcs.jpcs_8_18`.

[3] Donald W. Marquardt and Ronald D. Snee. "Ridge Regression in Practice". In: *The American Statistician* 29.1 (1975), pp. 3–20. ISSN: 00031305. URL: `http://www.jstor.org/stable/2683673` (visited on 08/07/2023).

[4] Wei-Yin Loh. "Classification and Regression Trees". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 1 (Jan. 2011), pp. 14–23. DOI: `10.1002/widm.8`.

[5] Rahul Shah. "Tune Hyperparameters with GridSearchCV". https://www.analyticsvidhya.com/blog/2021/06/tune-hyperparameters-with-gridsearchcv/.