# SCS3253 Machine Learning Project: DoorDash ETA Prediction

**Group 3**

Reynold Jong
Pavitra Modi
Denys Pashchenko

# Objective

The main objective of this project is to develop a highly accurate machine learning model to predict the estimated time of arrival (ETA) for DoorDash deliveries [1].

Accurate predictions of ETA are crucial for customer satisfaction as they provide reliable delivery options, enhancing customer trust and loyalty [1]. This, in turn, can drive increased revenue over time [1].

For the company, accurate ETA predictions enable data-driven decisions that optimize delivery processes and resource allocation [1]. By prioritizing orders based on the availability and workload of delivery personnel, the company can improve service efficiency and reduce operational costs [1].

Prior to model development, the dataset will be preprocessed and features analyzed to identify the most relevant variables impacting ETA. Various machine learning algorithms and techniques will be experimented with, and the best-performing model will be selected based on metrics such as minimized mean squared error and the time required to train and run the model. This optimization ensures that the developed model is scalable and can function effectively in real-time scenarios.

# Data preparation

## Data Collection

The dataset used in this project, DoorDash ETA Prediction [1], was obtained from Kaggle. It includes a subset of deliveries made by DoorDash in January and February of 2015 across select cities [1]. The dataset consists of 197,428 rows and 16 columns [1]. The columns are listed in Appendix (a).

The target variable to predict is the total time in seconds between created_at and actual_delivery_time, referred to as target_time in the notebook. All other columns were used as input variables, with additional feature engineering performed to create more relevant variables.

## Data Preprocessing

Given that this dataset is a subset of real data from a large-scale company, its quality is suboptimal. While there are no duplicate rows, there are numerous missing, unreasonable, or extreme values that need to be addressed for consistency and accuracy during model training.

Missing Values

Two methods were used to handle missing values:

1. Impute missing values with the average within defined groups
2. Drop missing values

For numeric variables such as total_onshift_dashers, total_busy_dashers, and total_outstanding_orders, missing values were imputed using their average counts based on the day of the week and hour within a market. Remaining missing values, due to a lack of data for specific day-hour-market combinations, were imputed using the average count based on the hour within a market.

For categorical variables and data such as market_id, actual_delivery_time, store_primary_category, and order_protocol, missing values were dropped directly to avoid introducing bias.

UnreasonableValues

Unreasonable data requires context-specific understanding to determine whether it should be removed. For example, total_onshift_dashers, total_busy_dashers, and total_outstanding_orders should always be greater than or equal to 0 since they are counts. Negative values in these fields were removed.

Pandas [13] and NumPy [14] libraries were utilized to handle missing and unreasonable data due to their capabilities for grouping, filtering, and computing data for filling and removing such values.

Outliers/Extreme Values

The dataset also contained extreme values that needed to be removed to avoid bias. The Matplotlib library [15] was used to visualize distributions with boxplots, guiding decisions on which outliers to remove. For the target variable, total_time, data outside the interquartile range were removed to prevent significant impacts on input variable predictions.

Similarly, estimated_order_place_duration and estimated_store_to_consumer_driving_duration were cleaned in the same manner since they directly relate to total delivery time. For variables like total_items, only extreme outliers were removed, considering the distribution and frequency of smaller orders.

Scaling

Features and outputs were scaled for model training to ensure fairness in feature comparison and reduce computational complexity. This scaling helps improve model performance and convergence during training.

## Feature Engineering

Some columns in the dataset did not adequately explain or predict the target variable, necessitating feature engineering to create more relevant features.

One-hot Encoding [12]

Some of the features were one-hot encoded. We chose to perform one-hot encoding on categorical features. To further improve the performance of our models, we split each category of a feature into its own feature having only 0 or 1 as a value. Although this process increased the total number of features in the dataset, we found this helpful because these categorical feature columns representing 0 or 1 were more correlated to the output variable of total_time. This resulted in us selecting better correlated features than the ones that originally existed in the dataset.

New features

New features were extracted from existing variables, such as month, day of the week, and hour from the created_at column. These features were added to account for busier times within a day, week, and month.

Combination of features

Some features have interrelationships, such as the number of distinct items being correlated with the total number of items in an order. To address this, a new feature was created: a binary variable indicating whether the number of distinct items equals the total number of items.

Bins

Delivery time increases with the total number of items, but the correlation is unclear due to the high frequency of orders with 1-2 items. To address this, total items were grouped into bins of size 2, except for the last bin, to improve prediction accuracy.

Clustering [12]

With 74 store primary categories, clustering was applied to reduce the number of categories. KMeans clustering [12] was performed based on the average delivery time of each store's primary category, resulting in 5 clusters determined using the elbow method. This clustering, combined with labeled store primary categories, enhances delivery time predictions by identifying typical delivery time ranges and subtle differences among categories.

## Challenges

Several challenges were encountered during data preprocessing due to the complexity and variability of the dataset.

Large Dataset

The dataset contains nearly 200,000 rows, many with missing, unreasonable, and extreme data. Decisions had to be made on handling these data points, balancing data retention and model training accuracy.

Picking features

Similar and correlated features required careful selection to ensure relevance and avoid unnecessary complexity. The focus was on choosing the most relevant features to optimize model performance.

Methods to add features

To prevent excessive feature dropping, various feature engineering techniques were explored to add relevant features. Significant time was spent evaluating feasible methods, with final selections outlined in the Feature Engineering section.

# Model Design

The choice of the algorithm for the model building is the main part of the design, then further development on the architecture, which is the tuning of hyperparameters will be needed to enhance the model's performance. Different approaches have been explored, including linear, non-linear and deep learning models.

## Features and data splitting

After data preprocessing and features engineering, new features were added and some of them were dropped. The total number of rows left are 137,825 and the features are listed in Appendix (b).

The data were splitted into training and testing set in the ratio of 80 % to 20%. For neural network training, 10% of the training set was used for the validation set.

## Multicollinearity and Dimensionality [12]

It all started with supervised models and linear regression [12] based models was considered. A simple equation with coefficients could be derived from training. Linear models generally have worse performance but it could be improved by using regularized versions, like Lasso and Ridge regressions [12]. However, it still could not yield good results compared to non-linear and neural networks, because that is not linear and it suffered from multicollinearity and dimensionality issues. Before fitting into the linear regression model, correlation analysis was already done and

the number of features left are listed in Appendix (b), but that would not totally address the issues. Feature importance analysis with Random Forest Regressor [12] and PCA curves could be used to reduce the number of features, which outlined that we only need 12 features in the notebook. Yet, if non-linear models and neural networks are used, then this would not be an issue.

### Advanced Machine Learning Algorithms

The next set of models to be considered are the one leveraging advanced machine learning algorithms. These algorithms could handle multicollinearity and dimensionality [12] issues pretty well, and that's the main reason that some of the less contributing features were still kept in the model. In this project, six algorithms were used and the details are outlined in Appendix (c).

### Deep Learning [10]

The final type of model that was considered is the unsupervised model, namely neural network, which should be a robust model under most scenarios. It could solve complex problems by making complicated formulas between inputs and outputs. Besides, it could recognize hidden patterns and can gradually improve performance by getting more training data. At the same time, this is also the downside of the model in this case because we have a limited amount of data. The complexity of it requires more time to train and it is easy to suffer from overfitting.

### Hyperparameters tuning [12]

Different models might have different optimal sets of hyperparameters which could greatly improve the performance. Therefore, GridSearchCV [12] and RandomizedSearch [12] were utilized to find the best sets of hyperparameters. In specific, RandomizedSearch was used for the best performing advanced machine learning algorithms, because the advanced machine algorithms could have better performance without much reliance on the best set of hyperparameters. Using optimal sets of hyperparameters will just contribute to a slight enhancement in the model performance. GridsearchCV was used for the neural network because optimal sets of hyperparameters make a huge impact. Although it is computational expensive, every combination of the hyperparameters should be tested to get the best performing neural network.

## Model Evaluation

Model evaluation is an important part of the machine learning workflow.It provides insights about model performance and guidance to the selection of the best model. There are several metrics used to assess the performance and cross validations were conducted after getting the initial scores.

## Metrics

The following three performance metrics were selected:

1. Mean Squared Error
2. Mean Absolute Error
3. Root Mean Squared Error

Their definitions are in Appendix (d). The goal of the machine learning models is to minimize the above metrics as low as possible. Because the data are scaled, the values of the metrics should be between 0 to 1. If it is more than 1, that would suggest the model performs poorly.

## Cross Validation

We performed KFold Cross-validation [12] on each of the selected models. This was done as a way of checking if the performance metrics remained fairly constant in each of the folds. The results of cross-validation are listed in the Table below. As shown in the table below, all the models performed well on cross-validation, indicating the robustness of the models trained. In particular, for MAE, XGBoostRegressor [12] performed the best because it had the least value of MAE on each fold. For MSE, XGBoostRegressor [12] performed the best because it had the least value of MSE across all the folds compared to the other models. For RMSE, XGBoostRegressor [12] performed the best because it had the least value of RMSE across all the folds compared to the other models.

| Model | Fold 1 - MAE | Fold 2 - MAE | Fold 3 - MAE | Fold 4 - MAE | Fold 5 - MAE |
|---|---|---|---|---|---|
| RandomForestRegressor | 0.6701 | 0.6709 | 0.6695 | 0.6753 | 0.6718 |
| **XGBoostRegressor** | **0.6360** | **0.6374** | **0.6364** | **0.6408** | **0.6398** |
| HistGradientBoostingRegressor | 0.643 | 0.6496 | 0.6457 | 0.6495 | 0.6495 |

***Table 1: MAE values for all selected models for 5 folds.***

| Model | Fold 1 - MSE | Fold 2 - MSE | Fold 3 - MSE | Fold 4 - MSE | Fold 5 - MSE |
|---|---|---|---|---|---|
| **XGBoostReg** | **0.6506** | **0.6374** | **0.6364** | **0.6408** | **0.6398** |

| | | | | | |
|---|---|---|---|---|---|
| **ressor** | | | | | |
| RandomForestRegressor | 0.7071 | 0.7086 | 0.7085 | 0.7167 | 0.7159 |
| HistGradientBoostingRegressor | 0.6618 | 0.6663 | 0.6629 | 0.6688 | 0.6752 |

***Table 2: MSE values for all selected models for 5 folds.***

| Model | Fold 1 - RMSE | Fold 2 - RMSE | Fold 3 - RMSE | Fold 4 - RMSE | Fold 5 - RMSE |
|---|---|---|---|---|---|
| **XGBoostRegressor** | **0.8066** | **0.8079** | **0.8063** | **0.8112** | **0.8131** |
| RandomForestRegressor | 0.8408 | 0.8418 | 0.8417 | 0.8466 | 0.8461 |
| HistGradientBoostingRegressor | 0.8135 | 0.8163 | 0.8142 | 0.8178 | 0.8217 |

***Table 3: RMSE values for all selected models for 5 folds.***

## Performance

As a summary of the previous sections, the evaluation of the performance of the model is based on the metrics, and cross validations were used to ensure the metrics would not deviate a lot after the models were trained on few different sets of training data. The performance of each sets of models are listed below:

Linear Regression Models

Following table outlines the performance metrics of each linear model

| Model | Mean Squared Error | Mean Absolute Error | Root Mean Squared Error |
|---|---|---|---|
| Linear Regression | 3.16 * 10^17 | 2950231.855 | 562512458.347 |
| Lasso Regression | 0.86 | 0.75 | 0.92 |

| | | | |
|---|---|---|---|
| **Ridge Regression** | **0.77** | **0.70** | **0.88** |

***Table 4: Performance Metrics for Linear models [12]***

From the table above, it shows that the best performing model is the Ridge Regression model. However, the multicollinearity and dimensionality issues were brought up in earlier sections and it was determined that this should not be evaluated further given its limitations.

Advanced Machine Learning Algorithm Models

Following table outlines the performance metrics of advanced machine learning algorithm models

| Model | Mean Squared Error | Mean Absolute Error | Root Mean Squared Error |
|---|---|---|---|
| **XGBoostRegressor** | **0.64** | **0.63** | **0.80** |
| RandomForestRegressor | 0.70 | 0.67 | 0.84 |
| HistGradientBoostingRegressor | 0.67 | 0.65 | 0.81 |
| GradientBoostingRegressor | 0.72 | 0.68 | 0.85 |
| AdaBoostRegressor | 0.89 | 0.79 | 0.94 |
| Support Vector Regression | 0.73 | 0.70 | 0.86 |

***Table 5: Performance Metrics for all non-linear models [12].***

Among all algorithms, XGBoostRegressor has the best performance overall. It has the lowest mean squared error, mean absolute error root mean squared error. This is an expected result as it is well-known to be a robust model providing good predictions. First of all, it requires less hyperparameters tuning and minimal features engineering because it could automatically capture non-linear relationships and interactions. Secondly, it has embed regularization which helps prevent overfitting and works well on future unforeseeable data. Finally, its mechanism is based on the building of decision tree in parallel so it greatly speeds up training and prediction. When the model was trained, it only required 0.6 second, saving a lot of time when doing hyperparameter tuning. The result in the table was based on the tuning of hyperparameters which includes subsample, n_estimators, max_depth, learning_rate and colsample_bytree

Neural Network

| Mean Absolute Error | Mean Squared Error | Root Mean Squared Error |
|---|---|---|

| | | |
|---|---|---|
| 0.74 | 0.69 | 0.86 |

***Table 6: Performance metrics for Neural Network [10] [12]***

Finally, the performance of the neural network is comparable to XGBoostRegressor [5] and HistGradientBoostingRegressor after hyperparameter tuning. It was built as a four-layered model with two hidden layers. The number of neutrons in each layer are 64, 32, 16 and 1 respectively, utilizing ReLU activation function for the first three layers and linear activation for the last layer [12]. Adam [12] optimization was used for compiling and the learning rate was 0.0005. To address overfitting issues, regularization measures were done: two dropout layers with dropout rate of 0.2 were added to the two hidden layers, in addition to batch normalization. L2 regularization was not added because it worsened the performance of the model after several experiments.

## Conclusion

The objective of the project was to predict the time when order was created to the time the order is delivered using a combination of different machine learning algorithms.
The goal was to leverage those techniques to build a robust, accurate with good performance model to predict ETAs based on historical delivery data.

The results indicate that the model performs well in predicting ETAs, capturing relationships and patterns in the data. But, there is room for improvement.
One of the ways to improve results is to enrich features with additional external data sources, such as weather conditions, real-time traffic information, local events, which might affect delivery times. In this way we can provide more context to improve prediction accuracy.
We can also try to experiment with more sophisticated feature engineering techniques, such as interaction terms between key features, polynomial features and lag variables for capturing temporal dependencies.
We've just tried a few ML algorithms, so we can explore other techniques, which can capture non-linear relationships more effectively. Ensemble methods combining multiple models could also be considered.
It's good to try to conduct more extensive hyperparameter search using techniques like Random Search [12] or Bayesian Optimization [12] to fine-tune the model parameters and improve performance.
One of the most effective ways to improve prediction is to implement the model in a real-time prediction system and continuously monitor its performance. Based on feedback to update the model with new data, and at the same time ensure it remains accurate and relevant.

Conduct detailed error analysis to identify specific cases where the model underperforms., analyse this data. Understanding these outliers can provide insights into potential improvements and adjustments needed for the model.

This project highlights the importance of feature engineering, data preprocessing, and applying different approaches and algorithms. While the current model offers a solid foundation, further improvements can be made through feature enrichment, advanced modeling techniques, and continuous optimization. Potentially this will enhance the efficiency of DoorDash delivery operations.

## References


[1] D. Suryaa, “Doordash Eta Prediction,” Kaggle, https://www.kaggle.com/datasets/dharun4772/doordash-eta-prediction (accessed Aug. 1, 2024).

[2] L. Breiman. "Random forests." *Machine learning* 45 (2001): 5-32.

[3] R.Tibshirani. "Regression shrinkage and selection via the lasso." *Journal of the Royal Statistical Society Series B: Statistical Methodology* 58.1 (1996): 267-288.

[4] A. E. Hoerl and R. W. Kennard. "Ridge regression: Biased estimation for nonorthogonal problems." *Technometrics* 12.1 (1970): 55-67.

[5] T. Chen and C. Guestrin. "Xgboost: A scalable tree boosting system." *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016.

[6] Breiman, Leo. *Classification and regression trees*. Routledge, 2017

[7] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." *the Journal of machine Learning research* 12 (2011): 2825-2830.

[8] Friedman, Jerome H. "Greedy function approximation: a gradient boosting machine." *Annals of statistics* (2001): 1189-1232.

[9] Freund, Yoav, and Robert E. Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting." *Journal of computer and system sciences* 55.1 (1997): 119-139.

[10] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *nature* 521.7553 (2015): 436-444.

[11] Chollet, François. "Keras: The python deep learning library." *Astrophysics source code library* (2018): ascl-1806.

[12] A. Géron. “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition”.

[13] “Pandas,” pandas, https://pandas.pydata.org/ (accessed Aug. 1, 2024).

[14] NumPy, https://numpy.org/ (accessed Aug. 1, 2024).

[15] “Visualization with python,” Matplotlib, https://matplotlib.org/ (accessed Aug. 1, 2024).

# Appendix

(a) All columns in the dataset (detailed descriptions are provided in the Jupyter Notebook) [1]

- market_id
- created_at
- actual_delivery_time
- store_id
- store_primary_category (food category)
- order_protocol (the method the restaurant receives orders)
- total_items
- subtotal
- num_distinct_items
- min_item_price
- max_item_price
- total_onshift_dashers (number of delivery persons working)
- total_busy_dashers (number of delivery persons busy with orders)
- total_outstanding_orders
- estimated_order_place_duration
- estimated_store_to_consumer_driving_duration

(b) Finalized Features:

- market_id
- store_id
- subtotal
- estimated_order_place_duration
- estimated_store_to_consumer_driving_duration
- month
- day_of_week
- hour
- busy_dasher_ratio
- category_cluster
- store_primary_category_label (Label encoded)
- order_protocol (one-hot encoded, contributing 7 columns)
    - order_protocol_1 dropped
- total_items_binned (one-hot encoded, contributing 5 columns)
    - total_items_binned_1-2 dropped
- max_min_price_same (whether maximum and minimum item price equals)
- all_items_dsistinct (whether all items are distinct)

(c) Details of the six algorithms used:

- **RandomForest Regressor** [12]
    - This model is able to handle complex interactions and non-linear relationships between features. Robust to overfitting, can achieve good performance with default hyperparameters, and good interpretability.

- **AdaBoostRegressor** [12]
    - Ensemble method that combines multiple weak learners to creates trong predictive model. It corrects the errors of the previous learners. Improve accuracy of weak learners, simple to implement, less prone to overfitting.

- **GradientBoostingRegressor** [12]
    - Advanced boosting technique that builds an ensemble of trees sdquentially, where each new tree aims to correct the errors of the combined ensemble or previous tree. Good predictive accuracy, robust to overfitting when properly tuned, handles a variety of loss functions.

- **HistGradientBoosting Regressor** [12]
    - Speeding up training on large datasets. Fast and scalable, often outperforming other gradient boosting implementations on large datasets, efficient memory usage.

- **XGBoosting Regressor** [12]
    - It's an advanced implementation of gradient boosting that is known for its speed and good performance, enhanced predictive accuracy and prevent overfitting. It works efficiently on datasets with missing data, support for parallel processing.

- **Support Vector Regression (SVR)** [12]
    - It is used for regression tasks. It attempts to fit the best line within a threshold value and uses kernel functions to handle non-linear relationships. It's effective in high-dimensional spaces, robust to outliers with the use of epsilon-insensitive loss function.

(d) Definitions of the metrics [12]:

- Mean Absolute Error
    - Measures the average of the absolute differences between the actual and predicted values. It's also non-negative and values closer to zero mean better fit. It treats all errors equally without giving more weight to larger errors, making it more sensitive to outliers.
- Mean Squared Error
    - Measures the average of the square of the errors, which is the average squared difference between the actual and predicted values. It's always non-negative, values closer to zero mean better feat. But it gives more weight to larger errors, making it sensitive to outliers.
- Root Mean Squared Error
    - Square root of the Mean Squared Error. It provides an error metric on the same scale as the original data. Gives more weight to larger errors due to squaring before averaging. It is preferred when the difference between predictions and actual values are expected to follow a Gaussian distribution.