CS M148 Group Project Report: DoorDash Delivery Analysis

Benjamin Gelman, Charlie Hoose, Riley Leong, Om Phadke, Shawn Reznikov

Main Project

i. Data Set

For this project we used the DoorDash Delivery Dataset, which includes operational and behavioral features related to delivery performance. Key variables in this dataset include delivery duration, delivery distance, order value, wait time, preparation time, and other operational timestamps. The dataset contains a large number of observations and it is suitable for insightful analyses and machine learning techniques. We sourced this dataset from Kaggle.

ii. Problem Overview

The goal of this project is to predict DoorDash delivery duration using available operational data. More specifically, we are seeing if we can predict if an order will be abnormally slow. This is important for improving ETA accuracy, optimizing driver allocation, and enhancing customer experience by minimizing complaints. Our analysis would provide DoorDash the ability to make informed optimization decisions and reduce overhead costs. By being able to predict orders that would be excessively slow, DoorDash has the ability to create more accurate wait times and avoid customers getting angry at an optimistically incorrect delivery estimate. Overall, using these machine learning techniques, we are able to capture the complex patterns of delivery times and make informed insights to improve DoorDash's delivery prediction systems.

iii. Methodologies

In our approach to predicting slow DoorDash deliveries, we evaluated multiple machine learning techniques such as linear and logistic regression models, random forests, PCA and clustering, and even complex models like neural networks. After careful consideration of the benefits and tradeoffs for every model, we concluded that our final methodology would be the Random Forest ensemble method. We chose this method because it handles interactions between nonlinear features and class imbalance, something that was present in the dataset that we were working with. Random Forest was applied to predict whether orders would be slow, with a slow order being defined as having a delivery duration that was above the 66th percentile. The model

used all available features after preprocessing, with delivery_duration_min being removed to prevent data leakage. We then implemented a preprocessing pipeline with median imputation to handle any remaining missing values. We used this method because it is more robust to outliers, something that was prevalent in our dataset. In the model training, we used 5-fold CV to maintain the 1:2 class distribution ratio of slow versus non-slow orders across all folds. We then used RandomizedSearchCV for hyperparameter tuning rather than a grid search because it provided enough exploration of the hyperparameter space while not being as computationally expensive. The chosen hyperparameters for our Random Forest method included the number of trees in the forest, maximum depth, minimum samples required to split a node, minimum samples per leaf, and the number of features considered at each split. The specific search space provided for RandomizedSearchCV was chosen to prevent overfitting and enhance model generalizability, since earlier models severely overfitted. We optimized using ROC AUC as the primary metric because it evaluates model performance across all classification thresholds.

Random Forest was ideal for this problem for several reasons that are supported by our analysis. First, the nature of delivery prediction involves certain feature interactions that decision trees capture through the way they partition recursively. For example, the impact of outstanding orders on delivery speed depends heavily on the number of available dashers, creating multiple interactions that a linear model cannot represent as appropriately. Second, our feature set included both continuous variables and categorical variables, which Random Forest handles without requiring multiple scaling and standardizing. Additionally, analysis through PCA and clustering validated our Random Forest approach, as PCA revealed that there weren't any prominent PCs that explained most of the variance. Without this dimensionality reduction, Random Forest's approach of using full features was justified and it makes sense why it performed the best. Lastly, Random Forest being an ensemble method also made it very robust to some of our data's characteristics. Individual decision trees are prone to overfitting, but aggregating predictions across 200 trees with different random feature subsets and bootstrap samples reduces variance while maintaining the low bias of deep trees. Overall, by choosing Random Forest as a method of classification, we can most accurately identify slow orders.

iv. Results

The Random Forest Classifier model performed decently well at predicting slow deliveries in the DoorDash dataset. Metrics that were output included accuracy, precision, recall, F1-score, and ROC AUC, as these would provide a good idea of the model's performance. The best hyperparameters found were 200 estimators, a minimum samples split of 10, a minimum samples leaf of 5, a maximum depth of 20, and sqrt for max features. The metrics for training were the following: 88.12% accuracy, a true positive rate of 0.853, a true negative rate of 0.8958, and an F1-score of 0.8301. It is important to compare these metrics with the cross-validation metrics to look for any overfitting or underfitting.

Across cross-validation, the final model achieved a best ROC AUC of 0.8217, with validation accuracy of 75.8% and F1-score of 0.649, meaning the model got solid performance on the imbalanced dataset. Compared to the training metrics, the lower cross-validation performance suggests the existence of some overfitting despite the actions taken to prevent it, but the model still has the ability to generalize relatively well.

Overall, the model achieved a test accuracy of 75.7%, with a test ROC AUC of 0.820 and an F1-score of 0.647. With an AUC of approximately 0.82, the model shows moderate ability to discriminate between slow and non-slow deliveries. These metrics therefore suggest that the model can reasonably determine slow vs. not slow deliveries accurately, with room for improvement. Additionally, a probability threshold of 0.503 was calculated on the training set in order to maximize F1-score, balancing precision and recall for the “slow” class. This was done to ensure that the threshold was as optimal as possible and didn’t simply assume the standard 0.5. This resulted in a 0.647 precision and 0.646 recall for the “slow” class, meaning it correctly predicts about 65% of truly slow orders while about 65% of its slow predictions are correct. Additionally, a confusion matrix was created to help visualize the distribution of predictions. It shows the predictions being mostly balanced, predicting most labels correctly. Also, a test ROC AUC curve was visualized to help understand the model’s performance even further.

Last but not least, both impurity-based and permutation importance were used to assess feature relevance. The most important predictors include total outstanding orders, total on-shift dashers, created hour, and estimated store-to-consumer driving duration, all reflecting delivery workload, dasher capacity, and DoorDash system congestion. Other important or moderately influential features include subtotal, created weekend, estimated order placement duration, total busy dashers, and item price statistics (maximum and minimum item price). This makes sense, as these variables describe demand, dasher availability, timing, and order complexity. Features such as specific market IDs and order protocols show lower importance, suggesting they contribute very little predictive value beyond the operational and workload-related features above.

Although we are mainly discussing the accuracy and metrics of our Random Forest model, we created many different types in order to make the best possible model. The other models not included in this results section, including linear regression, logistic regression, PCA/clustering, and neural networks, are all discussed later on in the appendix, and their respective Google Colabs are in the repository, as well.

Some limitations of our approach are that we used an arbitrary threshold of the 66th percentile to determine the “slow_order” class, which was our response variable. We figured this was the best point since a delivery taking more than ~50 minutes is a long time, and that threshold would leave a somewhat balanced dataset while still making logical sense. It would be illogical to split it at the 50th percentile and try to predict a 50/50 split of faster and slower orders, too, as an order in the 51st percentile would be considered “slow” in that case. It would be more accurate to say that is slightly above average, and to avoid misclassification of slow in a real life sense, we chose to identify the 66th percentile and onwards as slow. In addition, the gap

between training and validation performance implies there is still some overfitting, which could be addressed through stricter hyperparameters or model constraints. Last but not least, the dataset we used did not have many features surrounding the real world, such as weather or holiday proximity, which could have limited our performance since we're only dealing with the data that DoorDash has. Moreover, knowing the distance between the restaurant and the home would be extremely beneficial in creating more accurate models, however privacy concerns surely block that as a possibility moving forward. Future steps could include addressing issues such as the evident overfitting and incorporating external datasets to improve model accuracy and generalizability.

v. Code

Our main Google Colab notebook loads the original DoorDash dataset directly from our GitHub repository (using a raw file link), so no manual downloading or uploading data is required. For the appendix portions, you can either download the dataset from our GitHub repository and upload it directly into the Google Colab prior to executing the notebook, or run the EDA/Preprocessing part of our main Google Colab notebook ("Random_Forest_Final_Model.ipynb"), uncomment the code block that saves the dataset prior to developing our final model, and download the resulting dataset. This dataset can then be uploaded into Colab, as it is identical to the dataset that is used in the remaining notebooks. The main Google Colab notebook still holds all the required packages, the cleaned dataset, methodology for our best model, and the results. There are also additional Colabs for other components of the project, including linear regression, PCA, logistic regression, and neural networks. All of this can be found in our GitHub repository: https://github.com/rtleong/CSM148_DoorDash_Project_Fall25

Appendix

i. Exploratory Data Analysis (EDA)

For the exploratory data analysis portion of the project, we looked into various intricacies, relationships, and distributions of the data. To visualize the data, we mainly created bar charts and scatterplots. First, we made a plot of the percent of NAs per column in the dataset, which is important for imputation vs drop vs ignore purposes later on in the preprocessing. We also examined the distribution of the variable, delivery_duration_min, as that will be important for our later models. Continuing, we looked at relationships between that response variable and other variables in the dataset, like mean delivery duration based on the hour the order was placed and the amount of items in the order. We also examined delivery duration with respect to the cost of the order. All of these showed some hidden patterns in the data, which will be important for the later steps. Overall, by using EDA, we were able to better grasp what data we were looking at. To see the plots and more analysis of each plot, please look at the Google Colab titled "Random_Forest_Final_Model.ipynb," which contains all of the EDA code and a deeper analysis.

To split our data for training and testing, we decided to go with a stratified split on our response variable, slow_order, for binary classification due to its imbalanced class distribution. We wanted to ensure that the train and test datasets were both generalizable (to a certain extent). We saw this to be the most fitting given our task and problem.

ii. Data Preprocessing & Engineering

With regards to data pre-processing, we first started by ensuring that timestamp variables (created_at and actual_delivery_time) were both converted into proper datetime formats rather than strings. From these, we engineered predictors that we believed could influence delivery timing, such as the hour the order was created, the hour it was delivered, and whether the order was placed on a weekend or not, and afterwards removed the original datetime columns (which cannot be used directly by most models). We also created our continuous response variable delivery_duration_min.

Additionally, we addressed certain issues in the data that we noticed during EDA. Primarily, some numeric columns contained negative values (such as outstanding orders and the number of dashers on shift), which the dataset description noted was intentionally added as noise. Instead of removing these entire rows and making our dataset smaller, we clipped these values at zero to preserve realistic bounds. We also dropped a very small number of rows that had NA values for actual_delivery_time, since this variable was essential in computing delivery duration. We also created our binary classification target variable, slow_order, which we defined as an order whose delivery duration exceeded the 66th percentile (approximately 51 minutes).

Furthermore, we decided to drop store_id and store_primary_category. These variables are mainly identifiers and don't contain meaningful information for our predictive models. Additionally, with over 6,000 unique store IDs, including it would increase dimensionality without adding usable structure to our model. Similarly, while store_primary_category may provide some information about restaurant type, it would require creating over 75 dummy variables, which would unnecessarily expand the amount of columns the dataset has and increase the risk of overfitting. For these reasons, we removed both variables just for model simplicity.

We also dropped rows with missing values in market_id and order_protocol, since imputing these categorical identifiers (with mode, for example) would bias the dataset towards the most common categories while we also do not know the underlying meaning of these IDs. Because only a small number of rows contained missing values, removing them had minimal impact. Afterward, we also converted these variables to integers and one-hot encoded them so that the models could correctly interpret them as categorical.

To avoid data leakage, we applied all train/test splitting prior to any imputation or scaling. After splitting, we addressed outliers in delivery duration by removing observations that were above the 99.9th percentile in both the training and test sets (using a cutoff point that was determined from only the training distribution). By trimming the top 0.1% of observations, this allows us to

remove unrealistic outliers while preserving essentially all of our dataset. On the other hand, lower-end delivery times were kept because they appeared realistic.

Our final dataset that we used was before median imputation and scaling. We did not apply mode imputation because all categorical variables did not contain any missing values. By keeping the data in this pre-imputed and pre-scaled form, we ensured that imputation and scaling only occurred within each fold of our cross-validation pipelines for our models, as this would prevent information from the full dataset to leak into the cross-validation process and ensure that our evaluation metrics were not artificially inflated.

iii. Regression Analysis

Linear regression analysis was applied to our project in an attempt to predict the variable, delivery_duration_min. This could help in our project's goal of identifying which DoorDash orders are abnormally slow, as knowing how long something would take could easily help classify it lagging. To do this, we created both Least Squares (LS) and Least Absolute Deviation (LAD) models using the ten most correlated variables to delivery_duration_min. Of this, they had respective R^2 scores of 0.20 and 0.17, rMSE scores of 15.9 and 16.3, MAE scores of 11.9 and 11.6, and MAD scores of 9.7 and 8.8. As seen, these values are extremely close to one another, and also somewhat poor. We also created LS and LAD models using the entire dataset for predictors (except for slow_order, as that was engineered from the delivery_duration_min column), however there were extremely small improvements. Because of the downsides of a more complicated model like that (mainly overfitting, overcomplication, and loss of interpretability), we decided to stick with the LS model with only 10 predictors. We also applied Ridge regression to the data to see if that would affect anything, but it only caused similar results to above and did not change much.

From this, we were able to learn that the data is most likely nonlinear and the variables delivery_duration_min is not entirely correlated to the others. We can see that linear regression is not the best direction to go in predicting the slowness of an order, which means we should turn to classification techniques. We also were able to discover the variables that have the most correlation with delivery_duration_min, which can help in later model buildings. Some of these variables included estimated_store_to_consumer_driving_duration, subtotal, and total_outstanding_orders. Overall, by applying linear regression to our dataset to predict delivery_duration_min, we were unsuccessful in creating an accurate model, but we can use our findings to build classification models, as discussed below.

iv. Logistic Regression Analysis

Logistic Regression was applied in our project to predict whether an order would be considered slow, which was defined as having a delivery duration above the 66th percentile (approximately 51 minutes). Logistic regression was applied using all available predictors after preprocessing which would show, after scaling, what predictors the model deemed important. The delivery_duration_min variable, which leaks data as it provides direct information about the true

delivery time, was removed. Further, 5-fold Cross-Validation was applied to the training data to choose from four different logistic regression models: no regularization, L2 regularization, L1 regularization, or Elastic net regularization, which is a combination of both L1 and L2 penalties. Metrics that were looked at included accuracy, precision, recall, and F1-Score. Overall, the cross-validated accuracy, precision, recall, and F1-score were essentially identical. This indicates that the regularization type has minimal impact on model performance for our dataset, likely because most predictors are not highly collinear (as noted in our EDA/Data Preprocessing) and the model is already relatively stable. The uniformly low average recall and F1-scores (0.411 and 0.508, respectively) are consistent with the dataset's class imbalance, which in this case shows that a default threshold of 0.5 tends to under-predict the minority class. Alternate regularization strengths were not explored because the nearly identical performance across all penalty types suggests that regularization is not a limiting factor in this problem. The final logistic regression model applied an L2 penalty, since it is stable, avoids complexity that Elastic net adds, and acts as a safeguard to ensure that coefficients are stable and not inflated. A threshold of 0.34 was applied to the data since it is imbalanced, and thus the positive-class prevalence was used as our decision threshold. Overall, the model achieved a test accuracy of 69%, as well as an improved recall of 0.667. With an AUC of approximately 0.756, the model shows that it has a moderate ability to discriminate between slow and non-slow deliveries. Since our test accuracy is only a small improvement over the baseline accuracy (implied by predicting the majority class), this shows that the linear boundary is only partly effective in capturing our data, and thus, a more complex/flexible model might be able to perform better.

In terms of feature importance (magnitude of the scaled coefficients), total outstanding orders as well as total on-shift dashers are by far the most important predictors. Both of these make sense, since a high number of outstanding orders reflects that the system is currently slow and thus limits order fulfillments, whereas having more dashers available can directly reduce delivery delays by increasing capacity. Additionally, other features that dominate include estimated store-to-consumer driving duration, subtotal, total busy dashers, and estimated order place duration. These are intuitive as well because longer travel distances and larger/complex orders both naturally increase the overall time required for delivery. However, it seems that predictors such as the maximum and minimum item price, as well as most market IDs and order protocols, do not significantly affect our model and therefore likely contribute little to no predictive value beyond the features above (that are related to operation or workload).

v. Random Forest and Classification

Because we found that our Random Forest model had the highest accuracy and most generalizable/interpretable, we chose to dive into this part deeply in our main report. Therefore, to read about our methods and results for this, please look at the main report, which is above.

vi. PCA and Clustering Analysis

Before applying PCA and clustering, we implemented a preprocessing pipeline consisting of median imputation and standardization. Median imputation was used to handle any remaining missing values because the median is robust to outliers and appropriate for the skewed

distributions. This approach was consistent with the preprocessing strategy used in our Linear Regression and Random Forest models. Standardization using StandardScaler was essential for both PCA and K-Means clustering because these algorithms are sensitive to feature scales. Without standardization, features with larger numeric ranges would dominate the principal components and distance calculations, leading to biased results.

We applied PCA to all features after removing the target variable (slow_order) and delivery_duration_min to prevent data leakage. The analysis revealed that the first principal component explained only 15.47% of the total variance, with every following component explaining smaller and smaller portions (9.90%, 8.98%, 6.57%, and 5.89% for PC2 through PC5). To capture 80% of cumulative variance, 13 principal components were required, while 90% variance required 15 components and 95% required 17 components. The scree plot demonstrated a gradual, steady decline in variance explained rather than a sharp elbow, indicating that no single dominant pattern exists in the data. This finding suggests that our features are diverse and non-redundant, which validates our decision to use all available features in the Random Forest classifier rather than applying dimensionality reduction. The relatively even distribution of variance across components explains why ensemble methods like Random Forest achieved strong performance (72.5% accuracy) compared to linear methods that struggle with high-dimensional data.

K-Means clustering was applied to the same standardized features to discover groupings in delivery characteristics. We evaluated cluster counts from k=2 to k=10 using both the elbow method (inertia) and silhouette analysis. The elbow plot showed a notable decrease in inertia from k=2 to k=4, with diminishing returns beyond k=4. Silhouette scores remained relatively stable across all k values, ranging from 0.113 to 0.142, indicating that variance is distributed across many dimensions. Based on interpretability and the elbow analysis, we selected k=4 as the optimal hyperparameter for this situation. Using k=4, the final K-Means model identified four distinct clusters. Cluster 0 (n=14,311) represented low to moderate demand scenarios with 31.5 average outstanding orders, 27.1 on-shift dashers, and a 33.2% slow order rate. Cluster 1 (n=64,576) captures conditions with 34.5 outstanding orders, 29.8 on-shift dashers, and the lowest slow rate at 28.5%, suggesting this represents periods where capacity closely matches demand. Cluster 2 (n=38,662) clearly identifies high-demand periods with 126.7 outstanding orders, 90.1 on-shift dashers, 82.9 busy dashers, and the highest slow rate at 43.2%. Finally, Cluster 3 (n=39,03) shows lower demand with 30.6 outstanding orders, 25.9 on-shift dashers, and a moderate slow rate of 37.2%. The clustering analysis validates findings from our supervised learning models.

The features that drove cluster separation (total_outstanding_orders, total_onshift_dashers, total_busy_dashers) match with the features Random Forest identified as most important through feature importance analysis. Additionally, the clear difference in slow_order rates across clusters demonstrates that these groupings align with our classification target, even though the clustering algorithm never saw the slow_order labels during training. Overall, between PCA and clustering, K-Means clustering was more informative for this problem. PCA confirmed that our

feature set is non-redundant and justified our approach of modeling with all features, but clustering delivered clearer insights.

vii. Neural Network Analysis

The Neural Network section of our notebook focused on developing and evaluating a binary classification model to predict whether an order should be labeled as “slow.” The analysis began with data preparation, where the preprocessed train_df_doordash.csv and test_df_doordash.csv were created and loaded. Once again, the delivery_duration_min column was intentionally removed to prevent data leakage as it directly correlates with the target variable slow_order. All remaining features were standardized using StandardScaler and converted into PyTorch tensors to support efficient model training. The neural network architecture implemented as SimpleNN, consisted of a single hidden linear layer with a ReLU Activation and an output layer with a sigmoid activation, appropriate for binary classification. Hyperparameter tuning was conducted through a grid search evaluating combinations of hidden layer sizes (8, 16), learning rates (0.01, 0.001), batch sizes (32, 64), and epoch counts (5, 10), using the Adam optimizer throughout. The best-performing configuration achieved an accuracy of approximately 0.7364, corresponding to 16 hidden units, a learning rate of 0.001, a batch size of 32, and 5 initial epochs.

After identifying the optimal hyperparameters, the model was retrained with a significantly increased training budget of up to 5000 epochs and equipped with early stopping using a patience of 50 to avoid overfitting. Training terminated at epoch 482, indicating successful convergence before reaching the maximum epoch limit. Final evaluation on the test set produced an accuracy of 0.7383, showing consistent generalization performance. Several visual diagnostics supported the evaluation: the training loss curve demonstrated a smooth and steady decline before flattening, the training accuracy curve showed clear improvement over time as well. The ROC curve yielded an AUC of approximately 0.803, signifying strong separation between slow and non slow orders, and the confusion matrix provided further insight into prediction distribution across true and false classifications. Overall, the NN supported by proper preprocessing, systematic hyperparameter tuning, and controlled training achieved a solid performance and demonstrated meaningful predictive capability for identifying slow delivery orders.

viii. Hyperparameter Tuning

Within the linear regression analysis, we used Ridge Regression for regularization. By using 10-fold cross validation, we were able to find the optimal penalty term to be 0.73, which helped (very) minorly increase accuracy in the data, however the subsequent increase of accuracy was nothing of significance.

Within the random forest analysis, hyperparameter tuning was done through a random search, evaluating the optimal number of estimators, maximum tree depth, minimum samples per split, minimum samples per leaf, and the number of features considered at each split. Although RandomizedSearchCV is less thorough than GridSearchCV, it is often more than enough and is

much less computationally expensive. The best hyperparameters found were 200 estimators, a minimum samples split of 10, a minimum samples leaf of 5, a maximum depth of 20, and sqrt for max features. These hyperparameters were chosen by random search from a specific search space designed to prevent overfitting since previous options (such as max_depth being None) allowed for less model generalizability. Hyperparameter tuning through this approach improved model accuracy while also preventing overfitting.

Within the neural network analysis, hyperparameter tuning was conducted through a grid search evaluating combinations of hidden layer sizes (8, 16), learning rates (0.01, 0.001), batch sizes (32, 64), and epoch counts (5, 10), using the Adam optimizer throughout. The best-performing configuration achieved an accuracy of approximately 0.7364, corresponding to 16 hidden units, a learning rate of 0.001, a batch size of 32, and 5 initial epochs.