Team Members: Zane LaBute, Impana Chimmalagi, Achal Singh, Siddharth Sudhir, Aryan Singh

Main Document

Github: https://github.com/zlabute/Data_Dawgs_DoorDash_Project

I - Explanation of the Dataset

The dataset we worked on was the Kaggle DoorDash ETA predictions dataset. It contained a subset of orders from early 2015 and has details/information like order details, restaurant information, conditions, all aimed at helping predict delivery times.

These are the features we ended up using as our predictor variables in the neural network: 'total_items', 'num_distinct_items', 'Busyness Percentage', 'subtotal', 'total_outstanding_orders'. The response variable was 'delivery_time_minutes'. All of these were provided within the dataset, with the exception of Busyness Percentage, which was calculated by:

$$Busyness\ Percentage\ = \frac{Number\ of\ busy\ Dashers}{Number\ of\ total\ available\ Dashers}$$

II - Overview of the Problem

The problem that we addressed was how to accurately predict the delivery time of a DoorDash order given a set of features relating to available Dashers as well as order details including prices, items etc. This sort of problem would have major applications for delivery services, who would be able to provide a lot more accurate estimates to their customers with more advanced regression and classification pipelines.

III - Our Choice of Methodology

For the task of predicting delivery times based on driver business, total/distinct items, subtotal etc. after trying a variety of methods from linear regression, various types of classifiers, we decided to settle on a deep neural network with 3 hidden layers to produce a probabilistic evaluation of whether the delivery would take a short, medium or long amount of time.

This was certainly the most optimal one, as regression models such as multi-linear regression weren't able to capture the complexity of the problem and latch onto the patterns of the large dataset. This can be seen in our appendix, where the linear regression model had a MSE of ~ 204 minutes squared. With our neural network, this was reduced to ~ 33 minutes squared.

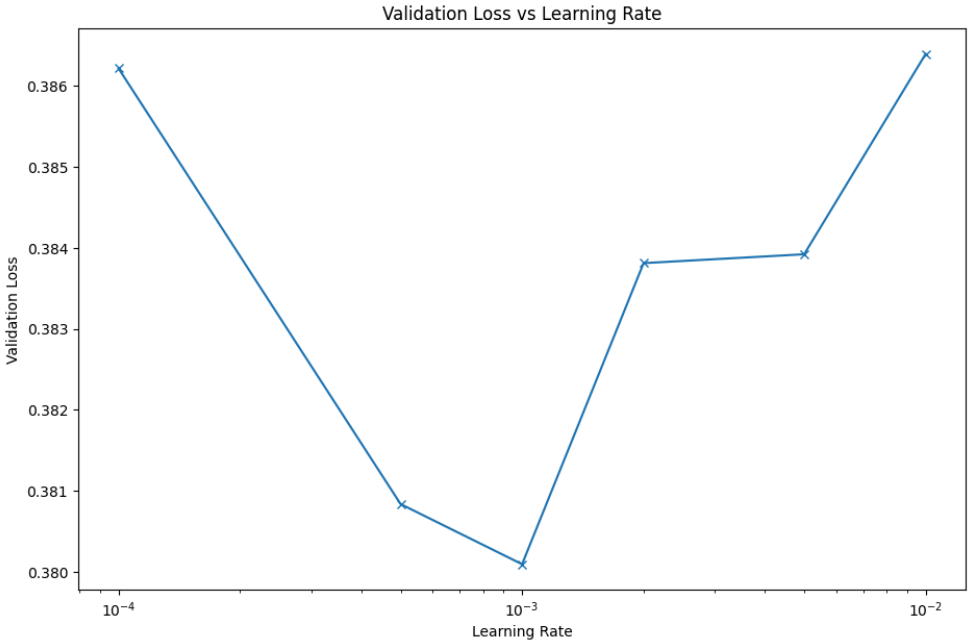
Classification via logistic regression wasn't very optimal for our problem as well, since with such a large dataset that included a good amount of noise, it was tough to predict exactly where a delivery time would lie between 3 classes, but wasn't that difficult to say if it was over/under a set value for example 25 minutes. While this gave us some promising results (with an accuracy of ~ 80%), it still wasn't fit for the problem that we were tackling.

Ultimately, we ended up experimenting a lot with neural networks to find the right one for the problem and after using them for regression purposes, we decided to make a categorical classification

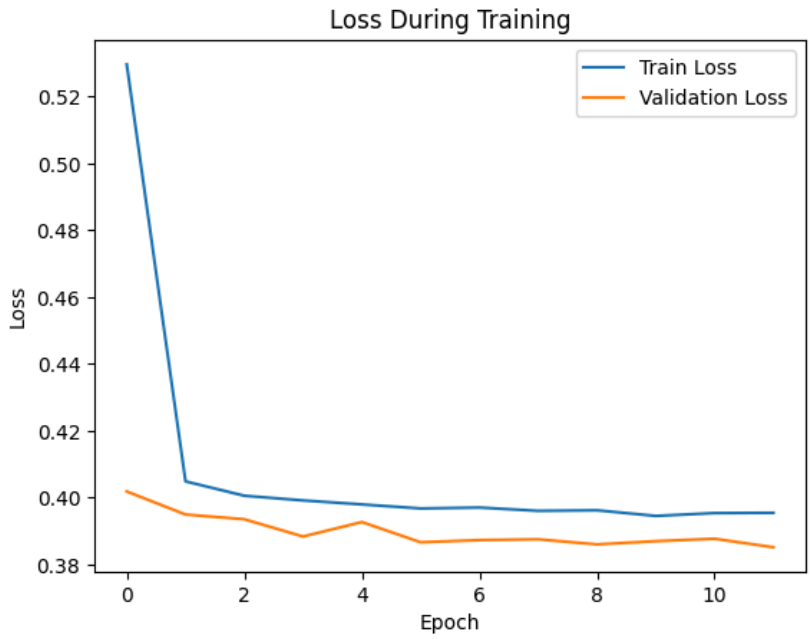
network, which would output probabilities of whether the delivery would be short / medium / long. This model certainly worked best with our research question of predicting which category the delivery would fall under, and yielded extremely good results with a MAE of 2 minutes, meaning that on average its prediction would only be off by 2 minutes which by DoorDash standards is extremely accurate.

IV - Results, Performance Metrics and Conclusions

5-fold cross-validation was used to assess the model's performance and generalizability, and the model was trained for 20 epochs.



The validation loss reaches its minimum at a learning rate of approximately 10^-3. Additionally, the graph displays a U-shaped curve, indicating that both very low (10^-4) and very high (10^-2) learning rates result in higher validation loss.



Training loss decreases significantly in the first few epochs. After about 2-3 epochs, both losses stabilize, with slight fluctuations. The validation loss remains slightly lower than the training loss throughout the training process. This graph indicates that the model is learning effectively, with both training and validation losses decreasing and stabilizing. The close tracking of validation loss to training loss suggests good generalization, and since the validation loss doesn't decrease significantly over the epochs, there's no clear sign of overfitting.

Fold	Accuracy	Loss	Val Accuracy	Val Loss	Precision	Recall	F1 Score	MAE	MSE
1	0.8127	1.1165	0.8580	0.3946	0.7363	0.8580	0.7925	2.15	32.99
2	0.7921	1.1710	0.8608	0.3955	0.7410	0.8608	0.7965	2.11	32.33
3	0.8157	1.1134	0.8623	0.3923	0.7776	0.8623	0.7986	2.09	32.30
4	0.8274	1.0607	0.8577	0.3970	0.7357	0.8577	0.7920	2.16	33.27
5	0.8163	1.0883	0.8583	0.3963	0.7369	0.8583	0.7930	2.15	33.20

The average precision across all 5 folds is 0.7387, showing that approximately 73.87% of the positive predictions made by the model are correct. The average recall is 0.8594, meaning that the model correctly identifies about 85.94% of all observed positive instances. The average F1 score is 0.7945, showing a balanced measure of precision and recall.

The average MAE is 2.13, indicating the average absolute difference between predicted and actual values. The average MSE is 32.81, providing a measure of the average squared difference between predicted and actual values. Since the absolute error is around 2 minutes and the root squared error is around 6 minutes, this overall does not create a significant effect when predicting delivery time.

Additionally, the metrics improve during each fold. The high recall, balanced precision, consistency, show that the neural network can capture the complexity of the problem.

As for limitations: the model has a high average recall of 0.8595, its precision is lower at 0.7387. This indicates that the model is more inclined to predict positive instances, potentially leading to more false positives, which does not indicate a perfect balance between precision and recall. Also, the MAE and MSE values may not be significant for many scenarios, it could be crucial for time-sensitive deliveries. Additionally, neural networks are more computationally expensive compared to regression models, which could have also helped to answer this problem. Additionally, since neural networks are more of a black box model, they aren't as interpretable as other models that could have been used.

V - Instructions for Replicating our Process

In order to run the code for our project, the following steps are needed:

1. Open our notebook .ipnyb file within a google colab notebook in order to use the correct environment.

2. From the top menu under **Runtime** select **Run all**.
3. Give the notebook ~10 minutes to generate all results

Note: To run this project you do not need to install the kaggle Doordash dataset, as our implementation automatically loads and updates the dataset from kaggle.

Appendix

Exploratory Data Analysis

The distribution for Total Items and Delivery Time was graphed using histograms. Additionally, outliers were removed using the IQR method, and the Delivery Time distribution was also graphed without regarding outliers. We also graphed other features using these histograms, but these two features displayed the most usable visualizations.

Additionally, we utilized scatterplots to graph Number of Distinct Items vs. Delivery time, as well as Total Items vs. Delivery Time.

From there, we filtered the dataset further on into a set of few features: 'total_items', 'num_distinct_items', 'total_onshift_dashers', 'total_busy_dashers', 'delivery_time_minutes'.

The last feature is the response variable, and we split the dataset vertically into an X and y set (for the predictor variables and the response variable). From there, the dataset was split horizontally into a training set and a test set using the train_test_split function from the scikit-learn library. This randomly samples the data and allocates 80% of the data for the training set, and the other 20% of the data for the test set.

Data Pre-processing and Feature Engineering

Before processing our dataset, we searched for all the missing values in the important features such as total_items', 'num_distinct_items' 'total_onshift_dashers', 'total_busy_dashers', and 'delivery_time_minutes'. Observations with missing values were removed to maintain the data, and since the missing value proportion was below 10% for each of these relevant features, removing the missing data points was a reasonable approach.

We also created a new feature called, "delivery_time_minutes" which calculated the difference between the 'created at' label, and 'actual_delivery_time' which helped us normalize our results. Additionally, we converted other features to minutes and datetime types wherever possible to ensure additional ease when accessing/utilizing data.

To ensure the model's robustness, outliers in 'delivery_time_minutes' were identified using the Interquartile Range (IQR) method. Deliveries with durations exceeding 50 minutes were considered outliers and filtered out.

Regression Analysis

We utilized regression analysis to predict **delivery_time** using the independent variables of **total_items**, **num_distinct_items**, **total_onshift_dashers**, and **total_busy_dashers**. Our linear regression produced low R^2 scores of 0.04 for both training and validation sets. This told us that these predictors explained only a small fraction of the variability in delivery times and that the linear model may struggle to capture the nonlinear nature of the data's relationship with **delivery_time**.

Output of our regression model:

- Training MSE: 204.23, Validation MSE: 204.62
- Training R^2: 0.04, Validation R^2: 0.04

Ridge regression with a hyperparameter alpha of 1.0 was employed to address the potential collinearity of predictors, but the result was similar to linear regression. This helped us to determine that the lack of accuracy came from a lack of predictive power from the model rather than overfitting.

Output of our regression model with ridge regression:

- Ridge - Training MSE: 204.23, Validation MSE: 204.62
- Ridge - Training R^2: 0.04, Validation R^2: 0.04

Via further analysis of single-feature regression, we confirmed that there was a low correlation between individual predictor and delivery time. These derived features, such as busyness percentage (the ratio of busy to onshift dashers) were introduced to capture nuanced effects but showed little improvement in the model's accuracy or evaluation metrics.

Our conclusion from this portion of experimentation with the doordash dataset was that in order to accomplish our goal of predicting **delivery_time** accurately we would need to use a form of non-linear prediction. Given the lack of direct linear correlation between independent variables and the desired response variable, we realized if we were to form a predictive model we would need to utilize the non-linear relationships that existed within the data.

Logistic Regression Analysis

For our dataset, we used logistic regression for a binary classification task, more specifically to predict whether a delivery would take less than 30 minutes or more.

We filtered the data set to include only samples that had a less than 50 minute delivery time. We did this because any delivery taking over 50 mins, could have been considered an outlier caused by some very specific/unlikely circumstance. Additionally, our goal was to predict whether a delivery would take 30 minutes or long within a reasonable window. By focusing on the deliveries under 50 minutes, the model becomes far more specialized in predicting within this range. Including the much longer delivery time samples would just introduce noise and make it hard for the model to learn patterns with typical delivery times.

We created a new feature that we called the “Busy-ness Percentage” which basically was calculated by taking the ratio of busy drivers/dashers with the total number of drivers available. However, we ended up not using it to train our Logistic Regression. Then we also had our delivery_time_binary variable indicating whether the delivery took >30 mins or not. We also cleaned the data set with all rows

with missing values removed, followed by splitting up this cleaned data set into training and testing groups. Then while training our Logistic Regression model finally, we used these 4 features as predictors: total items, distinct items, the subtotal, and total outstanding orders.

Once we trained our models we see that we have the following statistics:

- Prediction Accuracy: 0.7931
- True Positive (Recall): 0.9999
- True Negative (Specificity): .00016

This means that the model was able to put 79.3% of predictions in the right classification. Though this might seem high, it is important to consider Recall and Specificity, to draw conclusions. A high recall and low specificity indicates that our model does extremely well to classify when the delivery time is over 30 mins but performs poorly when looking at the negative result. So, it is biased towards the prediction of the positive result.

In order to fix this, we may need a vaster data set with more points that have a < 30 min delivery time, or just resample the existing data set better. Looking at the AUC of .66, it also indicates that perhaps we need to consider a new threshold, and since the dataset may be biased, a different model may work better to predict delivery time with the complexity of the data we have.

KNNs

We used KNN with 3 neighbors to prevent underfitting and overfitting of the data, and the categorical response variable used is 'delivery_time_binary'. This is the data of deliveries with delivery_time under 50 minutes, and then further if the delivery took under 30 minutes or over 30 minutes within this range. We can also experiment with a different number of neighbors later on to determine which one will yield the best results.

Additionally, we scaled the data before fitting the model. This is because scaling ensures all features contribute equally to distance computations and prevents features with large magnitudes dominating weightages. Cross-validation with 5 splits was also performed on the model.

The KNN had a high true positive rate, since it has a recall of 94.9%. The model excels at identifying positive cases. However, the true negative rate of 43% suggests the model struggles with correctly identifying negative cases.

The F1 score of 0.905 indicates a balance between precision and recall.

The ROC curve shows performance above the random classifier with an AUC of 0.66. However, the cross-validation AUC of 0.66 indicates only moderate discriminative ability, since an AUC closer to 1 would indicate better ability to distinguish classes.

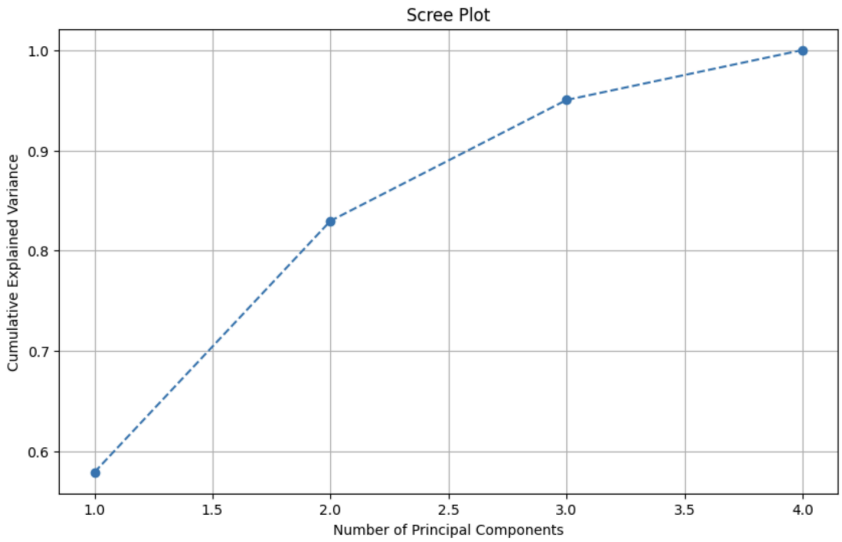
Additionally, although both the cross-validation and single-split accuracy for the model are relatively high, the cross-validation accuracy of 79.3% is slightly lower than the single-split accuracy,

suggesting some overfitting. Overall, the KNN model might not perform as well when generalizing to new, unseen data.

We used KNN as opposed to decision trees or random forest for classification because for large datasets of delivery time information like this one, random forests may require more computational resources and time to train compared to KNN. Also, decision trees can create complex decision boundaries, which we believed was not necessary when dealing with classification of orders with their delivery times. KNNs can also adapt to new data more easily compared to a decision tree, which might require a rebuild of the model for adequate performance.

PCA

We did use PCA to work with our data set in order to reduce dimensionality and single out the most important features for us, and the set of features we used was the same 4 features that we had when performing Logistic Regression and also in KNN.



Once the PCA was performed and the Scree Plot was drawn up, it was clever that there were two principal components contributing to around 83% of the variance.This suggests that most of the dataset's variance can be captured with only two components, making it suitable for dimensionality reduction without significant loss of information. Adding further components beyond this is basically diminishing returns for us as the variance only increases marginally.

Principal Component 1:

- num_distinct_items: 0.6091
- total_items: 0.5795
- subtotal: 0.5364
- total_outstanding_orders: 0.0738

Principal Component 2:

- total_outstanding_orders: 0.9883
- total_items: 0.1135
- num_distinct_items: 0.0739
- subtotal: 0.0705

Printing out the two main PCs and studying the weights for each feature gives us a lot of insight. From PC1, we can see that the first three features have similar weight and so are correlated. Out of the three, subtotal has the lowest weight. In PC2, those three have low weight while outstanding orders give the most new information. Thus, subtotal having low weights in both and likely being correlated to total items or distinct items, could be removed.

Neural Networks

This was discussed in the main document.

Hyperparameter Tuning

In training our neural network, one of the biggest tasks that would have a large impact on the results was selecting a learning rate. While we could have just selected one at random, we decided to use the methodology discussed in class where we experiment with varying magnitudes, to find the optimal learning rate of ½ * at which the loss starts increasing again. This was also further discussed in the cross-validation and metrics section, in which we saw, as expected, a U-shaped curve when modelling our learning rate vs the validation loss, suggesting that we should pick something in that minima to ensure that we minimize the errors but also maintain a generalizable model.

We also had to decide on the complexity of the neural network, which we were able to do by picking how many neurons we wanted per layer and the number of hidden layers in the model itself. The funnel-based model of going from 128 to 64 to 32 neurons in each layer provides a robust method of feature extraction where the model is able to identify patterns a lot more seamlessly than fit to the noise. ReLU activation functions provide faster convergence as well as prevent the vanishing gradient problem. By using dropout with a 30% dropout rate as well as L2-regularization on the weights, we prevent overfitting due to co-adaptation as well as to the noise in the data by limiting the neuron weights. In the output neurons, we have a softmax activation function to scale the outputs down to ensure they sum up to 1, and result in 3 neurons indicating the probabilities of the delivery belonging in either category.