Jason Dean
DS 450 Final Project
Instacart Customer Grocery Basket Prediction
Code: https://github.com/JTDean123

Objective and Project Overview

The goal of this project was to predict the ordered items in grocery baskets of new Instacart orders. The data from this project is originally from a Kaggle competition and the scope and objective were changed slightly for this project. Here we evaluate model performance using the F1 score.

For this project we were supplied three sets of customer order data sets (prior, training, and test), each containing order id, user id, order number, order dow, order hour of day, and days since prior order. Additionally, we were provided tabular data containing product information. A departments table contained department id and a description of department (ex: 3, bakery), an aisles table contained aisle id and a description (ex: 2, specialty cheeses), a products table contained product id, product name, aisle id, and department id (ex: 2, all seasons salt, 104, 13), and, importantly, an order table contained information linking order id, user id for the users in the training set (but not the test set!).

Initial Data Processing

The prior users data set contained 1384617 observations and the training and test data sets contained 91846 and 393634 observations, respectively. This data set was in essence a relational database, and my first step was to join the aisles, departments, products, and orders tables with the training and prior data, creating two data frames each containing prior user orders and training user orders with features mentioned above. Check out the jupyter notebook related to this project for more details. After loading and merging the data I moved forward with three data frames:

1) A prior data set containing prior orders for users in both the test and train data sets containing items ordered for each user, order id, and information about the product (aisle, department, etc)
2) A training data set containing orders for user in the training set containing items ordered for each user, order id, and information about the product (aisle, department, etc)
3) A testing data set containing order id, user id, order number, order day of week, order hour of day, and days since prior order.

The challenge in this project was to develop a model to predict the items ordered for each order in the testing data set starting from the four features listed above in 3.

Exploratory Data Analysis

I next embarked on EDA for the training data set to get an understanding of the relationship between the features. As shown in Figure 1, I found that the highest number of orders was placed at 3PM, however peak ordering times were between 9AM and 7PM.

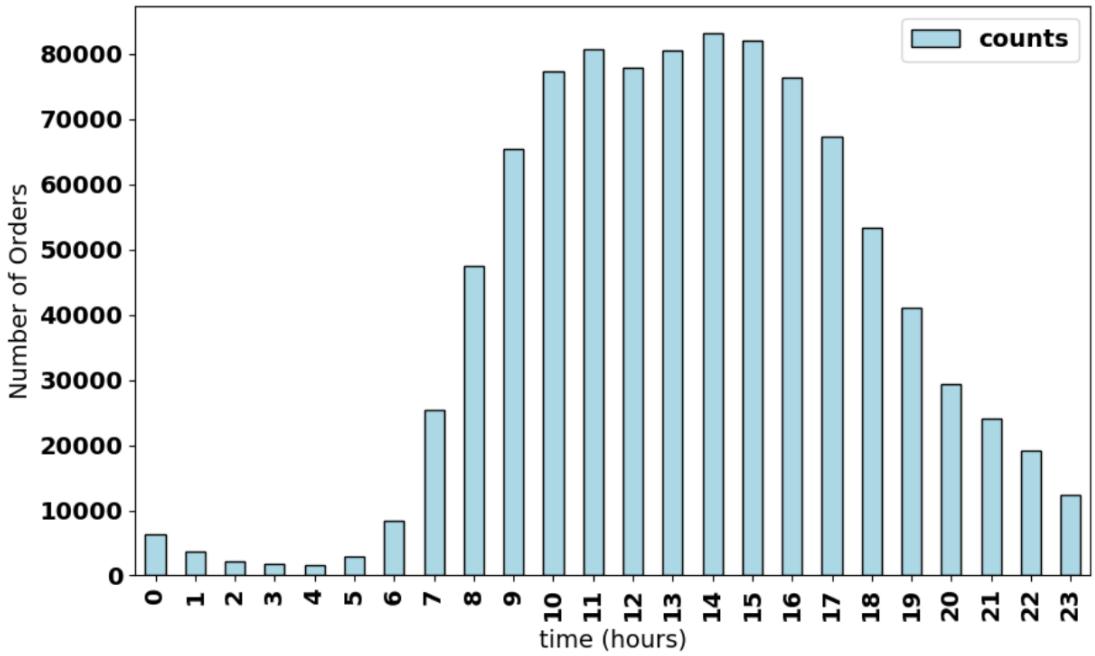

Figure 1. Total number of orders in the training data set for each hour.

Additionally, I found that there was a relationship between the total number of orders placed and the (anonymized) day of the week (Figure 2).

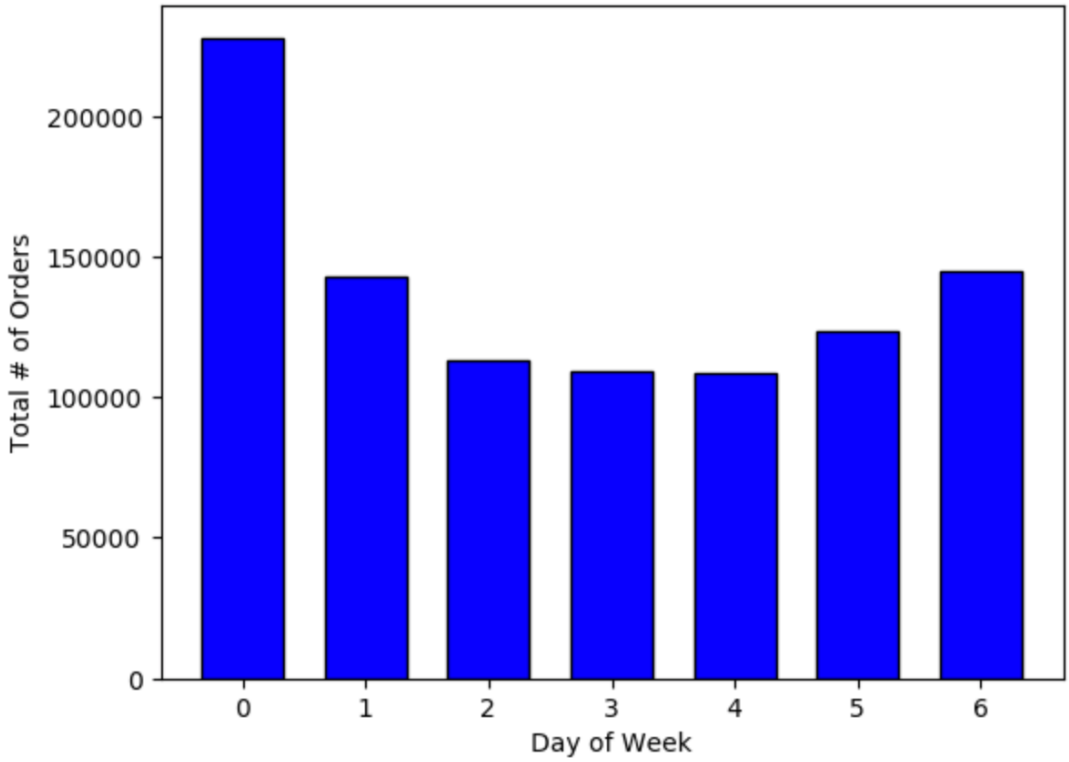

Figure 2. Total number of orders in the training data set for each day.

I next determined the most ordered item at each hour and found that it was bananas, indicating that Instacart was likely hacked by monkeys during the time frame that this data was generated. Total basket size is a feature that is available in the training and prior data sets but not in the test sets, and next I evaluated the importance of basket size on order. I found that the mean basket size was 10.0 (Figure 3).

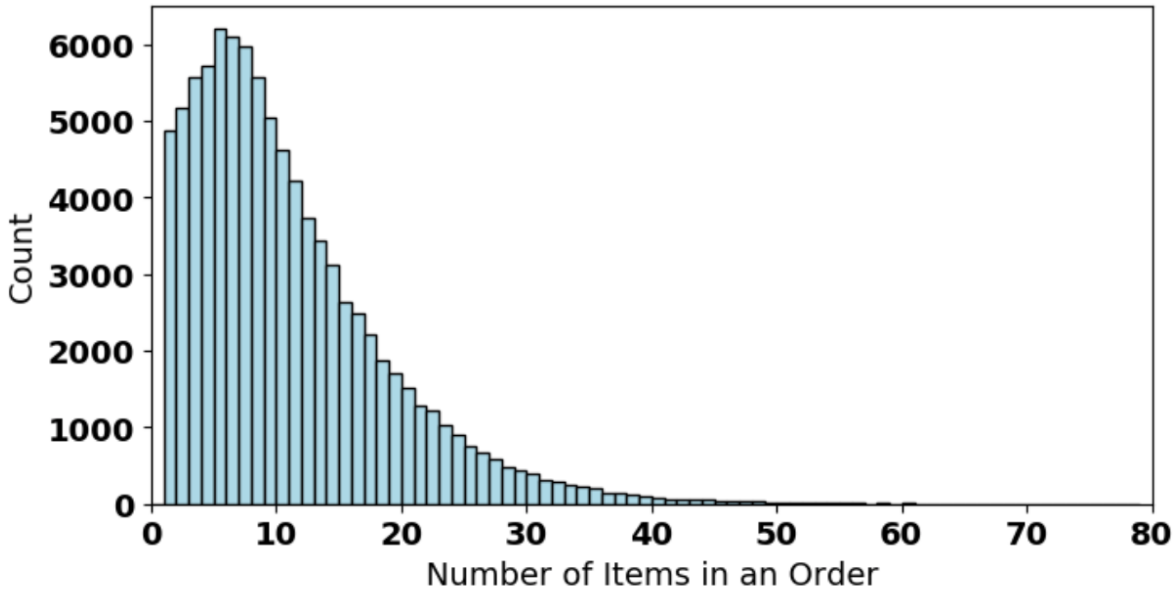

Figure 3. Histogram of number of items in an order for each order.

Additionally, I found that fraction of items from a department was dependent on the basket size (Figure 4).

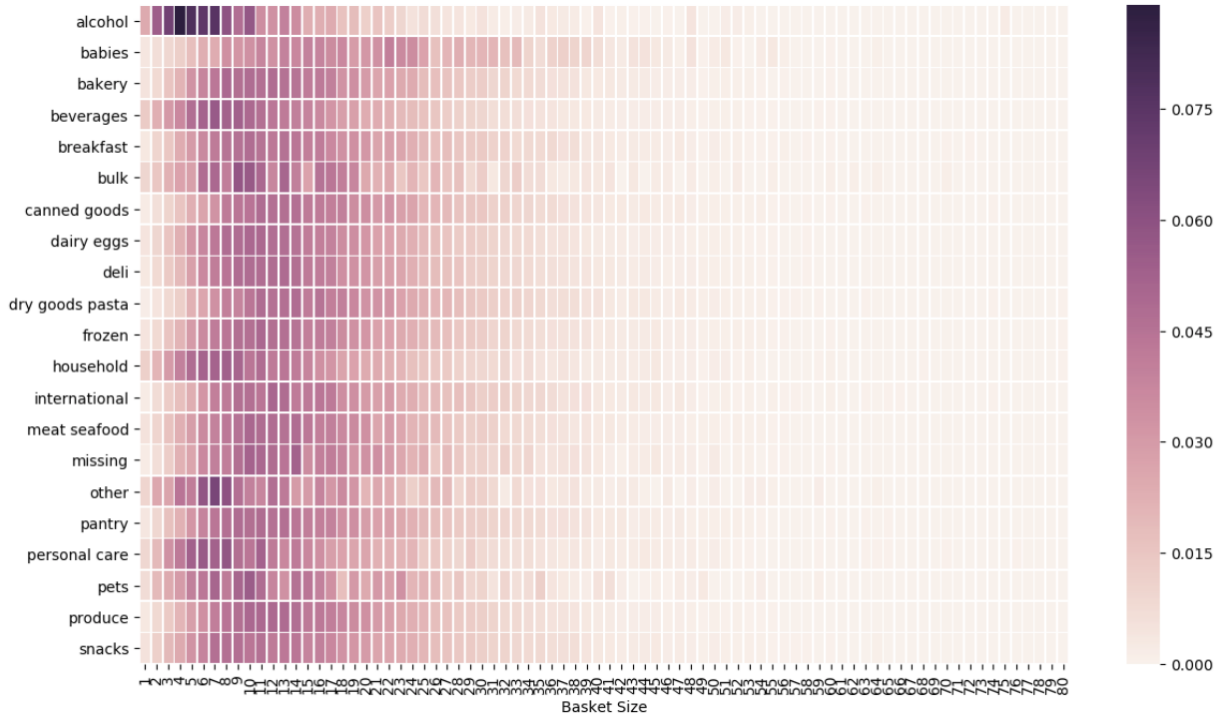

Figure 4. Fraction of total items ordered from a department vs basket size.

As shown in Figure 4, the majority of orders that contain alcohol have a basket size less than six items. Furthermore, the majority of items ordered are from the produce or dairy/eggs department (Figure 5).

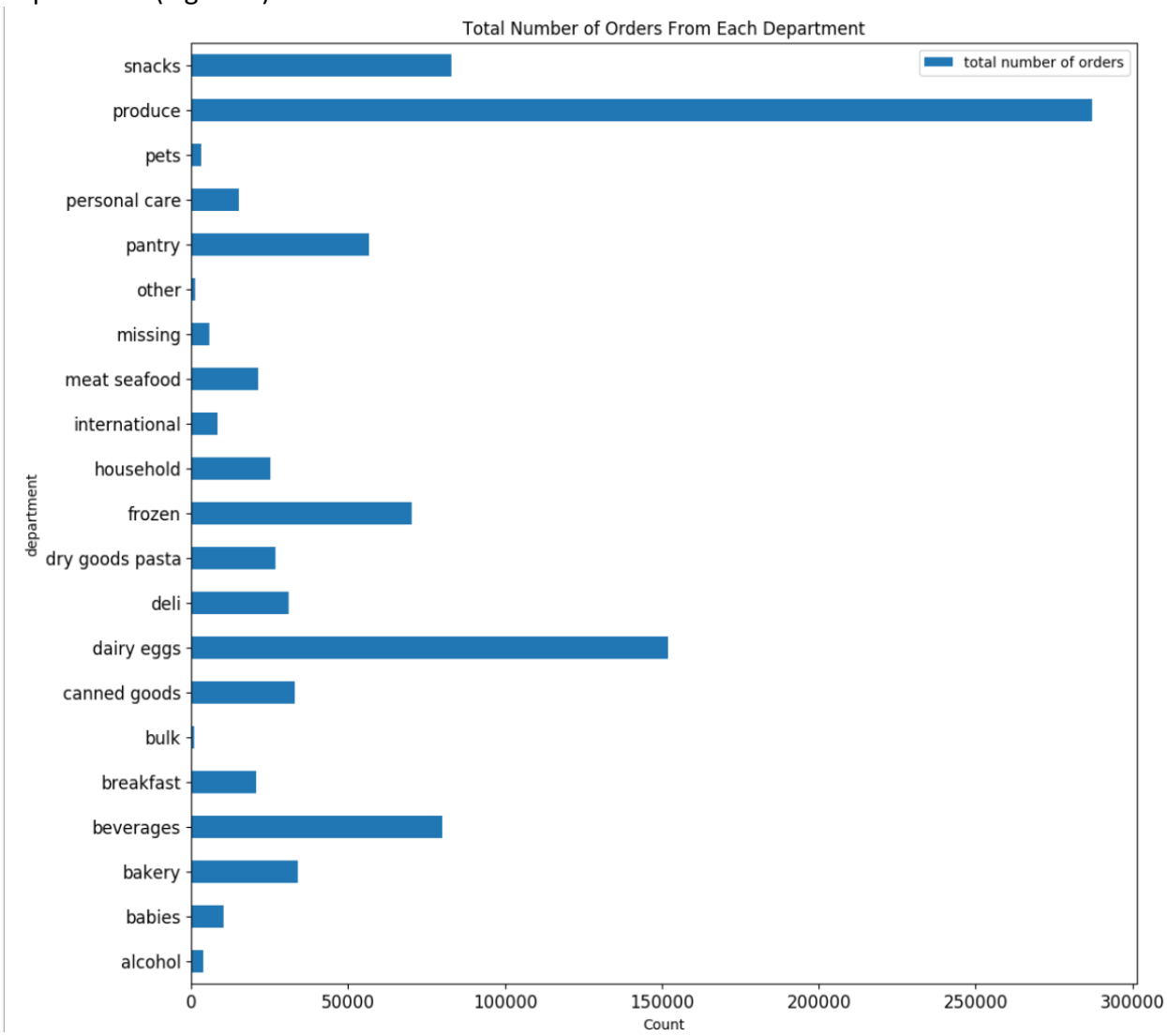

Figure 5. Total number of orders from each department.

This analysis indicated that basket size, order day of week, order time of day, and the department may contain information that can be used to predict if an item is ordered. The mean basket size for each user, as calculated from the prior data set, was added as a feature to the testing data set.

Model Building

This problem was formulated as a binary classification prediction task as follows. First, every item ordered for each user, obtained from the prior dataframe, was added to for each order to both the training and test data set. For example, for a given user and given order in the training data set the items that they were known to order were assigned a value of 1 and the remaining

items 0. In the test set each order, for each user, contained every item that the user has ordered in their history, as determined from the prior data. This strategy obviously generates an unbalanced data set since the majority of the items in an order were not ordered but rather represent unordered items that were ordered in previous orders. I found that after this merging the training data set consisted of 94.8% unordered items, and modeling strategies to deal with this unbalance are detailed below.

I decided to build a feed-forward neural network to predict whether an item was ordered. This model type was chosen not because I felt that a neural network was the best architecture for this project but because neural networks are fun and I hoped to learn more about them. As mentioned above, the goal of this model was to predict if an item was ordered based on the following features:

1) Order number
2) Order day of week (one hot encoded)
3) Order hour of day
4) Days since prior order
5) Basket size (calculated as described above)
6) Aisle ID (not one hot encoded)
7) Department ID (not hot encoded)

Before fitting a neural network to this training data the order day of week feature was one hot encoded. I was able to achieve better test loss by not one hot encoding aisle and department IDs. Next the training data was split into training and test data sets (70|30) and a model was built in Keras with a Tensoflow backend as shown in Figure 6.

```
# weighted model, 200 epochs
class_weight = {0 : 1.,
    1: 20.}

# build a model
modelW5 = Sequential()

modelW5.add(Dropout(0.2, input_shape=(13,)))
modelW5.add(Dense(128, activation='relu', input_dim=13))

modelW5.add(Dropout(0.2))
modelW5.add(Dense(64, activation='relu'))

modelW5.add(Dropout(0.2))
modelW5.add(Dense(32, activation='relu'))

modelW5.add(Dense(8, activation='relu'))

modelW5.add(Dense(1, activation='sigmoid'))

# compile the model
modelW5.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# run!
modelW5.fit(X_train, y_train, epochs=200, batch_size=1000, verbose=1, class_weight=class_weight)
```

Figure 6. Feed forward model architecture to predict if an item was ordered.

As shown in Figure 6, the model consisted of 3 hidden layers, the input layer contained 128 nodes, and each subsequent layer had half of less of the number of nodes as the previous node. The training data was unbalanced, with 95% of the data representing unordered items, and to compensate for this I introduced a class weight of 20 for class 1 (ordered), thus penalizing the model heavily for incorrect predictions on ordreded items. Without this class weighting the model generated a prediction of 0 for every observation. The model was found to generally be insensitive to the number of hidden layers and nodes, although the size of the model was limited by the computing capacity of my laptop. I found that adding dropout layers to the model greatly decreased overfitting in the model and improved its ability to generalize to new data. Additionally, I found that the adam optimizer was superior to sgd with regards to loss during training. Finally, a batch size of 1000 was used for each epoch, and I found that the loss generally converged after ~100 epochs.

Additional fun facts – batch normalization, as well as normalizing continuous variables, did not increase model performance.

In the process of building this model I discovered that parameter tuning a neural network is a bit of a rabbit hole – there are many possibilities and dependencies between model parameters. For this optimization I performed manual tuning, however moving forward I feel the best approach will be to test multiple model architectures in a cluster environment.

As shown in Figure 7, I found that a order probability threshold of 0.8 generated an F1 score of 0.35 on the 30% of the training data that was held out for testing.

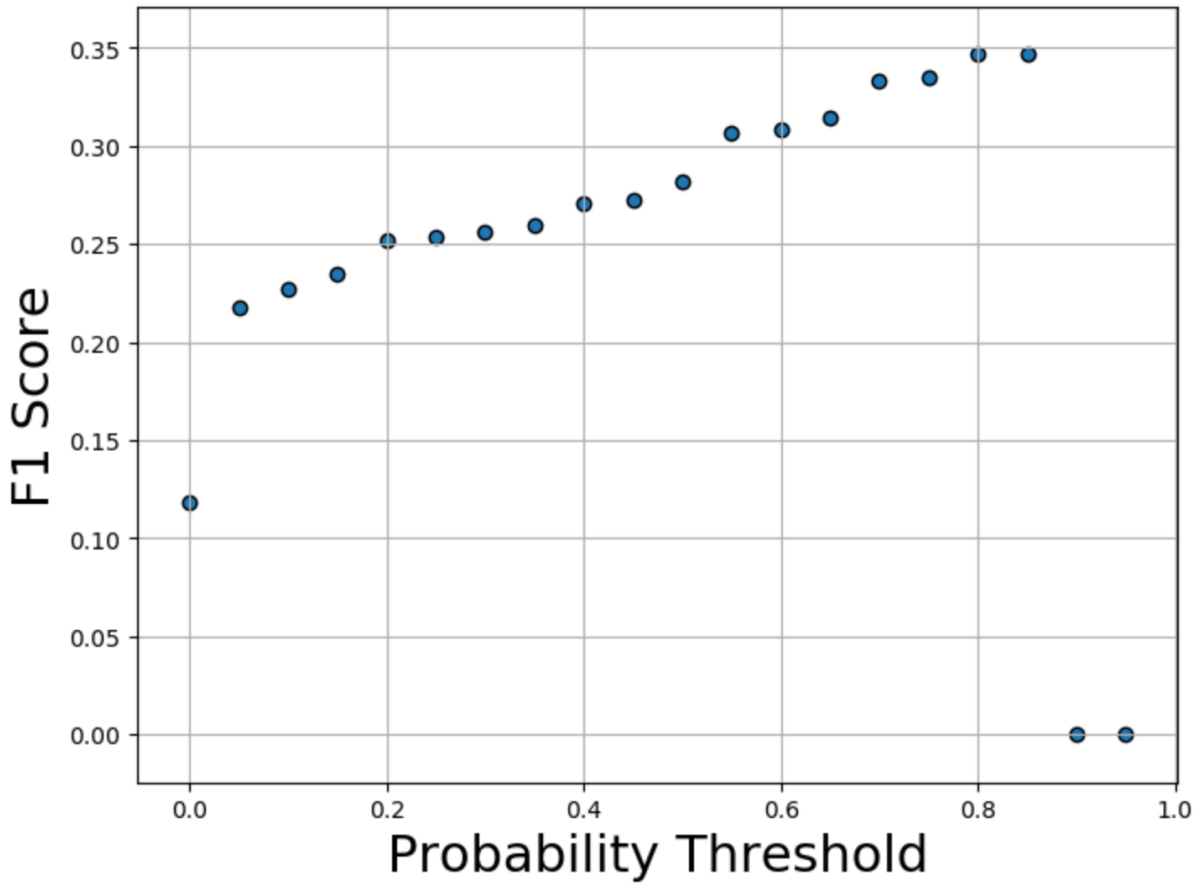

Figure 7. F1 score versus predicted probability of an order for the test data set.

As shown in Figure 7, by altering the probability cutoff of what was called an order we were able to walk the precision/recall tightrope and identify the optimal F1 score, the key metric in this project.

This model was used to predict whether or not an item was ordered in the test data set and the item with the highest probability submitted calculated and submitted for evaluation.

Thanks!