Predicting Customer Habits at the Grocery Store

1. Problem

Businesses would like to know what customers are going to buy in future visits. This knowledge will assist businesses in managing their supply chains to reduce excess inventory, ensuring products are on hand for customers, and boasting customer satisfaction.
This project looks at how data and machine learning can assist a business, grocery stores in particular, in understanding customers, driving sells, and improving customer experiences. Specifically, we are interested in how to predict reorders for a grocery store.

Who might care? Retail businesses, supply chain managers, product marketing agencies, product producers, and individual customers. A business will be better able to manage inventory levels. Products, perishable or fashion dependent, will be fresher or more in vogue and therefore more likely to sell. Customers could benefit from a recommender that makes product suggestions they actually want to purchase. Market agencies could adopt the recommender to find potential undiscovered customers whose shopping habits favored their product. Ultimately, it is a tool that a business can use in supply management and sales boosting.

2. Data

The data obtained from InstaCart contains purchase histories for 206,000 customers with 3.4 million orders. Each order lists the sequence items were bought in; the aisle and department each item is found in; the time of day and day of the week for each order; and days from previous order. There are 50,000 products, 134 aisles, and 21 departments in the data set.

3. The Process

The process began with acquiring, cleaning, and organizing. Once cleaned, we explored the data in order to get a picture of how reorders behaved and looked for any informative trends. Once explored, we used machine learning specifically a Support Vector Machine, a Random Forest, and a Single Vector Decomposition model to predict future reorders. We compared and analyzed the results. The goal was to find insights into customers purchasing habits and future purchases.

Three Juypter notebooks were used. One for the cleaning and exploration called "Capstone_EDA.iypnb". Another one, "SVD.ipynb" made for building and running the Single Vector Decomposition. Lastly, "ML_grocery_basket.ipynb", the main notebook, contained the Support Vector Machine, Random Forest, and final analysis.

4. Data Wrangling and Cleaning

The data, obtained from InstaCart's website, came in the form of six comma separated variable (csv) files. We used Python and Pandas to clean and organize ("wrangle") these files for data management and analysis. Not all the data provided had reorder information available ("labeled"). The supervised machine learning techniques required labeled data, therefore, we discarded data without reorder labels. This prevented us from using the most current customer order and for some the previous order before that one.

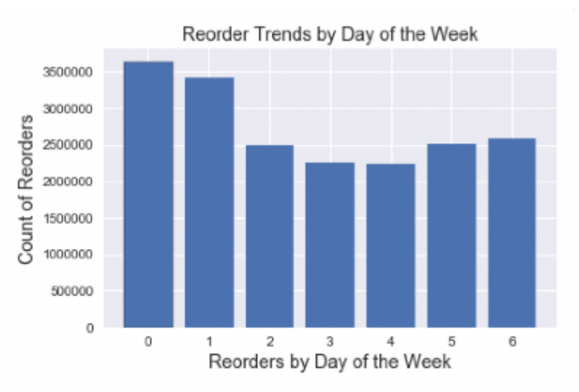
We did find missing data in the "Days since Prior Order" column, which was replaced with zero. This chose caused some information loss. We no longer had access to same day reorders from customers, thus, our analysis only dealt with reorders at least a day apart. The csv files are related by the “order_id” reference number, which we used to combine our data frames into a “tidy format,” an industry standard. Tidy format is one that has observations as rows and columns as features, which is supported by most packages like SciKit Learn. We used the SciKit Learn package for both the Support Vector Machine, and the Random Forest Classification. SciKit Learn has a robust and will tested set of algorithms to support these two models. The Support Vector Machine takes numerical size into account so all the features (columns in the database) were “one-hot-encoded”. One-hot-encoding takes the numerical values in each column and creates a corresponding column for each value adding a one in that column if it applies to the observation (row) or a zero otherwise. This prevents giving more significance to hour 12 over hour 2.

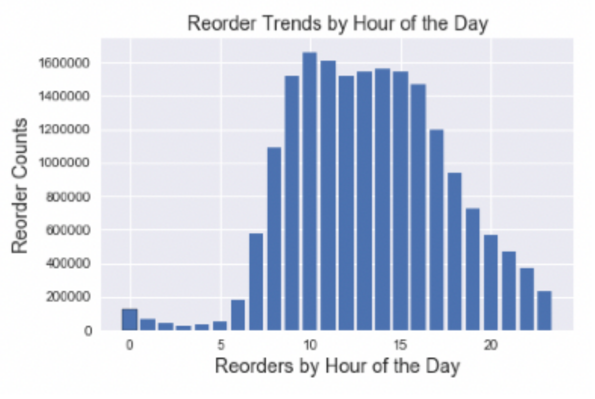
The Single Vector Decomposition was coded by myself in a separate Juypter notebook and imported into the “ML_grocery_basket” notebook for comparison. It is still in “tidy format’ but with user_id as the row observation and product_id as the features (columns). The value entries are the reorder history rate per product for each user_id.

The only other challenge in wrangling was the sheer size of the data. In order to ease computation time, we sampled 1000 customers from the 206 thousand available. The full analysis can be done with an increase in time or expenditure.

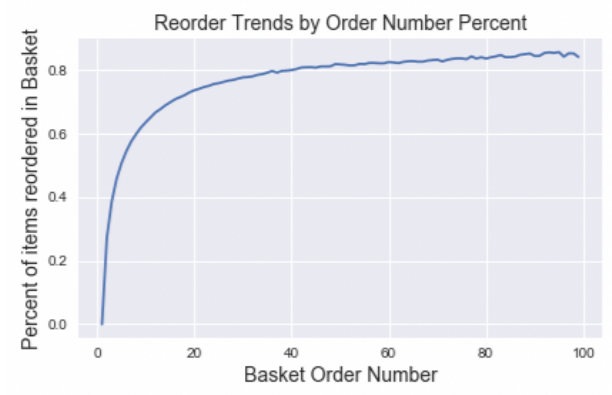
5. Exploration:

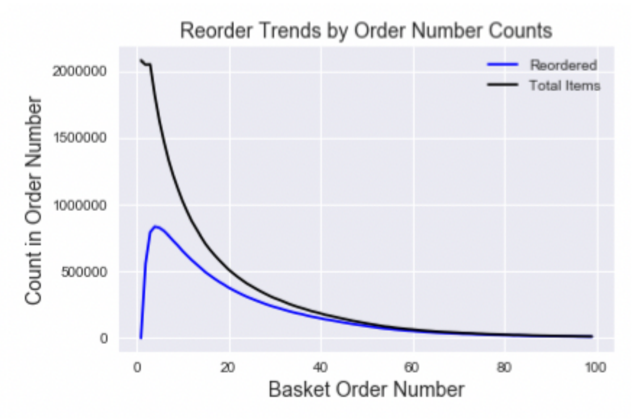
We focused our exploration on what the data told about customer reorder habits. We saw that overall 60% of purchases contain reorders. The first feature “order_dow”, which is the day of the week customers made purchases, told us customers make more purchases and more reorders on "day 0" and “day 1” of the week.



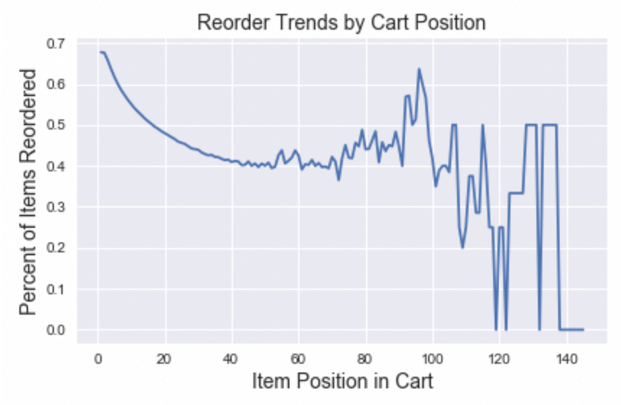


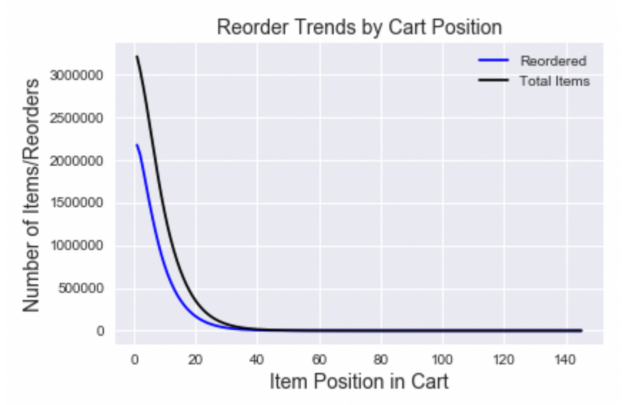
The next feature “order_hour_of_day” shows customers typically place orders reorders between 8 am and 5 pm. The feature “order_number” (graph below) tracks the orders chronologically by customer. We saw that reorders as a percent of the basket increased as customer stayed with the company. The first order, as expected, has no reorders, hence a percent of 0%. Afterward, the percent of reorders in the basket climbed exponentially for each customer leveling off at near 85% (left graph below). An indication that reorders drive sells for continued shoppers. We looked next at the raw values of reorders compared to the basket items, instead of percent, where the reorders climbed shortly and then declined at the same rate as items in the basket decreased (right graph below).



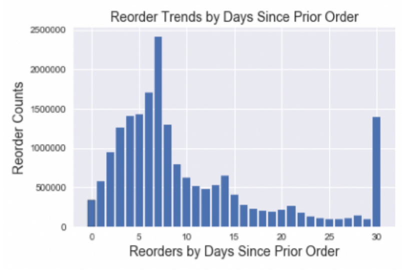


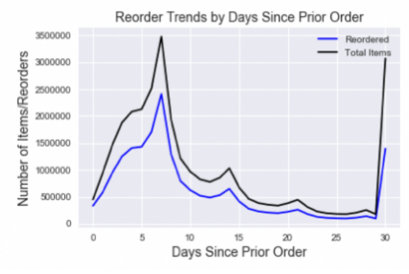
Continuing to the next feature “add to cart order," we answered an interesting question. Do customers buy reorders first or wait until the end? We saw that a customer’s first item added to the cart is a reorder near 70% of the time. Followed closely behind by the second item near the same rate. This begins to trail off to around 40% up to the 55th position in the cart. Afterward, things get a bit erratic, do most likely do to the low number of orders with basket items that large. This is a strong indication that reorders drive customer visits.

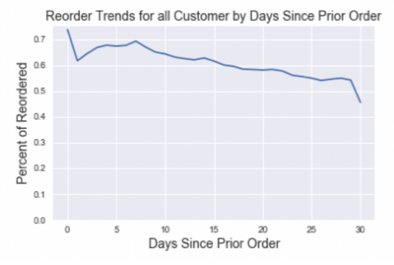




The next feature “days since prior order” gave us a picture that most reorders occurred within the first week of the last order (left graph below). In fact, reorders increased as the week goes on peaking seven days from the previous order after which reorders drop off substantially. The fact that anything over 30 is lumped into the 30th day accounts for the spike at the end of the graph. We see the percent of reorders started at close to 80% and dropped to just above 40% (right graph below). We can make some propositions from this that customers are more likely to reorder items if they make another order within a week of their previous order. Logically, this might be because customers remember more easily things they enjoyed from their last purchase and purchase those items again.

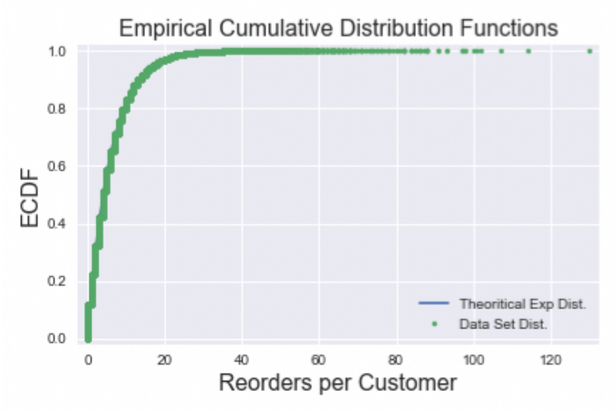


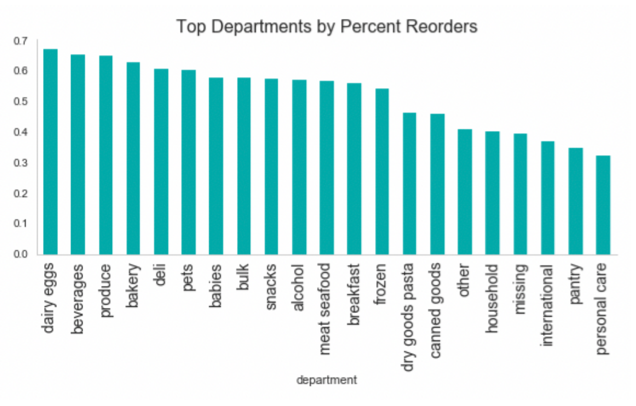




Pulling in some basic statistics by counting up and plotting purchases and reorders, we see a classic exponential decay suggesting an exponential distribution (left graph below).

Plotting these overlaid on one another we see that an exponential distribution does fit the data well.Using the properties of the exponential distribution, some probability questions can be easily answered. We see that 99% of customers reorders at least 28 items, 95% reordered 18 items, and 66% reordered at least 6 items. The Juypter notebook highlights several other interesting trends too for example: what department is the top reorders by percent or volume (right graph below).





6. Supervised Machine Learning

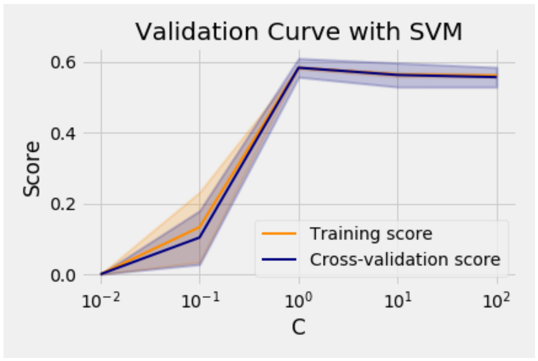
This section of the project has three parts: model training, model predicting, and model evaluation. The training requires splitting the data into a training set and a testing set. We first run both sklearn algorithms on the default setting. We then tune the hyper-parameters using Cross-Validation with Stratified K-fold RandomSearch CV. We inspect the learning curve for each parameter and refine the tuning parameters. We then take the best parameters and retrain on the training set. Next we run the model for predictions. Finally, we compare the models looking at Accuracy, F1 Score, Receiver Operator Curve (ROC), and the Area Under the Curve (ROC).

6.1 Support Vector Machine (first algorithm)

Support Vector Machine (SVM) with a radial basis function works by finding a separating hyperplane to classify the data samples: reorder (one / positive labels) or non-reorder (zero / negative labels). SVMs can be expensive to train but use less memory in predictions. It does this my saving only the data points that define the boundary (support vectors) and comparing new data points to where they are in relation to the decision boundary. Kernel tricks allow for non-linear boundaries. The draw back to this model is that it does not scale well for large numbers of observations or features. The model also does not do well with unbalanced data sets.

```
CPU times: user 9.48 s, sys: 127 ms, total: 9.61 s
Wall time: 9.61 s
Accuracy on training data: 0.56
Accuracy on test data: 0.42
F1 Score on train data: 0.0000
F1 Score on validation (train-test set) data: 0.0000
             precision    recall  f1-score   support

          0       0.42      1.00      0.59      4102
          1       0.00      0.00      0.00      5773

avg / total       0.17      0.42      0.24      9875
```

The initial default run of the SVM (results above) did poorly with a f1 score for positive labels at .0 and an AUC score of .44. To improve, we adjusted the “class_weight” setting for label balancing, and then tuned the hyper-parameters. The SVM has two parameters to tune: “C," regularization the amount of freedom to allow the model to fit also referred to as “smoothness” and “gamma” the radius of influence of the support vectors. We tuned with Cross-Validation using the “StratifiedShuffleSplit” to also assist in preventing the k-folds from having unbalanced label portions. We used the RandomSearchCV instead of the GridSearchCV as it has been found to be almost as accurate but with much shorter run times. After the initial tuning of the hyper-parameters, we inspected the learning curves to see if the ranges for the parameter could be refined. We looked for results near the edge of the range, for learning curves that stagnated, or that diverged between the testing and training. We found that the C value could be tuned better between 10^-2 and 1.



We retuned the parameters with the better defined parameter ranges and obtained new values for the parameters. We then retrained these parameters on the entire training set. We were able to improve the overall f1 score to .48 and the AUC .46.

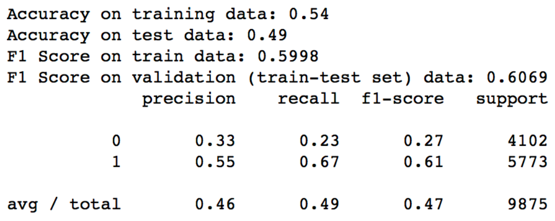
```
CPU times: user 11.5 s, sys: 82.6 ms, total: 11.5 s
Wall time: 11.5 s
Accuracy on training data: 0.57
Accuracy on test data: 0.47
F1 Score on train data: 0.5137
F1 Score on validation (train-test set) data: 0.4798
             precision    recall  f1-score   support

          0       0.40      0.54      0.46      4102
          1       0.56      0.42      0.48      5773

avg / total       0.49      0.47      0.47      9875
```

6.2 Random Forest (second algorithm)

The Random Forest works by averaging the results of decision trees. A bagging process, but better in that it also randomly samples features to train on. Each tree is built on a bootstrap sample. Classification is made by splitting data points into label groups on feature selections.The model is easy to parallelize and feature importance is easy to extract. This model is also sensitive to unbalanced data but sampling techniques (oversampling, subsampling, etc) can help. Random Forests are know to overfit the data easily. The default Random Forest model did fairly well.

```
Accuracy on training data: 0.60
Accuracy on test data: 0.44
F1 Score on train data: 0.4524
F1 Score on validation (train-test set) data: 0.3689

             precision    recall  f1-score   support

          0       0.40      0.68      0.50      4102
          1       0.55      0.28      0.37      5773

avg / total       0.49      0.44      0.42      9875
```

SciKit Learn has many parameters to tune, which we did again using Cross-Validation with the “StratifiedShuffleSplit” and the “class_weight” balanced setting. We used RandomSearchCV again for speed. We initially tuned for “max_depth," the number of splits allowed per tree, and “min_sample_leaf," the minimum number of nodes required per leaf. These parameters help in preventing the model from overfitting. The depth restricts the model from fitting to each data point. The minimum samples per leaf limits when to stop splitting as well. We found that the best depth of trees was 4 with a minimum number of samples per leaf of 50. We investigated the learning curve again and confirmed that these result were good. Next, we tuned the number of trees and the max number of features. The Random Forest error is strongly influenced by these two parameters. The error rate depends on the correlation between trees and the strength of each tree. The fewer trees and the more depth results in trees that are similar. Increasing the number of trees and reducing the depth decreases the correlation between trees and improves the error rate. Increasing the max features increases the correlation but the strength of each single tree too, so a balance has to be found. We found that the best number of trees was 10, and maximum number of features to be the square root of the feature space. We retrained the model on the entire training data set before predicting on the test set. We gained a significant improvement in predicting the positive labels moving from .37 to .61. However, the AUC did not improve and was still at .45.

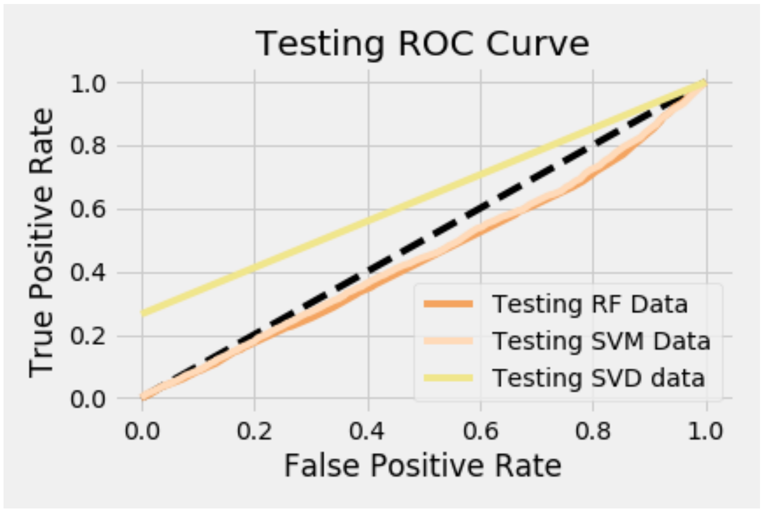


```
Accuracy on training data: 0.54
Accuracy on test data: 0.49
F1 Score on train data: 0.5998
F1 Score on validation (train-test set) data: 0.6069
             precision    recall  f1-score   support

          0       0.33      0.23      0.27      4102
          1       0.55      0.67      0.61      5773

avg / total       0.46      0.49      0.47      9875
```

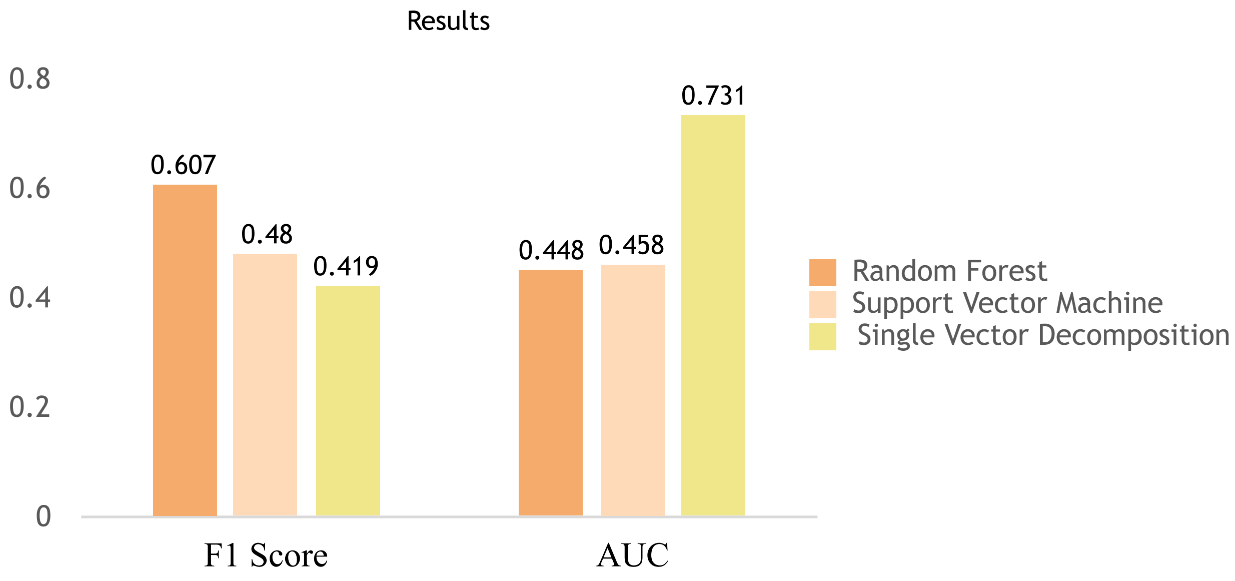
6.3 Single Vector Decomposition (third algorithm)

This model is the method that won the Netflix Prize. It is a method of collaborative filtering. Popular with movie recommending based off of user ratings. Here we used the customer’s product reorder rate as the “rating”. This model recommends items to customers based off of what similar costumers reordered. SciKit Learn does not have an algorithm to support it but using linear algebra and the Scipy package we built a recommender. My code developed for this can be found in the “SVD.ipynb” named Juypter notebook. It works by decomposing the data into 3 matrices and multiplying. The first matrix is the User interaction with the products. The second matrix is a diagonal with the entries as the singular values of the decomposition. The third matrix is the products relevance per feature. User’s preference for a product is the weighted sum of the user’s interest in each feature times the products relevance to the feature, in our case the reorder proportion. The code calls the defined function to make predictions for each user and adds these recommendations to a data frame for all users. The notebook has one parameter the setting for labeling positive samples. This sets the rate at which the notebook labels positive labels. It looks at the cosine similarity number labeled “Prediction” in the notebook. We chose to set this to .05 for our project. We can tune this later to find a possible better solution. Once all the predictions were gathered, we then organized the data into a format acceptable to pass to SciKit Learn for comparison to the other models.

7. Model Comparison:



The first thing to note is ROC curve; doesn't look great. The Support Vector Machine and the Random Forest does poorly here, both are below .5 percent. This indicates that we are doing fairly close to random chance. Two things could be happening. The model is fitting to noise, in which case we can improve by feature engineering. The worst case is there is no pattern detectable by our algorithms or that there is no generalizable indicator outside the training set. I suspect that the features are not informative enough for a class separation/ classification. (Example: Not using color in separating apples and oranges. Using roundness as a feature and it doesn't separate well enough). We could improve here by looking at more distinguishing features; that is by doing some more feature engineering.



We see that the Random Forest did best at predicting positive levels. The Single Vector Machine did best at predicting reorders overall. There is something to be said for both recommenders and would depend on the business function. The Random Forest is more accurate in predicting positive label reorders but worse misclassifying overall. The Single Value Decomposition is less accurate predicting positive labels but better classifying overall.

It is worth noting that Single Value Decomposition uses collaborative filter to decide while the Random Forest considers just the individual customer preferences. This means that the Random Forest is more likely to give results that the customer is more sure to like, but not recommend things the customer has not been exposed too. A lower risk of really bad recommendations. The Single Vector Machine is more likely to give products the user has never tried, based off of similar customer preferences. A higher risk of really bad recommendations.

The last thing to note is that the Random Forest is supported by SciKit Learn while the Single Vector Decomposition is not. Future employees would have to familiarize themselves with the code and process to maintain. Therefore, the Random Forest is easier to maintain moving forward.

8. Recommendations:

We recommend there courses of action. First, build an online platform to recommend products for revisiting customers in order to improve reorder purchases. The best option is the Random Forest with additional improvement to feature engineering.

The second option is to build the online platform to recommend products to revisiting customers using the Single Vector Decomposition with an additional improvement of Cross-Validated tuned hyper-parameters.

The third is to initiate a marketing campaign to lure customers back into stores before a week passes in order to capitalize on the frequency of reorder purchases during that period. The Random Forest algorithm could assist by providing enticing coupons for products customers would be interested in.