11.23.22 - Katin Lind - katinansa@proton.me

Instacart Recommendation Problem

Which items from an Instacart user's order history should be recommended when they open the app next?

Summary of Findings

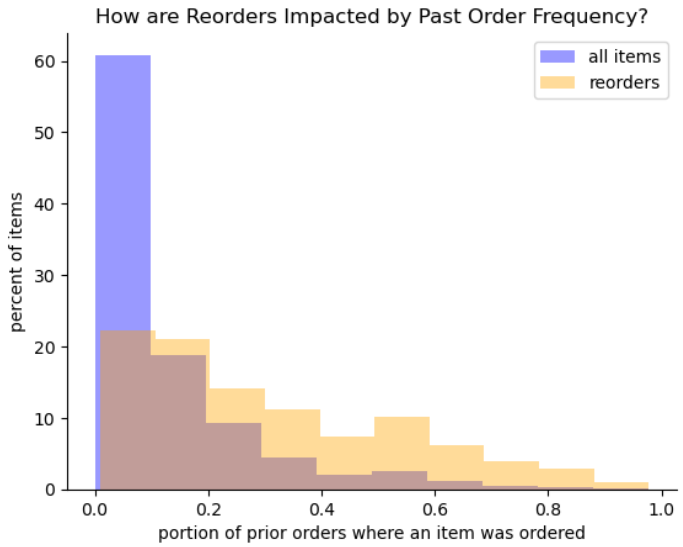
Phrased another way, the central question could be: Which items do we predict each user will reorder next? This is an important question because reminding users of items they wanted to buy anyway increases sales and customer satisfaction. Repurchasing patterns are quite idiosyncratic across users and not always consistent across a single user's orders; from all items they've ever purchased, each user is likely to only reorder a few on any given shopping trip.

Ideally, items will be recommended, as often as possible, that users do actually reorder, so true positives should be prioritized rather than accuracy. The goal of this project, therefore, has been to create a preprocessing pipeline and model that maximizes predictive efficacy for the minority class of "reorders" (strong F-measure and ROC-AUC and low Log Loss). Following feature engineering, a Random Forest classifier performed best (but not definitively so) in predicting reorders, with the following evaluation metrics: log loss score of 2.55, area under the ROC curve of 0.71, and an F1 score of 0.4. This compares with a log loss of 5.68 and F1 score of 0.09 (and, of course, roc_auc score of 0.5) if recommendations were made at random.

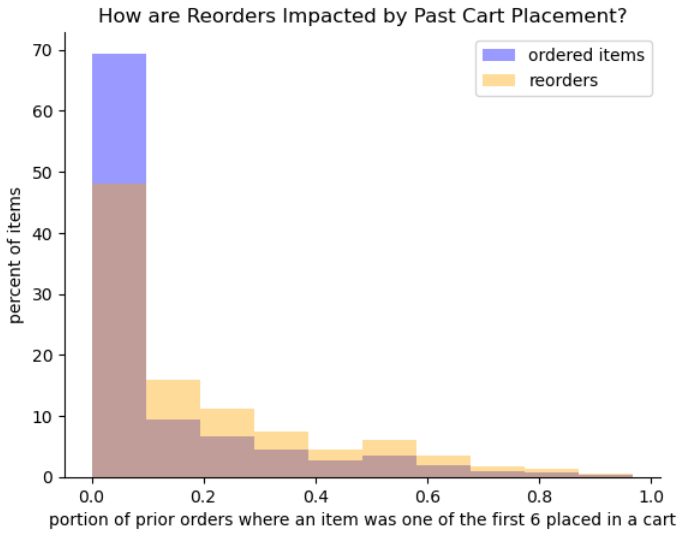
Process

In order to effectively predict reorders, some features need to be engineered to supplement raw data from users' order records. First, a trace of all items from a user's past orders should be added to each subsequent order. This way, it will be possible to predict whether any given item somebody has ever ordered will be a reorder or a non-order in their next order. Next, each item in each order should include information about a user's historical purchasing practices with that product. This is because the strongest predictors of whether a user will reorder an item are as follows: An item is more likely to be reordered if. . .

. . . it has been ordered and reordered frequently in the past. . .

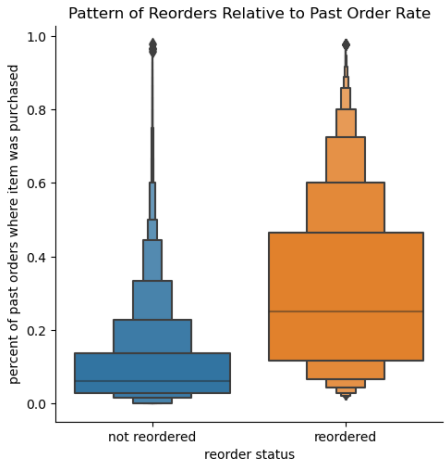


. . . and it has historically been placed in a user's shopping cart early (within the first 6 items).

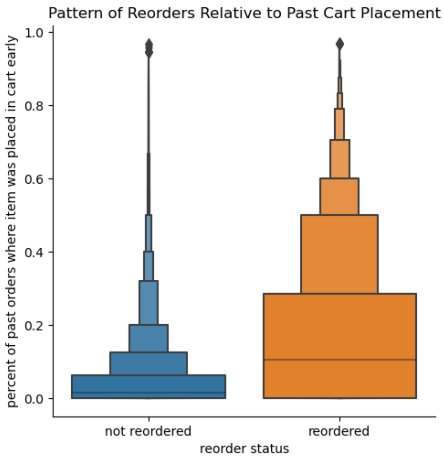


For more context on why past orders and “add to cart sequence” were key features to engineer, it is because they are more correlated with reorders than any other features. The feature next-most likely to be associated with an item reorder is “days since prior order.” Logically, more items get reordered the longer it’s been since the preceding order. Nevertheless, even this indicator is much less likely to help differentiate between non-orders and reorders:

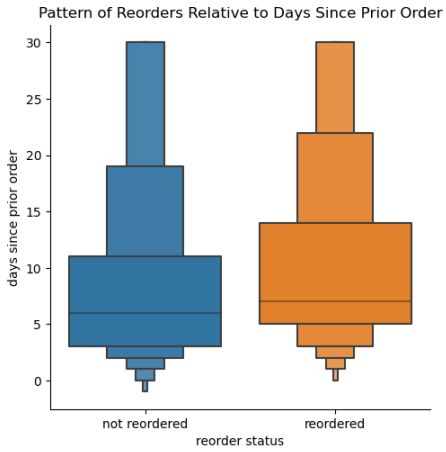
Reorders Relative to Past Order Rate



Reorders Relative to Past Cart Placement



Reorders Relative to Days Since Prior Order



Product type is also correlated with whether an item gets reordered. For example, if one were to simply guess that every item ever ordered by a user would be reordered in that user’s next order: They would be correct 3% of the time if that item was in the “Dairy & Eggs” department (the most popular department to reorder from) but only 1% of the time if it was a “Household” item (a department with median popularity for reordering) and next to never for “Personal Care” items.

Because there is some correlation between a product’s type and whether it gets reordered, I did engineer features that extract keywords from product names which would not otherwise be accessible to a model. For example, in the raw data, there was no “Organic” department, but many products are organic, so I created a column to indicate whether a product is organic or not.

Predictions were made better with the presence of product keyword features than without, but no single product keyword or department is, on its own, highly predictive of whether or not an item will be reordered.

One final step in data preparation was removing rows for items from the “Missing” department because predictions were weakened when those rows were present. My hypothesis is that this is because items currently in that department could realistically be reclassified across many other departments. These items’ presence in a department together may muddy any intrinsic ‘reorder-ability’ that a model can detect from other departments.

After munging data in these ways, multiple strategies were tried for encoding categorical features, such as binary, target, weight of evidence, and leave one out. A Target Encoder with default hyperparameters performed best. Target Encoder was trained on a test set prior to separately transforming independent variables’ train and test sets.

Following the completion of this preprocessing, a random grid search with cross-validation was used to find the best classifier for making predictions, along with its best-performing hyperparameters. SVC, KNeighbors, ADABoost, and

Bagging classifiers were ruled out, but there is not one overwhelmingly best-performing model between a Gradient Boosting Classifier and a Random Forest Classifier. Over several trials, the log loss score for gradient boosting comes out slightly lower (better) than that for a random forest, but the roc_auc and F1 scores for the random forest classifier are far higher (better). To reiterate, the random forest classifier resulted in the following evaluation metrics: log loss: 2.55, roc_auc: 0.71, f1: 0.4. Scores for a gradient boosting classifier were as follows: log loss: 1.8, roc_auc: 0.51, f1: 0.05.

Concluding Recommendations

I would recommend using a Random Forest Classifier with data processed as described above in order to make predictions about items each user will reorder, so that those predictions can be shared as recommendations when they log in next.

Specific hyperparameters for the classifier can be found in the 'models' folder of my project repo, and code used to perform all the processes described can be found in my notebooks, the contents of which are outlined in the project's ReadMe.

Finally, I do not know why some items (i.e. yogurt) that could obviously fit in more meaningful departments (i.e. dairy) are instead categorized as being part of the 'missing' department. If it is feasible, however, I would recommend completing a reclassification. This would likely improve model performance and enable better predictions for not only items currently in the missing department but all items, as well.

Additional Considerations

Modeling in this project has been done with a goal of predicting, for each item a user has ever ordered, whether it will get reordered in their next order. Another fruitful outcome could be to answer the question: "From among all the items a user has ever ordered, which 5 are they most likely to reorder first on their next order?" Then, if a user is recommended 5 items next time they log in, if even one of them is actually reordered, that would constitute a partial success for the recommendation system. I look forward to exploring models that provide such a flexible outcome in future iterations of this project.