STA141C Project: Kaggle Instacart Basket Prediction

Jiaping Zhang *, Austin Chi†, Jeremy Weidner‡
University of California, Davis

June 2016

1 Introduction

For our STA 141C project we chose to take on the Kaggle competition posted by the company Instacart, a grocery ordering and delivering app. The problem presented is given a set of various details about a user's purchase history can we make a prediction on what items from previous orders will be in the next order of any given user. This is a challenging multiclass multilabel prediction problem where we are asked to predict a basket of about 50,000 potential items for being reordered. The evaluation metric for the leaderboard is the average F1 score. We derived relevant features and experimented with 3 modeling approaches: Multi-label Classification with sklearn OneVSRest Classifier, Neural Network in tensorflow/tflearn and Gradient Boosting Tree algorithm with LightGBM.

2 Data

There are almost 50,000 unique product ID's. These serve as both items in the y variable for each prediction for a user as well as in their purchase history. In addition to the grocery item product ID's we are also provided with aisle and department ID's for a product, as well as day of the week and time of day for each order, the order number for a given order ID and if an item was reordered. The following picture shows the relationships between different data files.

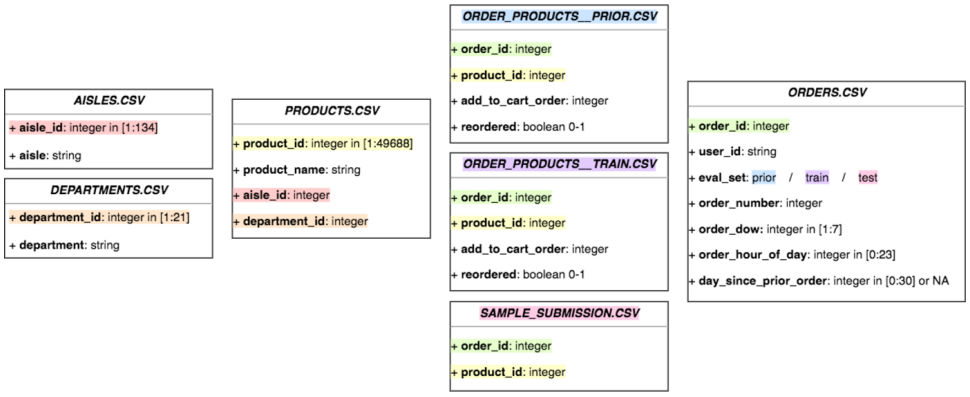

Figure 1: Links between Data Files

*Email: jpzhang@ucdavis.edu
†Email: aschi@ucdavis.edu
‡Email: jeweidner@ucdavis.edu

3 Approaches

3.1 Feature Engineering

As we know about Kaggle competitions, feature engineering plays a critical role to get a good accuracy performance. Based on the data provided by Instacart, we crafted mainly three categories of features about products, users and orders. It's worth mentioning that since there are large number of products, we ultilized the idea of word2vec to train a model based on the sequences data of products for each order to get the 100-dimension embedding vectors for each product. Given these complex data files relationships, it involves multiple steps of data munging to efficiently create features and join them together for generating training and test sets. The detailed explanation for the features are as following:

Table 1: Products Related Features

Features	Description
aisle id	aisle category for a product
department id	department category for a product
product orders	how many times a product was ordered
product reorders	how many times a product was reordered
product reorder rate	percentage from product reorders divided by product orders
product embedding	100 dimensional vector for a product

Table 2: Users Related Features

Features	Description
total orders	total orders from a user
total items	total products ordered by a user
total distinct items	number of distinct products ordered by the user
average days between orders	average number of days between orders for a user
average basket	average number of products in an order for a user
top reordered product	products being reordered most frequently before
purchase history embedding	weighted sum of product embedding vectors for all the products ordered in the user's purchase history

Table 3: Orders Related Features

Features	Description
day of week	day of week for an order
order hour of day	hour of day for an order
days since prior order	number of days since the last order for the same user
days since ratio	days since prior order divided by average days between orders for the user

3.2 Multi-label Classification with One-Vs-Rest

One of the methods we looked into was a One-vs-Rest classifier. This is a strategy that involves training one classifier per class that you are trying to identify. How it works are all samples of a given class are treated as positives for that classifier and all other samples are negatives. It then repeats this process, building n classifiers where n is the number of classes. It takes in a set of samples X and their corresponding Y labels for training. It returns a list of classifiers which are then used on a test set to generate the predicted probability for each Y label. This is a traditional and intuitive approach to tackle with the multilabel classification

problem. However, it could be inefficient due to the large number of classes and data size. With the sklearn One-VS-Rest Classifier, one can specify a basis classifier, such as "SGDClassifier" or "SVM". Our code for this model is in our github repository for the project under OvR in the models folder.

3.3 Neural Network

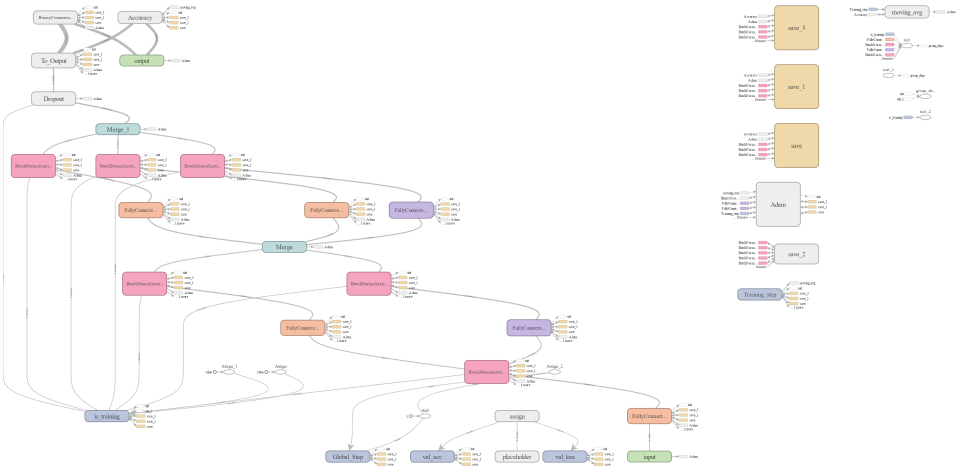

Figure 2: Our Neural Net Structure

A second method that we used is Neural Networks. Neural Networks are as system modeled after the biological nervous system's way of processing information, composed of a deeply complex series of interconnected elements. It learns by many iterations through a structure of interconnected nodes in layers connected by weights with biases. Types of layers include the input layer which takes the data from your source to feed to the network, Fully connected layers, batch normalization, and dropout. Our Neural Network takes in the 100 embedding vectors and output a feature vector of probabilities of all the products, using sigmoid output with binary cross-entropy. The structure we chose to use for our network is a 3 block network with batch normalization in each block. The code is in the models folder.

3.4 Gradient Boosting Trees with LightGBM

From our experiments with the above models, we realized it would not be computation efficient to use the multilabel training framework by using variable length of reordered items as the Y labels(used as a large sparse indicator matrix). Inspired by the discussion on the Kaggle competition forum, we could also translate this specific multilabel problem to a binary classification problem. Basically, we expanded the training data with each line of record corresponding to a potential product item from a user's purchases history, so that the Y label we are predicting is whether this product bought before would be reordered again or not. In this way, our training data has totally 8474661 samples per product and user, as compared to 131209 samples per user in the previous train set. Although the data dimension gets even larger, however, thanks to the great GBM library "lightGBM", it actually became breeze to train.

4 Results

4.1 One-Vs-Rest

While we thought that our One-Vs-Rest classifier would be an appropriate approach to solving this problem we found that since the time frame of this project was just a few weeks, by the time the data could be

sorted, joined and feature engineering conducted on it we did not have enough time to train the One-Vs-Rest classifier. Even when running on as many cores as we had access too it was training at a rate that would take many days and so we had to leave this approach as a hypothetical without true empirical results to show from it. We did however have takeaways in the form of knowledge from setting this model up. Things learned from it include that a main struggle of this approach is it can suffer from training off a skewed set of data where it has far more negative examples for a label than positive labels for a class. The main advantage of this classifier was supposed to be it's time complexity being simpler than other models as it builds only n classifiers but our n was so large (roughly 50,000) that we still ran out of time and were limited on computing power even if this method's computing power is valued as lower than similar methods.

4.2 Neural Network

The Neural Network we chose to use was very computationally intensive. After training for 12 hours the network had only completed 15 epochs. While the network was not particularly deep, the block structure made the network wide, making the training process computationally intensive. Our results were not good, with an F1 score of .04. This could be due to a lot of reasons. We chose the n largest probabilities as features, where n is the average number of items a user orders per orders, but that may not be accurate. Also, due to the extremely long training time, we could not tune hyper-parameters at all (number of layers, number of nodes, structure of graph, etc...). Overall, we were not surprised that this NN structure, with untuned hyper-parameters performed poorly.

4.3 Gradient Boosting Trees with LightGBM

The challenging part is preparing for the training data since we need to expand the data for each unique combination of user and product(bought by the same user before). We usually join the data with feature tables by user id and product id. For this binary classification problem, we train the LightGBM model with the "binary logloss" metric, 30 percent of the training data as the validation set and 500 iterations with early stopping in 10 runs. Other tree related parameters we tweaked are "num leaves" and "max depth".

The following graph shows the top 30 feature importance ranking. It shows that many of our features turn out to be quite useful.

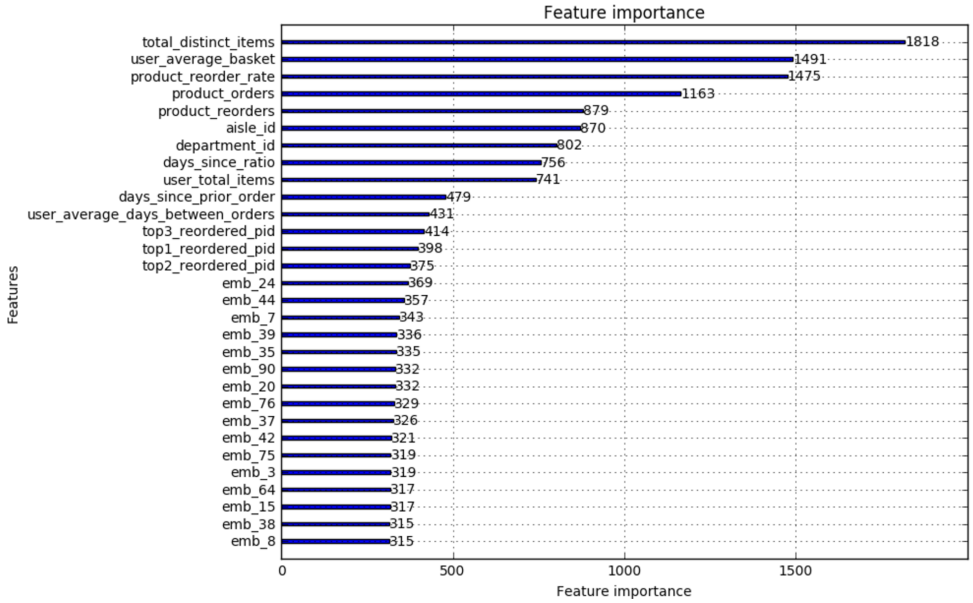

Figure 3: Top 30 most important features for binary classification

Since the final predictions format should be labels, we need to choose a certain threshold to select specific product ids and then concatenate them together for the same order ids or user ids. We compared our predictions directly through the leaderboard scores although it is not a comprehensive test set.

Table 4: Prediction Performance by Threshold

Threshold	Leaderboard Score
0.3	0.1673362
0.2	0.2310302
0.15	0.2560709
0.1	0.2586575

The result only led us to the 523/650 position in the leaderboard, which indicates it's still a long way for improvements. Although our predictions are not good enough, we think this is partly because we haven't spent enough time on parameter tuning for LightGBM and selecting thresholds.

5 Future Work

Since One-VS-Rest classifier is not promising in this application, we want to focus more on the neural network and gradient boosting tree methods. For neural network, we can train for more epochs and tune hyper-parameters with more features. For LightGBM, we also need to perform more rigorous parameter tuning and threshold selection. Especially, a dynamic threshold for each product would be more useful in this case. Besides, we should delve more into feature engineering. For example, we can do clustering for customer segmentation and use it as an additional feature. Other methods we would want to explore for this type of problem include seq to seq Neural Nets, and field-aware factorization machines.

6 Conclusion

This project was a big learning experience in many ways. We learned firstly not to underestimate a multi-label multi-class classification problem of such scale especially considering there are large number of classes. Those limitations aside we learned lots about the construction of a neural network for a classification problem as well as other things such as what a One-Vs-Rest Classifier is, what it's used for and what its hindrances are, and also how to implement a end-to-end LightGBM training pipeline with significant amount of data wrangling and feature engineering. We applied various programming techniques to address the memory computation issues, such as converting more efficient data types and controlling garbage collection process. In this project, Jeremy worked on the sklearn One-Vs-Rest Classifier, Austin worked on the Neural Network approach and Jiaping worked on the feature engineering and LightGBM implementation. All the code for this project can be found on this github repository. Please let us know if you have any feedback or questions.