# Classifying Black Friday Products (Kaggle Competition)

## Objective

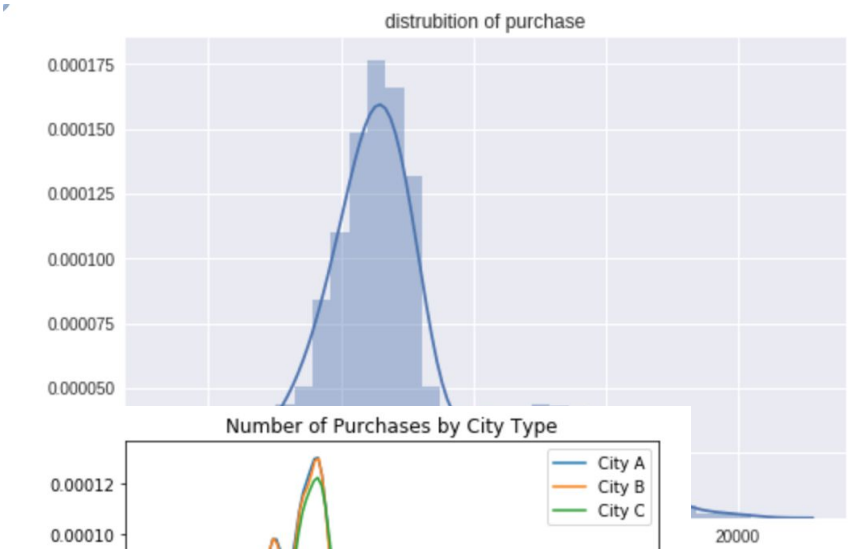
To classify a list of black friday products correctly.

## Team Members

1. Fan, Jordan
2. Whitney, Bizuayehu

## Data Preprocessing

We thought the data was pretty messy and difficult to make sense of at first. We first listed all the columns and the types for each column to explore the data. Then we explored all the unique values for each of the columns, getting their counts. We saw that there are 21 unique values for occupation, 3623 unique products, 5891 unique users, and 18 categories 2, 7 age categories, and 3 city categories. We also found that there are 18 possible “Product_Category_1” labeling that our model needs to label each “Product ID” to.

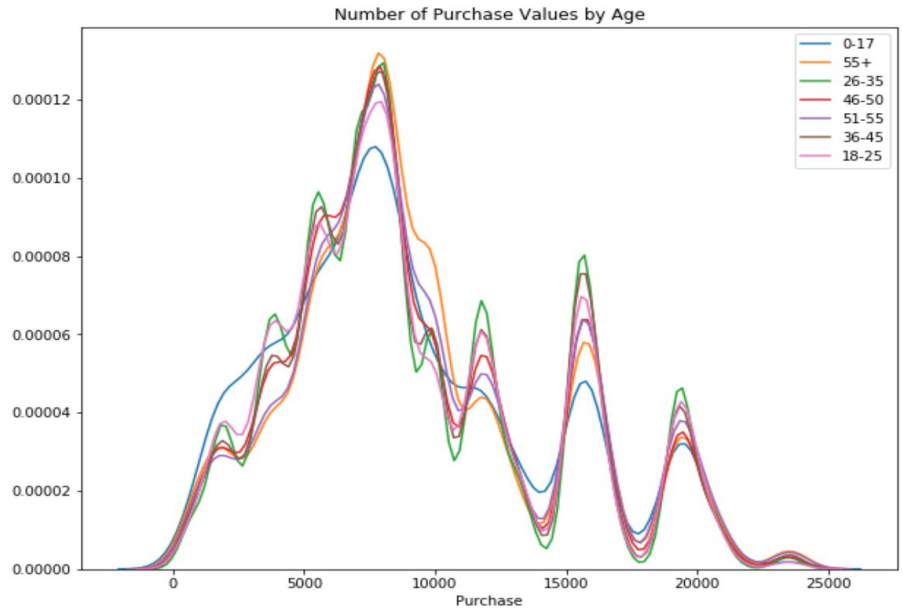




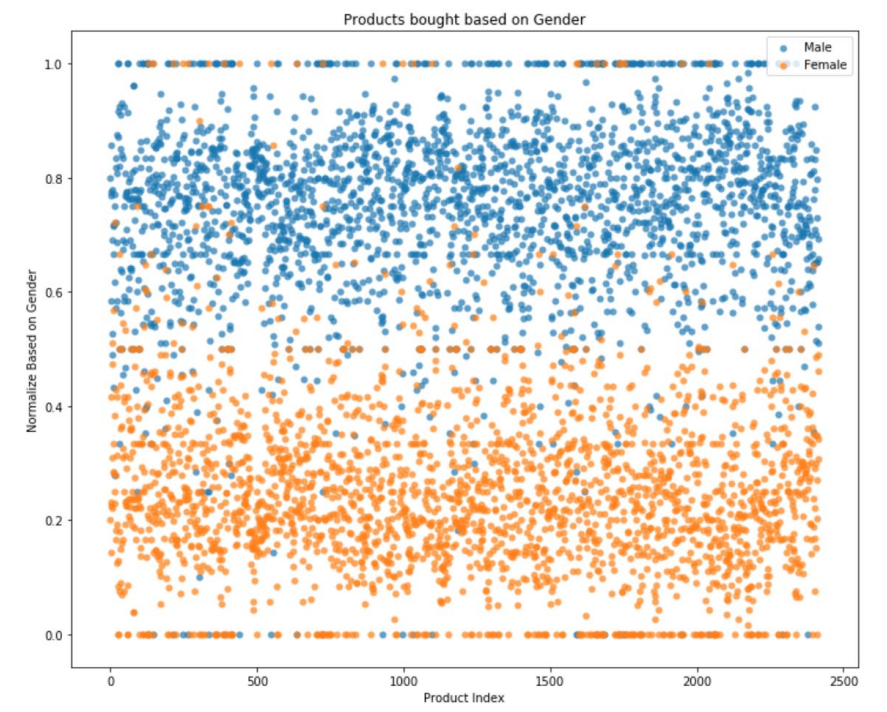
We graphed some of the columns to try to get a visual understanding of the data. The number of purchases seem to demonstrate a bimodal distribution, with the first mean at around 7000 purchases and the second at 13000 purchases.

Breaking down the distribution into city types, each individual city distribution all demonstrate the same oscillating trend

in purchases. However, City A and B seem to demonstrate more density in lower purchase amounts while City C tend to be more concentrated in higher purchase amounts.




Unsurprisingly, the age group with the lowest amount of purchases is from 0-17. The 0-17 age group also have the smoothest distribution, not exhibiting the same intensity of spikes and dips. The age group with the most purchases is 26-35.




An interesting thing we found was that some products were mostly purchased by male customer while other products were mostly purchased by female customers. However, the number of men is about 3 times that of females.

We also needed to clean up the data because there were missing values. We filled the missing values for the "Product_Category_2" and "Product_Category_3" with zeros initially. We later realized that

Product Category 3 isn't very helpful because it has so many missing values. So we ended up dropping it for our final training data.

# Feature Engineering

We ended up using "City_Category", "Age", "Gender", "Martial_status", "Product_Category_2", "Number_Users", and "Purchase". We decided on these features because they gave us the highest accuracy compared to the other combination of features we used to train and test our model. We hot encoded most of the features and we took the sum or mean when we grouped by the "Product_ID". We dropped the other features because they had missing values or weren't improving the accuracy of our model.

# Models / Techniques

We used classification models. After grouping by the product ids, our dataset wasn't that big and therefore a neural network probably wouldn't be a good model to use since it requires large amounts of data to train well. We first used KMeans and KNN to cluster the product ids then used the prediction from the KMeans and KNN as features for training our final model. However, we ended up not using the KMeans and Knn because their accuracy weren't that high and using their predictions as features didn't improve the accuracy of our final model. We then tried using a decision tree with AdaBoost. We saw a significant improvement in our testing accuracy and then decided to try bagging with a decision tree to see if that would improve our model more than the AdaBoost. From our test set, we saw a slight improvement. We then tried a random forest model since it has a similar concept as bagging but is able to look at subsets of the variables and should therefore be less susceptible to overfitting based on the features. However, the accuracy we observe with test set is similar to that of bagging. We decided to make submissions with AdaBoost, bagging, and random forest.

# Results

We used 10 and 5 fold cross validation to evaluate our model and split the data into training and testing sets with the training set containing ⅔ of the data and the test set having ⅓ of the data. We tested the model on the test set and used accuracy to evaluate the model on the test data. We mostly changed the max depth of our decision tree and use the accuracy of the prediction of the model to evaluate it. The results of each model is listed below.

| Ensemble | Model | Cross Validation Accuracy | Testing Accuracy | Kaggle Accuracy |
|---|---|---|---|---|
| None | K Means | N/A | 87% | 63% |
| None | Knn | 57% | 58% | N/A |
| Adaboost | Decision Tree | 69% | 70% | 77% |
| Bagging | Decision Tree | 69% | 73% | 75% |
| Bagging | Random Forest | - | 73% | 75% |

# Conclusion

When first working with the data, we one hot encoded all the columns except for the product and user id and then after grouping the dataset by product id we aggregated the data by taking the mean of each column. After fitting our model and getting accuracies, we realized that some of the columns that are really categorical data, such as occupation and product category 2 were encoded as numeric. So we converted those columns to strings and then one hot encoded those columns and retrained our models. Surprisingly, we ended up with a lower accuracy, going from 75% to 71% on the kaggle submission using random forest and 77% to 75% with AdaBoost. A possible explanation could be that some of the categories in occupation or product category 2 aren't very helpful in predicting the product category 1 and one hot encoding those columns caused the model to overfit on the data. Thus, being able to split on the data as numeric would allow the splits to bypass this overfitting. So using the original one hot encoded data and then adding an extra feature of the logged number of users who purchased the product, we trained our models and then submitted to kaggle, with AdaBoosting having the highest accuracy, and the decision tree and random forest having similar accuracies.