Introduction.

Our project comes from a kaggle competition and uses data from a company called Home Credit. Home Credit is a company that operates in several different countries including several countries in Europe and Asia in addition to the United States. The company's business model focuses on making loans to people with little or no credit. Their loan products primarily include point of sale loans (POS), cash loans, credit cards, debit cards, revolving loans, or car loans.

Our project uses supervised learning to predict a client's ability to repay a loan. Given the company's loan products, business model, and regions of operation, there are several unique challenges with this task. First, given that the company operates primarily by making loans to people with little or no credit, the ability to which we can use default on previous loans or credit history is limited. Moreover, since there is variation in which the company operates in, there is variation in measurement in various credit bureaus in each country. For example, lenders in the United States often use FICO scores as a large predictor in a borrower's ability to repay a loan. Given these challenges, we make use of several datasets provided by Home Credit.

We divide the project as follows. I wrote the introduction and Diana wrote the description of the dataset. For implementing the code, Zoey did support vector machine, I did random forest, and Diana did Naïve Bayes. Zoey did much of the preprocessing of the data, but we each customized based on what was necessary for our models.

Description of Individual Work

I wrote the proposal and introduction and implemented the random forest model. Random Forest is an ensemble method that makes use of several decision trees and can be used for both regression and classification. Since our problem is a classification method (a person either defaults or does not), we use it in this context.

The basic principle behind Random Forest is to create several different decision trees from random pulls of the dataset, and then average these decision trees. The general idea is to average several unbiased decision trees to reduce the variance. The random forest algorithm corrects for overfitting that often happens with decision trees, and it also allows for us to more easily observe feature importance. Since Random Forest pulls several random samples of the data to create several decision trees, we do not need to worry as much about dimensionality reduction or correlation across variables as much with this model. Please see the image below for a more in-depth description of the random forest model, which comes from *The Elements of Statistical Learning*:

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

After preprocessing, we noticed that the training dataset was highly imbalanced, with 8,076 observations in class 0 (not default) and only 526 observations in class 1 (default). At first, we performed the trainings using the imbalanced dataset, but only inputs in class 0 are correctly classified. To tackle this issue, we resample the data by randomly deleting the observations in the over-represented class (i.e. class 0). As a result, we create a new balanced training dataset with 526 observations in each class.

Description of the work I did

I wrote the introduction and proposal for the project. I also implemented the random forest aspect of this project. A walkthrough of my code is provided as follows. I first start by importing the necessary packages and loading in the dataset. After inspecting the data, we do some necessary preprocessing, which includes dropping observations with missing values and looking at summary statistics:

```
# %%-------------------------------------------------------------------
# Import packages
import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix
```

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, roc_curve, roc_auc_score
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
import os
warnings.filterwarnings("ignore")
pd.set_option('display.max_columns', 100)
os.chdir("/Users/Kim/Documents/GWU Class/Machine Learning I/Final-Project-Group5-master/all")

# %%------------------------------------------------------------------------
# Import the data
data=pd.read_csv("application_train.csv")
print(data.head())
print(data.isnull().sum())
print('\n')
# Drop rows with missing values
data=data.dropna()
data.info()
print('\n')
print("Dataset No. of Rows: ", data.shape[0])
print("Dataset No. of Columns: ", data.shape[1])
print('\n')
print(data.describe(include='all'))
print('\n')
print(list(data))
print('\n')
print(data.dtypes)
print('\n')
print(data["TARGET"].value_counts())
print('\n')
```

Next, we encode all the categorical variables:

```
obj_columns = data.select_dtypes(include=['object']).columns
print(obj_columns)
data[obj_columns] = data[obj_columns].astype('category')
data[obj_columns] = data[obj_columns].apply(lambda x: x.cat.codes)
```

After completing these preprocessing steps, we noticed that our dataset was severely imbalanced, with many more targets of 1 than 0 in the data. In order to correct for this, we randomly resample the data to reduce the overrepresented class:

```
# %%------------------------------------------------------------------------
# Re-sample the data to reduce the over-represented class
print(data["TARGET"].value_counts())
print('\n')
data1=data.loc[data['TARGET']==1]
data0=data.loc[data['TARGET']==0].sample(n=data1.shape[0])
print(data1.shape)
print(data0.shape)
data=pd.concat([data0,data1])
print(data.shape)
# split the dataset
# separate the target variable
x = data.values[:, 2:]
y = data.values[:, 1]
```

Finally, we are able to split the dataset into features and targets. Since I am using the random forest algorithm, it is not necessary off the bat to reduce the features. This is because the random forest algorithm works by creating several decision trees based off random samples of the data and averages results. Finally, I encode the targets using LabelEncoder, and I split the dataset into training and test data using the standard 70/30 split:

```
# split the dataset
# separate the target variable
x = data.values[:, 2:]
y = data.values[:, 1]

# encloding the class with sklearn's LabelEncoder
class_le = LabelEncoder()
y = class_le.fit_transform(y)

# split the dataset into train and test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=100)
#
# # %%----------
```

Note that I do not include the first column in my features, which is simply because the first column is a unique identifier for the client. Thus, this will have no predictive ability in my model. Finally, my next step includes performing the training. I start by initializing the classifier from sklearn's RandomForestClassifier and then I fit the model using the training and test sets. One very nice aspect of the random forest classifier is that it gives us information about feature importance. Using this information, I create a graph that displays feature importance in descending order. Next, I use the predict function to make predictions on the test data and assess

the accuracy of our model using predict_proba. Using this information, I am able to run a classification report to get the appropriate metrics for evaluating whether or not this model worked well:

```
rf = RandomForestClassifier(n_estimators=100)

# perform training
rf.fit(x_train, y_train)


#plot feature importances
# get feature importances
importances = rf.feature_importances_

# convert the importances into one-dimensional 1darray with corresponding df column names as axis labels
f_importances = pd.Series(importances, data.iloc[:, 2:].columns)

# sort the array in descending order of the importances
f_importances.sort_values(ascending=False, inplace=True)

# make the bar Plot from f_importances
f_importances.plot(x='Features', y='Importance', kind='bar', figsize=(121, 9), rot=90, fontsize=6)

# show the plot
plt.tight_layout()
plt.show()

# predicton on test using all features

y_pred = rf.predict(x_test)
y_pred_score = rf.predict_proba(x_test)


# calculate metrics all features

print("\n")
print("Results Using All Features: \n")

print("Classification Report: ")
print(classification_report(y_test,y_pred))
print("\n")

print("Accuracy : ", accuracy_score(y_test, y_pred) * 100)
print("\n")
```

```
print("ROC_AUC : ", roc_auc_score(y_test,y_pred_score[:,1]) * 100)
```

In order to attempt to improve the model's accuracy, I rerun this code using just the 25 most important features rather than all the data in the dataset:

```


#perform training with random forest with k columns
# specify random forest classifier
rf_k_features = RandomForestClassifier(n_estimators=100)


# select the training dataset on k-features
newX_train = x_train[:, rf.feature_importances_.argsort()[::-1][:30]]

# select the testing dataset on k-features
newX_test = x_test[:, rf.feature_importances_.argsort()[::-1][:30]]
# train the model
rf_k_features.fit(newX_train, y_train)
# prediction on test using k features
y_pred_k_features = rf_k_features.predict(newX_test)
y_pred_k_features_score = rf_k_features.predict_proba(newX_test)

print("\n")
print("Results Using K features: \n")
print("Classification Report: ")
print(classification_report(y_test,y_pred_k_features))
print("\n")
print("Accuracy : ", accuracy_score(y_test, y_pred_k_features) * 100)
print("\n")
print("ROC_AUC : ", roc_auc_score(y_test,y_pred_k_features_score[:,1]) * 100)
```

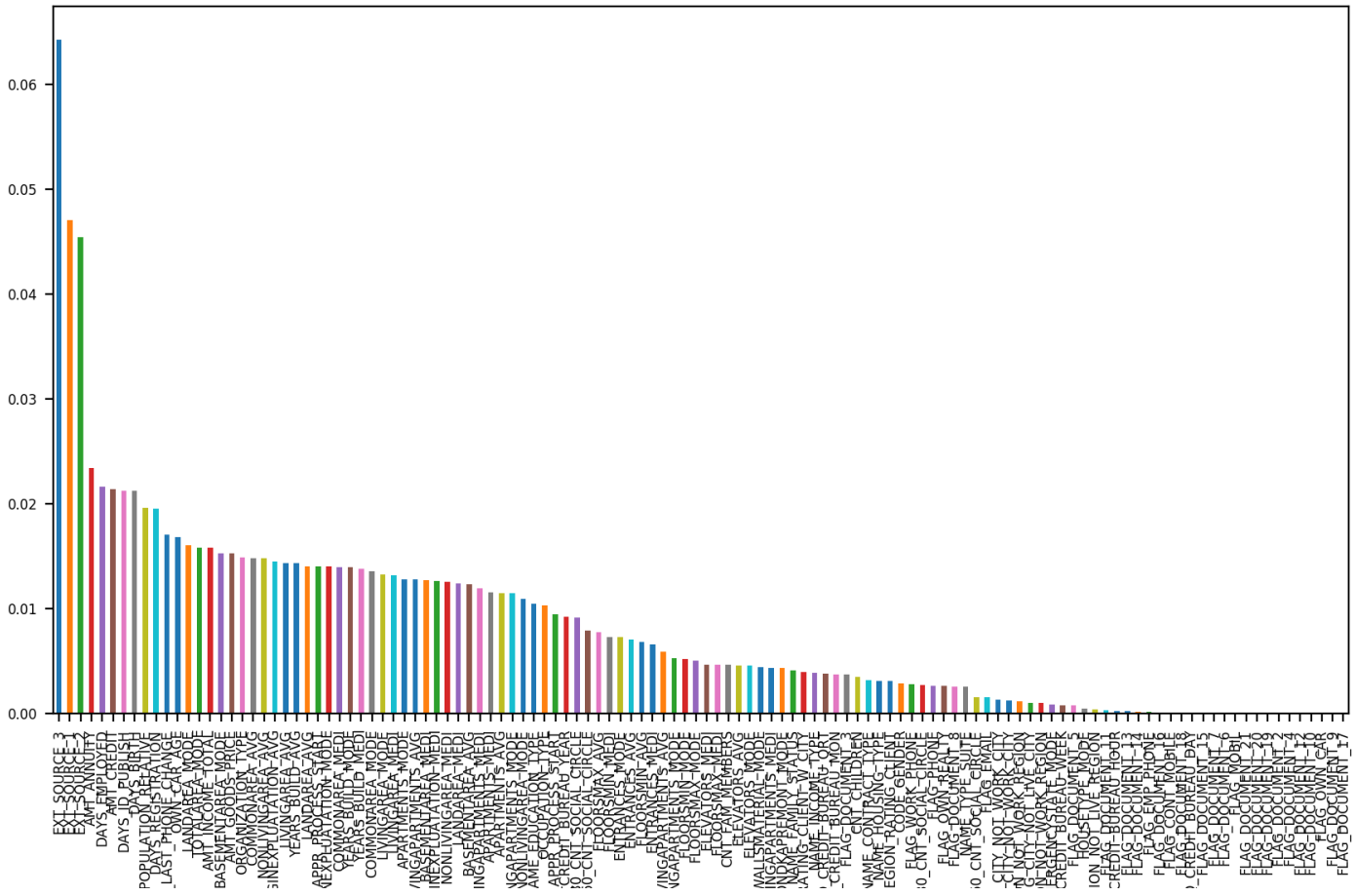
Finally, I compute confusion matrices for the model using all features and the model using 25 features:

```
# %%------------------------------------------------------------------
# confusion matrix for all features
conf_matrix = confusion_matrix(y_test, y_pred)
class_names = data['TARGET'].unique()


df_cm = pd.DataFrame(conf_matrix, index=class_names, columns=class_names )

plt.figure(figsize=(5,5))

hm = sns.heatmap(df_cm, cbar=False, annot=True, square=True, fmt='d', annot_kws={'size':
```

```
20}, yticklabels=df_cm.columns, xticklabels=df_cm.columns)

hm.yaxis.set_ticklabels(hm.yaxis.get_ticklabels(), rotation=0, ha='right', fontsize=20)
hm.xaxis.set_ticklabels(hm.xaxis.get_ticklabels(), rotation=0, ha='right', fontsize=20)
plt.ylabel('True label',fontsize=20)
plt.xlabel('Predicted label',fontsize=20)
# Show heat map
plt.tight_layout()
################################################
# confusion matrix for k features
conf_matrix = confusion_matrix(y_test, y_pred_k_features)
class_names = data['TARGET'].unique()


df_cm = pd.DataFrame(conf_matrix, index=class_names, columns=class_names )

plt.figure(figsize=(5,5))

hm = sns.heatmap(df_cm, cbar=False, annot=True, square=True, fmt='d', annot_kws={'size':
20}, yticklabels=df_cm.columns, xticklabels=df_cm.columns)

hm.yaxis.set_ticklabels(hm.yaxis.get_ticklabels(), rotation=0, ha='right', fontsize=20)
hm.xaxis.set_ticklabels(hm.xaxis.get_ticklabels(), rotation=0, ha='right', fontsize=20)
plt.ylabel('True label',fontsize=20)
plt.xlabel('Predicted label',fontsize=20)
# Show heat map
plt.tight_layout()
```

Results

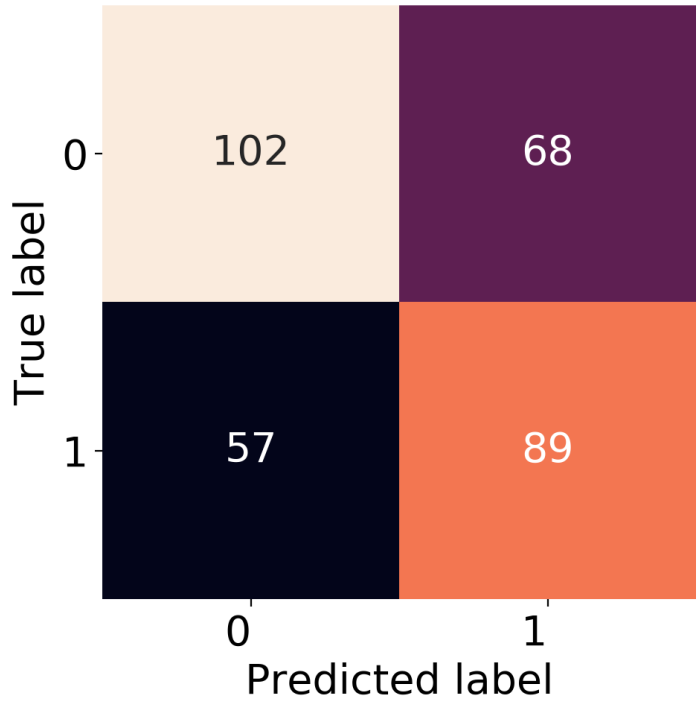
I start by using all the features in the dataset to implement my random forest algorithm. The feature importance, classification report and confusion matrixes are shown below:



From this picture, we can see that the EXT_SOURCE_1, EXT_SOURCE_2, and EXT_SOURCE_3 variables are the most important features in predicting whether or not a client will default. These variables are scores from external sources. The next most important features are the amount of annuity, length of employment, and amount of credit. For more in depth descriptions of each variable, please see “HomeCredit_columns_description.csv”.

```
Results Using All Features:

Classification Report:
             precision    recall  f1-score   support

          0       0.64      0.60      0.62       170
          1       0.57      0.61      0.59       146

avg / total       0.61      0.60      0.60       316


Accuracy :  60.44303797468354


ROC_AUC :  65.03827558420629
```

Above is the classification report for this model. I used 100 trees in this model, although I also tried 200, 400, 800, and 1000 trees and it did not do much to improve performance of this model. We can see from the above classification report that the model actually performs quite poorly. This can be seen in the overall accuracy of the model, which is only approximately 60.44%. This means that only about 60% of the predictions that were made were made correctly. We also note that the ROC_AUC score is about 65%. This represents the model's ability to correctly classify those who will default and who will not fault. A score of 65% is quite poor by this metric. Below is the confusion matrix for this model as well:

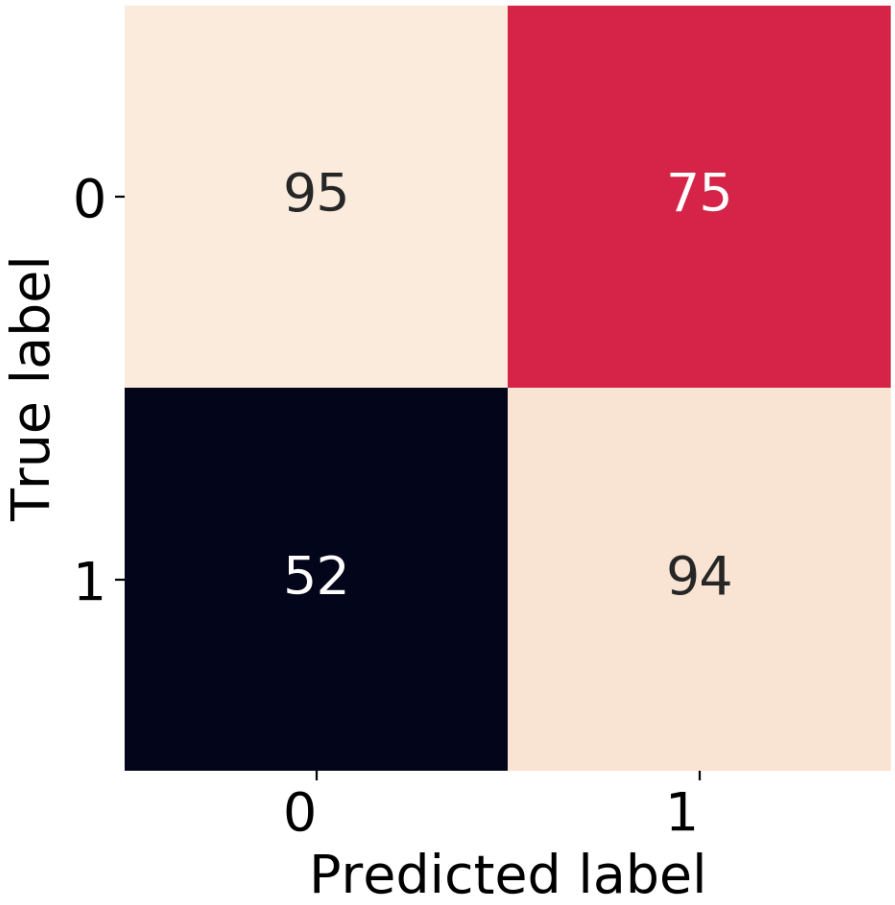


In interpreting this confusion matrix, we can see that we correctly predict that 102 of the clients will not default, while we incorrectly predict 68 clients who will default, but actually do not default. Similarly, we incorrectly predict that 57 of the clients that actually defaulted will not default, and we correctly predict that 89 of the clients that actually defaulted will default.

I rerun this model using just the top 25 most important features from the feature importance graph above. Below is the classification report from running this model:

```
Results Using K features:

Classification Report:
             precision    recall  f1-score   support

          0       0.65      0.56      0.60       170
          1       0.56      0.64      0.60       146

avg / total       0.60      0.60      0.60       316


Accuracy :  59.81012658227848


ROC_AUC :  65.75543916196615
```

As you can see from the results, the accuracy of the model is roughly the same when limiting to just 25 features. The ROC_AUC score improves slightly, but it still poor. The confusion matrix is show below:



The results of the confusion matrix actually suggest that the model with 25 features performs slightly worse than the model using all of the features. Similarly to the confusion matrix that uses all the features, we can see that we correctly predict that 95 of the clients who did not default will not default, while we incorrectly predict 75 clients who will default, but actually do not default. Similarly, we incorrectly predict that 52 of the clients that actually defaulted will not default, and we correctly predict that 94 of the clients that actually defaulted will default.

Summary and Conclusions

The random forest gives several interesting results, but ultimately does not serve as a very effective model for classification in this context. The results from the first model that uses all of the features first tells us some important information about feature importance in this model. Scores from external sources, the amount of annuity, length of employment, and amount of credit are among some of the most important features in this model. The classification report tells us the accuracy of this model as well as the area under the ROC curve. Since the accuracy of the model is only about 60% and the area under the ROC curve is about 65%, we can conclude that this model performs poorly. Similarly, when we limit to just the 25 most important features, we see a similarly poor performance.

I learned much from this project on random forest. First, I learned that having a balanced dataset is very valuable. If you have a dataset that has several more of one class than the other, the model will have substantial trouble classifying the smaller class correctly. This can be solved for with a simple random resampling of the data. Next, in context of this problem, the number of decision trees actually used in the random forest model did not make much difference in generating an accurate model.

I believe that several improvements could be made to this specification if there were more time. For one, I believe that adding in more features would be very helpful for predictive ability. For example, we might be able to pull in more information about a client's past loans (from Home Credit as well as other institutions). This would need to be dealt with carefully, since there are many clients in Home Credit's dataset that simply do not have any other existing loans or credit. Therefore, we would have to limit the sample to just those with previous loans or credit, and the results from that sample might not be able to extrapolated to those with no credit history. We might also consider running models with several different feature specifications as a way to further test for feature importance. For example, we found in the first model that external scores were the most important features in predicting if a client would default or not. It would be worth exploring to what degree these features are correlated, and then attempting to select features that are not highly correlated.

Percentage of Code from the Internet

For my code, I used the class example codes and code on Professor Jafari's github as a starting point. All in all, I used 111 lines of code, and I modified roughly 20 lines of code to fit my specific data needs. Thus, my percentage of the code I found/copied from the internet is about 82%.

References

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning.* Springer Series in Statistics Springer New York Inc., New York, NY, USA, (2001)