Data Science Final Project: Credit Card Approval Method

Hannah Adamson
Department of Computer Science, University of Georgia
Kavya Ahuja
Department of Computer Science, University of Georgia
Lakshmi Yetukuri
Department of Engineering, University of Georgia

Abstract

The objective of this project was to build models to review credit card applications and determine if they should be approved or denied. We were given historical data from credit card applications, already split into training, validation, and testing data sets. This data was given to us in the form of three .csv files, so preprocessing was necessary to have usable data to train our models. The three models we chose to analyze were Logistic Regression, Random Forest, and Gradient Boosting. Our Logistic Regression model had a test accuracy of 0.9354, Random Forest had a test accuracy of 0.9362, and Gradient Boosting was our most accurate model with a test accuracy of 0.9372.

1. Introduction

The project is to create a model and decide which credit card applications should be approved based on the historical data provided. Our group found it intriguing to work on a model that is close to a real-world problem.

The main objective of the project was to conduct an exploratory analysis and develop models that are used in determining which credit card applications should be approved. The analysis and the models support each other in making sure that the prediction is nearly accurate. The data sets used in this project are a training data set, which has a model of 20,000 accounts, validation data set, which has a model of 3,000 accounts, and testing data set with 5,000 model accounts.

To experiment, we focused on five models- Logistic Regression, Random Forest, Naive Bayes Classifier, Neural Networks and Gradient Boosting. The Logistic Regression required scaling using the StandardScaler() from sklearn's preprocessing package. We also experimented with overfitting the data when training the Random Forest Model, using SMOTE. We experimented to optimize the Random Forest Model by use of feature importance. Our Gradient Boosting Model required us to find the optimal learning rate, where we iterated between rates from 0.05 and 1.0.

Upon training these models, we had the following results:

	Test	**Validation**
Logistic Regression	0.9354	0.942
Random Forest	0.9362	0.942
Gradient Boosting	0.937	0.936

Table 1: Overall Model Results

2. Data Analysis/Preprocessing

We were given historical data from credit card accounts for a hypothetical bank XYZ, defined as a regional bank in the south-eastern region. The data given to us was split into a training, validation, and test set. The training data set contained 20,000 accounts, the validation data set contained 3,000 accounts, and the test data set contained 5,000 accounts. It was first necessary to read these csv files into a dataframe, using the Pandas pd.read_csv() method [4].

Our next step to make the data usable was to separate it into X and Y for the training, test, and validation sets. The Y value for each set is the 'Default_ind' value, which represents the Indicator of Default. The binary value of 1 means the account defaulted after being approved and opened within a period of 18 months, and the binary value of 0 means the account was not defaulted (defaulted means there were no payments for three consecutive months). After setting the Y value of each data set to the 'Default_ind', the remaining columns made up the data frame for the X value.

```
y_Train = train['Default_ind']
x_Train = train
x_Train.drop('Default_ind', axis='columns',inplace=True)
```

All of the columns consist of numerical data, except for the States. It was therefore necessary to use the Pandas get_dummies method in order to transform the categorical data into indicator variables. This is a straight forward way to transform the categorical data in one step.

The training, test, and validation sets were all imbalanced. The value counts of the 'Default_ind' value of each dataset, the Y value for our data, were found with the command print(y_Train.value_counts()), replacing train with Test and Validation sets to find the respective counts. These value counts are listed in the table below:

Training	**Test**	**Validation**
0-18414	0-4599	0-2778
1-1586	1-401	1-222

Table 2: Indicator of Default value counts

There were missing values in the uti_card_50plus_pct(the utilization, or ratio of balance divided by credit limit, on all currently available credit card accounts) and rep_income(self reported annual income) columns of our data, with the number of null counts shown in the table below:

	Training	**Test**	**Validation**
uti_card_50plus_pct	2055	499	297
rep_income	1570	383	253

Table 3: Null counts

We decided to replace the null values with the mean value of the given column. Since there were a large number of null values, removing them all would result in a great loss of data.

x_Train['uti_card_50plus_pct'].fillna((x_Train['uti_card_50plus_pct'].mean()), inplace=True)
x_Train['rep_income'].fillna((x_Train['rep_income'].mean()), inplace=True)

We originally trained the models with all values given, receiving fairly high accuracies. In an attempt to increase the accuracy of the models, we extracted the feature importance values for the Random Tree model.

importanceRF = pd.DataFrame('feature': list(x_Train.columns), 'importance': rf.feature_importances_). sort_values('importance', ascending = False)

The features with the lowest importance, 'ind_acc_XYZ' with a value of 0.007916 and 'auto_open_' '36_month_num' with a value of 0.006956 were removed, very slightly increasing the accuracy of the random tree model from .9358 to .9366.

We scaled the data for the logistic regression model using the StandardScaler() from sklearn's preprocessing package. We did this with the training, test, and validation data, so all would be uniform. The StandardScaler() works to standardize features by removing the mean and scaling to unit variance. The standard score of a sample x is calculated as: z = (x - u) / s where u is the mean[3].

scaler = preprocessing.StandardScaler().fit(x_Train)
x_Train = scaler.transform(x_Train)

Another technique we experimented with when preprocessing the data is SMOTE. SMOTE, or the Synthetic Minority Over-sampling Technique, uses a combination of over-sampling the minority (abnormal) class and under-sampling the majority (normal) class [1]. Incorporating SMOTE to our data set did not improve our accuracy. For example, the Random Forest model trained with the original data had an accuracy of .9358 when run on the test data. The Random Forest model trained with data synthetically sampled with SMOTE had an accuracy of .9292 when run on the training data.

from imblearn.over_sampling import SMOTE smote = SMOTE() x_TrainOV, y_TrainOV = smote.fit_resample(x_Train, y_Train)

3. Experiments

3.1 Reasoning For Chosen Models

Logistic regression frameworks are used for predictive analysis when the dependent Y value is binary, as it was in our problem. The purpose is to calculate the probability of a binary event occurring, based on occurrences in the training set.

Random forest models combine the output of multiple decision trees in order to arrive at the final predicted outcome. A benefit of these models is that there is a reduced risk of overfitting the data, and it is easy to determine feature importance.Additionally, Random forest can handle large amounts of training data. [2]
Gradient Boosting builds decision trees one at a time, and each tree helps to correct the errors made by the previously trained tree . Gradient boosting(GBM) focuses step by step, which works well with a unbalanced data set. This leads to why Gradient Boosting was chosen. As mentioned in the data set is unbalanced and the Gradient Boosting Model seems to work as a anomaly detector when the data is very unbalanced. GBM had a learning to rank approach, in which it can rank and improve on models, which is why it can detect anomalies with the highest precision.

The task states to chose one machine learning model, however our group decided to implement three. Both Random Forest and Gradient Boosting had benefits that would work with the data set. For example, Random Forest works well with large data sets and determining feature importance. Moreover, Gradient Boosting works well with unbalanced data. By implementing both, out group was able to compare the performance for both models and determine which one performed better. Neural Nets were also implemented. The benefits of Neural Nets is that it is reliable with many features. However, Neural Nets are computationally very expensive and time consuming to train with CPUs. Additionally, Neural Nets are considered black boxes, which means the influence of the independent variable on the dependent variable cannot be determined. This was a huge con, due to the fact that our task included to determine which independent variable was more important. Neural Nets also work best with large amounts of data, a million or more data points. The data set was not large enough to compute a good Neural Net model.

3.2 Logistic Regression

The first model created was the Logistic Regression model. This was created by using the library sk.linear_model. The model was fit using the training set data into x and y train. The x train data was scaled for this model using the StandardScaler() function. The sample of code below shows how the Logistic Model was implemented.

```
logReg = LogisticRegression()
logReg.fit(x_TrainScale, y_Train)
```

Once the model was created, the predicted validation and predicted test scores were determined. The code below shows how the predicted values were determined.

```
predictedVAL = logReg.predict(x_ValScale)
predictedTEST = logReg.predict(x_TestScale)
```

Using these values a classification report was printed using the predicted testing values. The classification report was determined created using the the classification_report method imported from the sklearn.metrics library. Additionally, the accuracy of the testing and validation set was determined using the predicted values obtained. The accuracy values were determined from the accuracy_score method imported from the sklearn.metrics library. Furthermore the feature importance's were printed out for all the features. Below is the code to determine the feature importance.

```
importanceLR = pd.DataFrame('feature': list(x_Train.columns), 'importance':
logReg.coef_[0]). sort_values('importance', ascending = False)
```

Another Logistic Model was created using SMOTE, which re-scales the data. This was done to obtain better accuracy. However, the results were not more accurate using the SMOTE function.

3.3 Random Forest Model

The Random Forest Model was created using the sklearn.ensemble library. The model was fit using the split training set x and y train. However, a scaled x training set was not used unlike the Logistic Regression model. This is due to the fact that model performed better without using the scaled values. 64 trees were chosen for the model. The following code sample shows the implementation of Random Forest.

```
rf = RandomForestClassifier(n_estimators = 64)
rf = RandomForestClassifier(n_estimators = 64)
```

Similarly to the Logistic Model, prediction values for the validation set and testing set were determined. This was done by using the predict() method. Additionally, classification report and accuracy scores were obtained in a similar manner to the previous model. The following code demonstrates the prediction of the values and creation of the classification report.

```
predictedRFVAL = rf.predict(x_Val)
predictedRFTEST = rf.predict(x_Test)
accuracyRFVAL = metrics.accuracy_score(y_Val, predictedRFVAL)
accuracyRFTEST = metrics.accuracy_score(y_Test, predictedRFTEST)
print(classification_report(y_Val, rf.predict(x_Val)))
```

Additionally, the important features found for Random Forest model. The two least important features were removed from the data set, and another Random Forest Model was created. This was done to test if the accuracy would chance if the least important feautures are removed.

3.4 Gradient Boosting Models

Two Gradient Boosting Models were created. These models were created using the sklearn.ensemble library. The first one was created to test the classifier's performance at different learning rates . The model was fit using the training set, which was split up into x and y train. The following sample of code shows how the first Gradient Boosting model was implemented.

```
for learning_rate in lr_list:
gb_clf = GradientBoostingClassifier(n_estimators=20, learning_rate=learning_rate,
max_features=2, max_depth=2, random_state=0) gb_clf.fit(x_Train, y_Train)
```

The model printed learning rate, accuracy score of the training and validation test. The learning rates were in a range from 0.05 and 1. Furthermore, the most important value to look for is the accuracy score for the validation data set. The learning rate that produces the greatest accuracy score for the validation will be the learning rate used to further evaluate the classifier model. In this case, the learning rate 0.5 gave the best performance in the validation data set, with an accuracy score of 0.936.

```
Learning rate: 0.5
Accuracy score (training): 0.937
Accuracy score (validation): 0.936
```

Furthermore, a second Gradient Boosting model was created to further evaluate and determine classification scores, accuracy, and the importance of features. Like the previous model, this model was fit using the x and y train values. Using the predict method, prediction values for the validation and testing values were determined.These predicted values were later used to create a Confusion Matrix and Classification Report. Both were built using the sklearn.metrics library. The following sample of code shows how the second Gradient Boosting model was implemented.

```
gb_clf2 = GradientBoostingClassifier(n_estimators=20, learning_rate=0.5,
max_features=2, max_depth=2, random_state=0) gb_clf2.fit(x_Train, y_Train) predictions
= gb_clf2.predict(x_Val)
```

Like the previous models, each feature and it's importance was printed out. The values were sorted by importance, with the more prevalent features being printed first.

3.5 Further Information

The software Jupyter Notebook was used, by use of Google Research Colaboratory (colab.research.google.com). The code was typed in Python and multiple python libraries were used like Numpy and Seaborn. To perform the Random Tree and Logistic Regression experiments, a 2018 MacBook Air with 1.6 GHz Dual-Core Intel Core i5 processor in version 10.15.4 of macOS Catalina was used.

While experimenting, only one model stood out as taking significantly more time. The Random Forest Model that was run with SMOTE overfitted data and with 1000 estimators took around two minutes to run, while all other models took less than 10 seconds. This model did not give a significantly higher accuracy, so it was not necessary to keep using it.

4. Model Analysis and Comparison

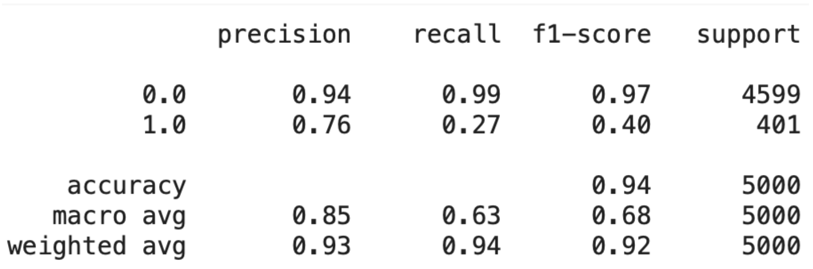
4.1 Logistic Regression

The results of the three different models we conducted during this experiment were slightly different. The first of the three models we did was the Logistic Regression Model. It presented a .9311 accuracy on test set, .934 accuracy on validation set.

To reach these results, we initially had to replace the null values as described in the Data Analysis/Preprocessing section. When we replaced the null values with the mean; we got an accuracy of .9306. We then wanted to see, if replacing the null values with the median of the column would result in a greater accuracy. When the null values were replaced with the median, the accuracy came out to be .9311. As assumed the median did raise the accuracy of the dataset, even though the value was not significantly greater compared to the accuracy of the mean, we decided to conduct the rest of the logistic regression analysis by replacing the null vales with median.

We printed a Classification report that summarized the important information regarding the dataset like precision, recall, and f1 score.

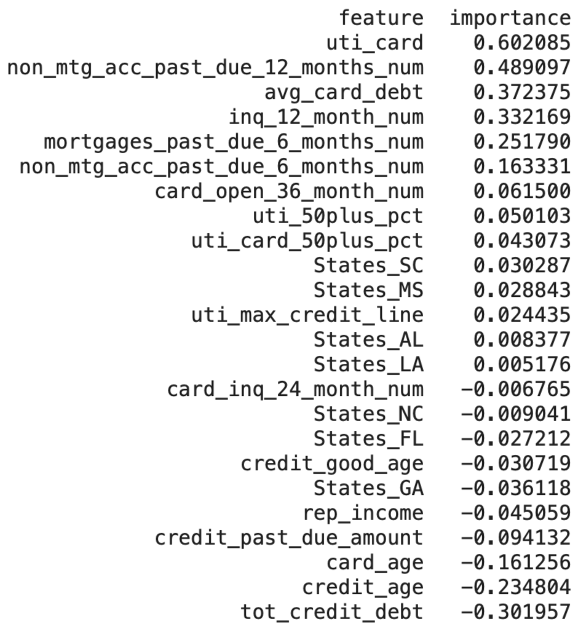
```
print(classification_report(y_Test, logReg.predict(x_TestScale)))
```



```
              precision    recall  f1-score   support

         0.0       0.94      0.99      0.97      4599
         1.0       0.76      0.27      0.40       401

    accuracy                           0.94      5000
   macro avg       0.85      0.63      0.68      5000
weighted avg       0.93      0.94      0.92      5000
```

Logistic Regression Classification Report

We also decided to print of a table of Feature Important for Logistic Regression that helped visualize which variables played an important role in the credit card approval process. The numerical values to the right of the 20 features signifies the importance of that specific variable. When the different variable are thoroughly analyzed, you notice that one variable holds a small or large significance compared to the next in determining the output which is the response. For example, the 'non_mtg_acc_past_due_12_months_num' variable can give more information on whether a candidate should be approved for a credit card based on their delinquent activities compared to the 'uti_max_credit_line' variable that just measures the utilization of the credit account.



```
                           feature  importance
                          uti_card    0.602085
non_mtg_acc_past_due_12_months_num    0.489097
                     avg_card_debt    0.372375
                  inq_12_month_num    0.332169
   mortgages_past_due_6_months_num    0.251790
 non_mtg_acc_past_due_6_months_num    0.163331
             card_open_36_month_num   0.061500
                   uti_50plus_pct     0.050103
              uti_card_50plus_pct     0.043073
                         States_SC    0.030287
                         States_MS    0.028843
               uti_max_credit_line    0.024435
                         States_AL    0.008377
                         States_LA    0.005176
              card_inq_24_month_num  -0.006765
                         States_NC   -0.009041
                         States_FL   -0.027212
                   credit_good_age   -0.030719
                         States_GA   -0.036118
                        rep_income   -0.045059
            credit_past_due_amount   -0.094132
                          card_age   -0.161256
                        credit_age   -0.234804
                    tot_credit_debt  -0.301957
```

Logistic Regression Feature Importance

The third and the final analysis we did with Logistic Regression was the Confusion Matrices with the Not Defaulted and Defaulted variables.

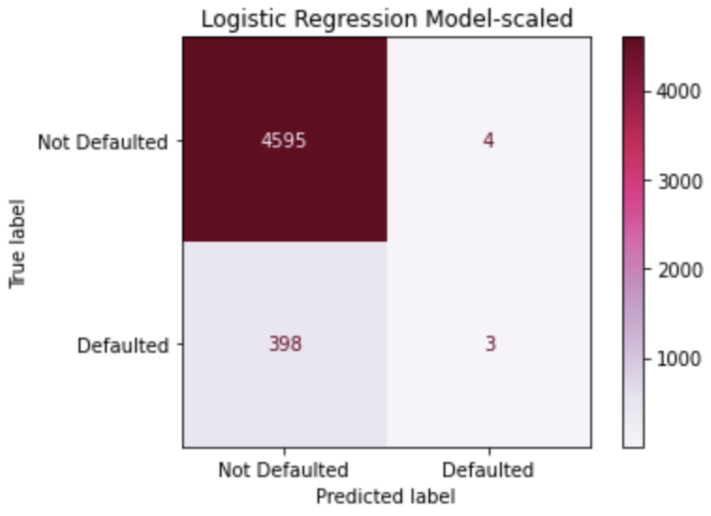
SMOTE is an oversampling technique that generates synthetic samples from the minority class. It is used to obtain a synthetically class-balanced or nearly class-balanced training set, which is then used to train the classifier. This is the process we followed to obtain the test set and the validation set accuracies. We had to rescale the sample size to obtain the following results:

plot(logReg, x_Test, y_Test, "Logistic Regression Model-scaled") print("Test set accuracy:") print(accuracyLRTEST)

Before SMOTE the accuracy was .9311, after SMOTE .87.

print("Validation set accuracy:") print(accuracyLRVAL)

Before SMOTE : .934, after SMOTE: .8,



```
Test set accuracy:
0.935
Validation set accuracy:
0.942
```

Logistic Regression Confusion Matrix

4.2 Random Forest

Similar to the analysis we conducted for Logistic Regression, we had to follow the same steps to analyze the dataset using Random Forest and Gradient Boosting machine learning models.

For the Random Forest model, we started off by instantiating the model with 1000 decision trees and training the model on the training dataset.

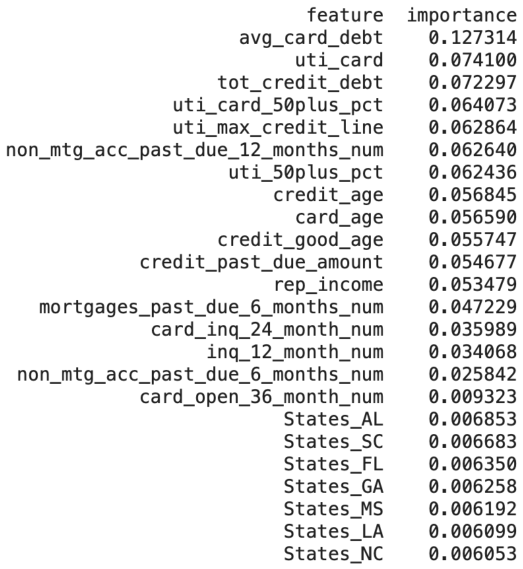
accuracyRFVAL = metrics.accuracy_score(y_Val, predictedRFVAL) accuracyRFTEST = metrics.accuracy_score(y_Test, predictedRFTEST)

We obtained a .99 accuracy on the testing set and a .942 accuracy on the validation set.

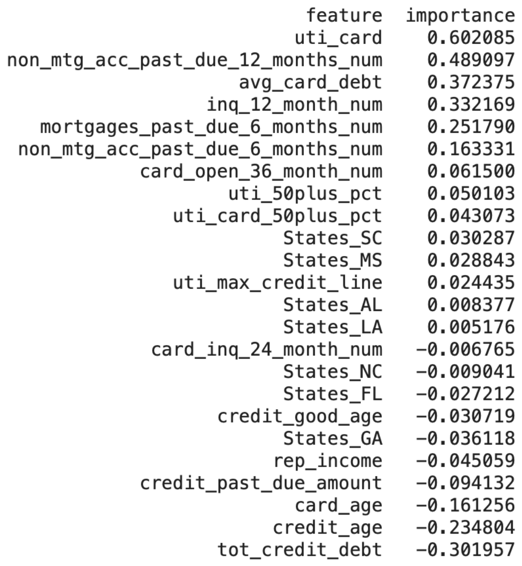
```
              precision    recall  f1-score   support

         0.0       0.94      1.00      0.97      2778
         1.0       0.83      0.27      0.40       222

    accuracy                           0.94      3000
   macro avg       0.89      0.63      0.69      3000
weighted avg       0.94      0.94      0.93      3000
```

Random Forest Classification Report

We then continued to extract the feature importances :



```
                           feature  importance
                      avg_card_debt    0.127314
                           uti_card    0.074100
                     tot_credit_debt    0.072297
                 uti_card_50plus_pct    0.064073
                uti_max_credit_line    0.062864
non_mtg_acc_past_due_12_months_num    0.062640
                     uti_50plus_pct    0.062436
                         credit_age    0.056845
                           card_age    0.056590
                    credit_good_age    0.055747
             credit_past_due_amount    0.054677
                         rep_income    0.053479
    mortgages_past_due_6_months_num    0.047229
              card_inq_24_month_num    0.035989
                   inq_12_month_num    0.034068
 non_mtg_acc_past_due_6_months_num    0.025842
              card_open_36_month_num    0.009323
                          States_AL    0.006853
                          States_SC    0.006683
                          States_FL    0.006350
                          States_GA    0.006258
                          States_MS    0.006192
                          States_LA    0.006099
                          States_NC    0.006053
```



```
                           feature  importance
                           uti_card    0.602085
non_mtg_acc_past_due_12_months_num    0.489097
                      avg_card_debt    0.372375
                   inq_12_month_num    0.332169
    mortgages_past_due_6_months_num    0.251790
 non_mtg_acc_past_due_6_months_num    0.163331
              card_open_36_month_num    0.061500
                     uti_50plus_pct    0.050103
                 uti_card_50plus_pct    0.043073
                          States_SC    0.030287
                          States_MS    0.028843
                uti_max_credit_line    0.024435
                          States_AL    0.008377
                          States_LA    0.005176
              card_inq_24_month_num   -0.006765
                          States_NC   -0.009041
                          States_FL   -0.027212
                    credit_good_age   -0.030719
                          States_GA   -0.036118
                         rep_income   -0.045059
             credit_past_due_amount   -0.094132
                           card_age   -0.161256
                         credit_age   -0.234804
                     tot_credit_debt   -0.301957
```

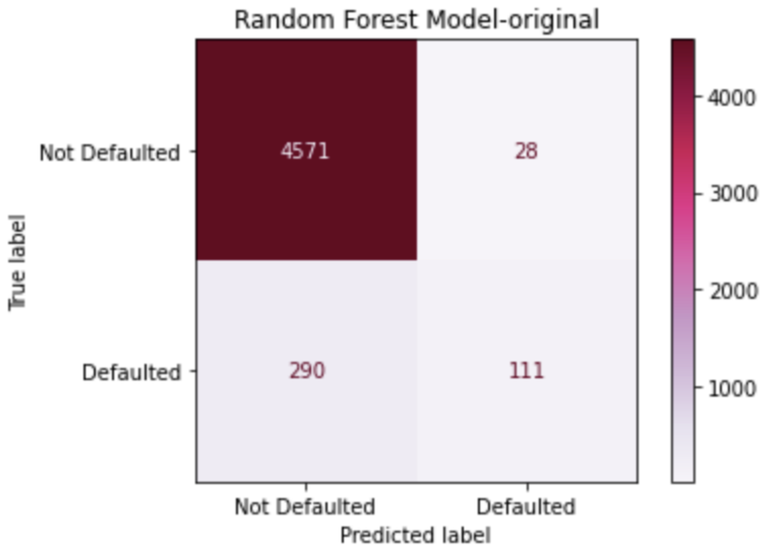
Random Forest compared to Logistic Regression Feature Importance

The order of the feature importance of the Random Forest model compared to the order of the Logistic Regression model varies slightly. A side by side comparison of the two lists will show us that the Logistic regression model has the states included between other variable and considers that information important. For example States_SC, and States_MS are included in the middle of the list, with other variables that SHOULD be considered important toward the end. Random Forest model does the complete opposite and considers all the non-state variables more important and ranks with their significance before adding the stated to the list.

This is a more accurate representation of the Feature Importance because the residence of the applicant is less significant than the total credit debt which the Logistic Regression goes against. While the accuracy values vary for the different data set for each model, I believe that this is a good representation of while model is considered more accurate.

Random Forest machine learning model does the analysis based on the right set of feature importance, while Logistic Regression does not.

The accuracy values for the Confusion Matrices are also very similar between both the models. They only vary by a hundredth of a decimal place.

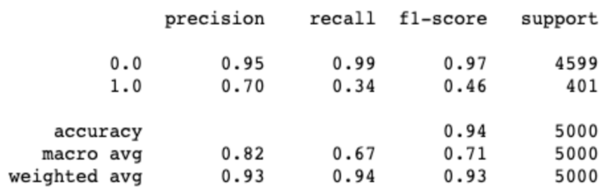


```
Test set accuracy:
0.9364
Validation set accuracy:
0.9416666666666667
```

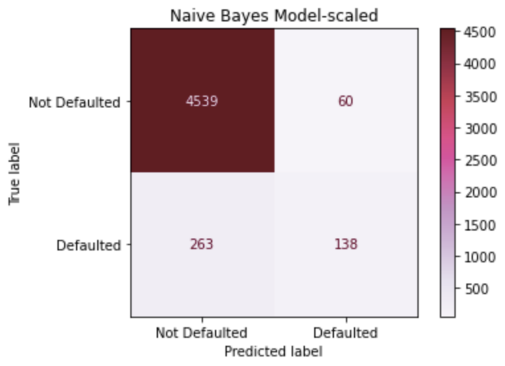
Random Forest Confusion Matrix

4.3 Naive Bayes Classifier

The Naive Bayes Classifiers are types of classifiers that apply Bayes' theorem to produce a probabilistic classifier. It is a simple type of Bayesian Network Model that assigns class labels to problem instances. This model assumes complete independence between each variable, which often causes the model to be over-simplistic. Because of this it's usually outperformed by other classification models such as boosted trees and random forest models. In addition, because of the assumed independence in these variables, the use of feature selection with this model is unwarranted. Do not take this to mean that it is not an effective model however. Historically, the Naive Bayes models can be very effective, particularly when the data that is being analyzed is relatively simple. For this scenario we used the GaussianNB Model from sklearn. The accuracy score on the training dataset was 0.9319 and the accuracy score on the testing dataset was 0.9354. The macro average f1 score was 0.71. That particular f1 score is the highest f1 score that we saw while creating models. The average precision was 0.82, while the average recall was 0.67. As you can see from the data, the model performed very well.



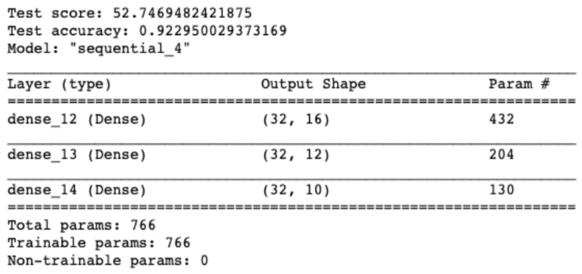
```
              precision    recall  f1-score   support

         0.0       0.95      0.99      0.97      4599
         1.0       0.70      0.34      0.46       401

    accuracy                           0.94      5000
   macro avg       0.82      0.67      0.71      5000
weighted avg       0.93      0.94      0.93      5000
```



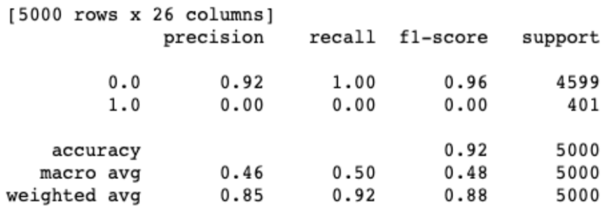
Naive Bayes Confusion Matrix and Classification Report

4.4 Neural Networks

Neural Nets are models that are typically used for complex data sets and work by running data through multiple layers of interconnected units in order to process and analyze information. This is particularly useful for data that features non-linear relationships between the explanatory and response variables. For this model we created a 3 Layer Neural Net using Keras in python and ran it with 30 epochs. The accuracy score for this model was 0.92, with an average precision of 0.46, an average recall of 0.50, and an average f1-score of 0.48. While the accuracy score is very high, we can look at the F1 score as well as the precision and recall to confirm that this is not the most appropriate model to be working with. This makes sense overall, as the data is not particularly complex. Not only this, but due to the binary response variable, we could likely infer that a classification model would be more appropriate. This inference can be backed up by the models that we have run thus far and the data that we have collected.



```
Test score: 52.7469482421875
Test accuracy: 0.922950029373169
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (32, 16)                  432
_________________________________________________________________
dense_13 (Dense)             (32, 12)                  204
_________________________________________________________________
dense_14 (Dense)             (32, 10)                  130
=================================================================
Total params: 766
Trainable params: 766
Non-trainable params: 0
```



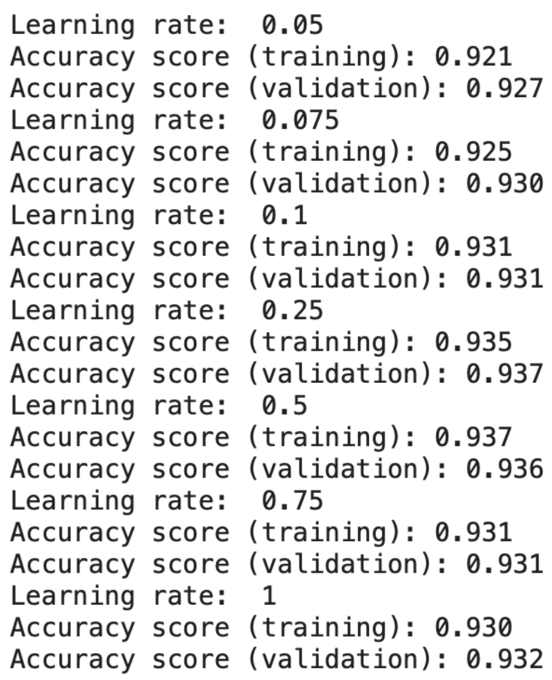
```
[5000 rows x 26 columns]
              precision    recall  f1-score   support

         0.0       0.92      1.00      0.96      4599
         1.0       0.00      0.00      0.00       401

    accuracy                           0.92      5000
   macro avg       0.46      0.50      0.48      5000
weighted avg       0.85      0.92      0.88      5000
```

Neural Net Classification Report

4.5 Gradient Boosting Models

Finally onto our most accurate model for the Credit Card Approval Dataset.

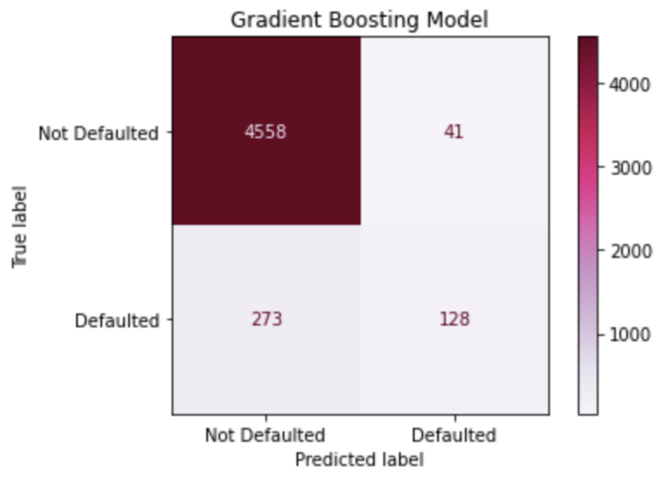
The Gradient Boosting Machine is a powerful ensemble machine learning algorithm that uses decision trees. We experimented with many different learning rate values to output the most accurate training and validation scores. As you can see from the picture below, we incremented the learning rate values by 0.025. Among the many combination, the accuracy when the learning rate is 0.5 is the highest for testing set.



```
Learning rate:  0.05
Accuracy score (training): 0.921
Accuracy score (validation): 0.927
Learning rate:  0.075
Accuracy score (training): 0.925
Accuracy score (validation): 0.930
Learning rate:  0.1
Accuracy score (training): 0.931
Accuracy score (validation): 0.931
Learning rate:  0.25
Accuracy score (training): 0.935
Accuracy score (validation): 0.937
Learning rate:  0.5
Accuracy score (training): 0.937
Accuracy score (validation): 0.936
Learning rate:  0.75
Accuracy score (training): 0.931
Accuracy score (validation): 0.931
Learning rate:  1
Accuracy score (training): 0.930
Accuracy score (validation): 0.932
```

Gradient Boosting Learning Rate/Accuracy Scores

We then continued to output the Confusion Matrix for the Gradient Boosting Model which showed that this model produced the highest accuracy.



```
Test set accuracy:
0.9372
Validation set accuracy:
0.9363333333333334
```

Gradient Boosting Confusion Matrix

4.6 Further Analysis

Overall, the results we obtained are very significant because this is the most important part of our analysis. The simple Logistic Regressing and the machine learning Random Forest model, were our two main focus on analyzing the data. The results we got from each model were what helped us determine the importance of the variables and the weight they had in playing a role in determine the results or the output variable which was determining if the credit card would be issued. We outputted quite a few different tables and graphs, so the analysis and each ones specific significance will be under the picture in the form of a caption.

Our main focus was to conduct the regression and machine learning analysis through Sklearn, but to compare the different results we did conduct an experiment using TensorFlow and the results has a slight variance. After observing the different results that were the output of the Machine Learning models, the results varied from one framework to another but the difference was not very significant. All the outputs varied by a few decimal places. This was the same case between different models. Comparing the results of the different models to different frameworks, we came to a conclusion that the different models like Logistic Regression and Random Forest tended to have results that were closer in value, varying by hundredth o decimal place. While, different frameworks with the same model had a more different result. The output started to vary in the tenths decimal place and after some research, we discovered that the different frameworks have different featured included within that results in varying outputs.

It is very possible that the results are misleading due to the data, the way they are partitioned, the different machine learning models we used, and the way the models are configured. The data that is imported from the raw file is always not in its best form to be used for analysis. Before any regression or analysis is done, we trained the data set and cleaned it up to get rid of the null values. We had to find different ways to refine the data set without affecting the results when regression was done. This is hard to do with so many rows of data, but by replacing the null values with the mean value of the columns was one way that we ensured that the data stayed similar to what was given to us. As for the problem of different machine learning schemes affecting the results, we decided to accurately depict the results by using two different models for the machine learning aspect. This helped us compare the results of one model to another and get a more accurate understanding of the results. As for the most preventable part of the analysis, partitioning of the data, we separated it into X and Y for training, testing, and validation sets. More information can be found in the Data Analysis/Preprocessing section of this report.

5. Application

Describe how you would use it to make decisions on future credit card applications.
When a customer is looking to apply for a credit card, they should give the bank as many of the features as they have, marking the rest as null. Then, their profile would be run through our Gradient Boosting model, predicting either a 0 or 1. A 1 means the customer is likely to default their account in the first 18 months, meaning they do not make payments

for three consecutive months. We would be more inclined to deny these customers. A 0 means the customer is not predicted to default their account, so we would be more inclined to accept these customers.

The model is a good start on being able to make decisions on future credit cards. However, it is not 100 percent accurate. I do believe worthy applicants may lose the chance to obtain a credit card with the model. Therefore, I recommend to keep having jobs that manually check information in addition to the model. Furthermore, more models should be tested to determine if we can get a more accurate model.

Do customers who already have an account with the financial institution receive any favorable treatment in your model? Support your answer with appropriate analysis.
The feature that deals with whether a customer has a preexisting account in the financial institution was dropped from the data set in the beginning. In the data preprocessing , it was determined that 'ind_acc_XYZ', which is previous account holders, had the lowest importance with the value of 0.007916. To improve the accuracy of our models it was dropped. Therefore, our model does not favor customers with a preexisting account in the financial institution.

Suppose a credit card application is rejected using your model, and the applicant asks you to provide an explanation on why it was rejected. How would you explain the results to the customer?
The most accurate model, Gradient Boosting, only has an accuracy of 0.9372. It is fairly accurate, however there is a chance where a credit card application that deserved a card is rejected. I would recommend, until a better or more accurate model is created, to have in-person jobs that will review the credit card applications for applications that do seem worthy. I would tell my customer, the model stated that your credit card application has been denied, but someone will look into and determine if you will be granted a credit card.

Wells Fargo Data Science Competition

Kavya Ahuja, Lakshmi Yetukuri, Hannah Adamson

Goal

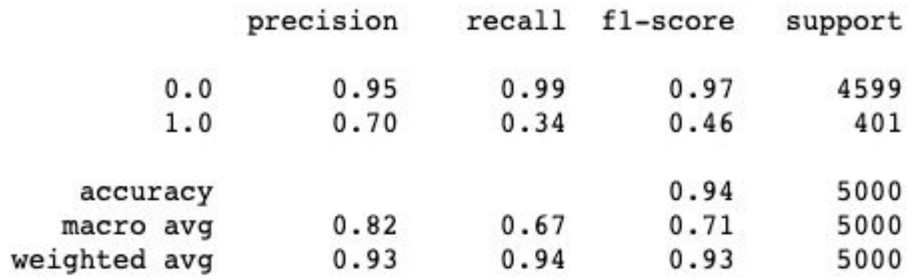
- Given historical data, build models to review credit card applications and determine if they should be approved or denied
- To accomplish this goal, we were provided with three datasets (csv files) of historical data, containing the default indicator value as well as 20 predictor values:
  - Training set with 20,000 account entries
  - Testing set with 5,000 account entries
  - Validation set with 3,000 account entries

Data

- It was necessary to import these files into a Pandas dataframe
- Two predictor value columns had null values. We replaced the null values with the mean value of the column.
    - ‘Uti_card_50plus_pct’- 2055, 499, and 297 null values across the training, test, and validation sets
    - ‘Rep_income’- 1570, 383, and 253 null values across the training, test, and validation sets
- Data had to be scaled to be used in the logistic regression model
    - Used StandardScaler() from the scikit learn preprocessing package
    - Standard score = (score - mean)/std.dev.
- Experimented with overfitting the data using SMOTE(Synthetic Minority Over-Sampling Technique) from imbalanced-learn library

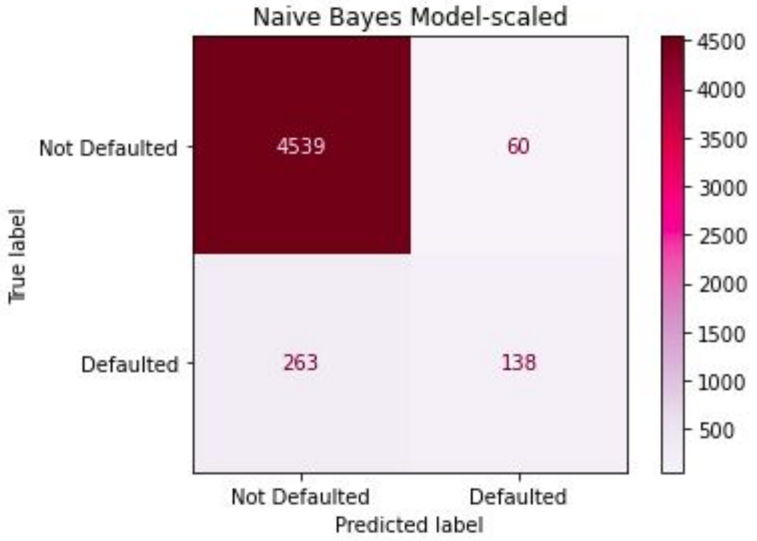
Naive Bayes

- The Naive Bayes Classifiers are types of classifier that apply Bayes' theorem to produce a probabilistic classifier. It is a simple type of Bayesian Network Model that assigns class labels to problem instances.
- For this scenario we used the GaussianNB Model from sklearn and had accurate results. The accuracy score on the training dataset was 0.9319 and the accuracy score on the testing dataset was 0.9354



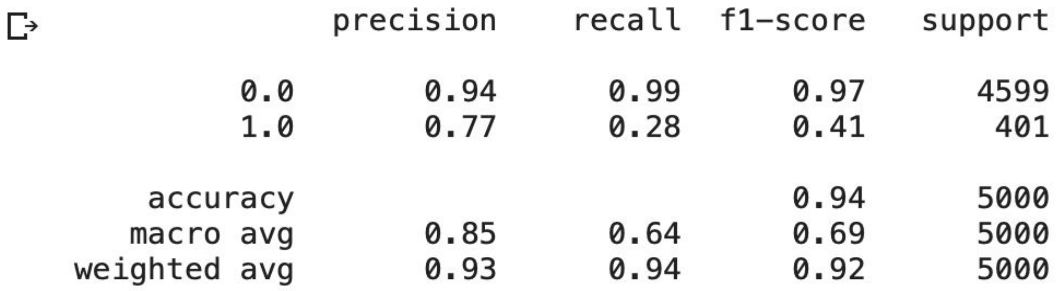
```
              precision    recall  f1-score   support

         0.0       0.95      0.99      0.97      4599
         1.0       0.70      0.34      0.46       401

    accuracy                           0.94      5000
   macro avg       0.82      0.67      0.71      5000
weighted avg       0.93      0.94      0.93      5000
```

Naive Bayes Confusion Matrix



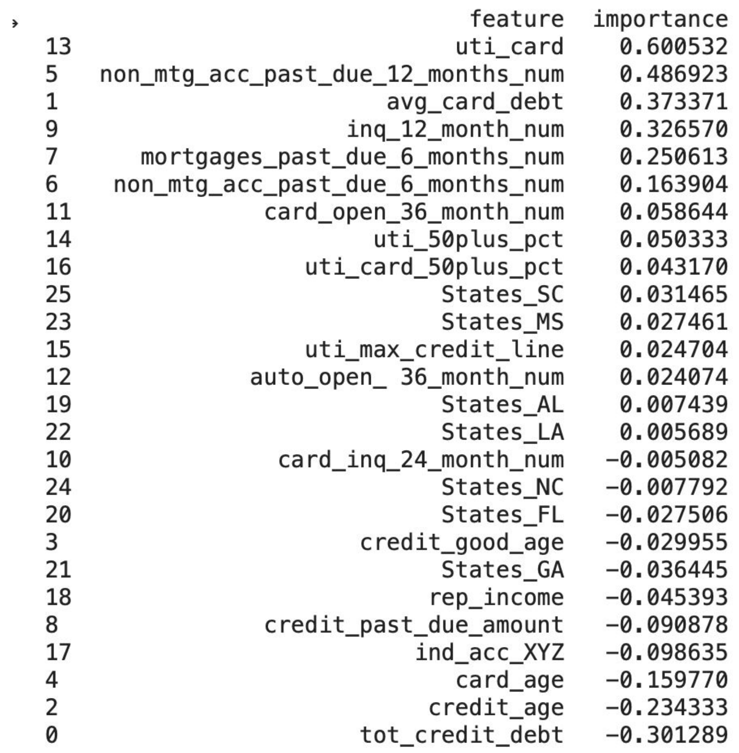
Logistic Regression Model

- Logistic regression frameworks are used for predictive analysis when the dependent Y value is binary, as it was in our problem. The purpose is to calculate the probability of a binary event occurring, based on occurrences in the training set.
- We used the LogisticRegression() model from sklearn.linear_model, and trained this model on the scaled data
- All 20 features were incorporated when training this model, resulting in a test set accuracy of 0.9354, and a validation set accuracy of 0.942



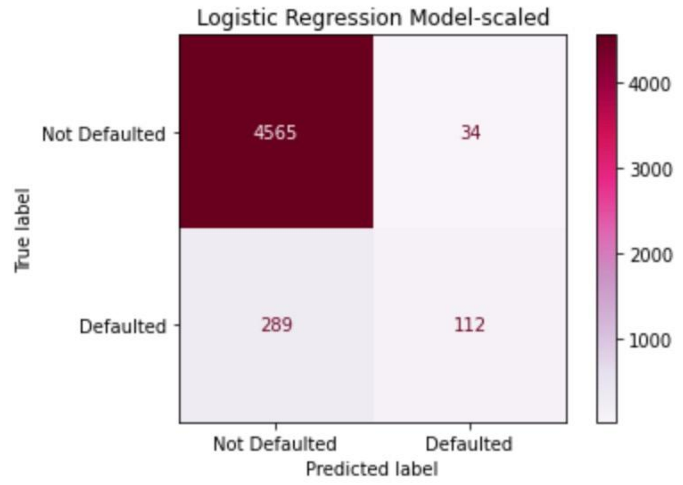
```
              precision    recall  f1-score   support

         0.0       0.94      0.99      0.97      4599
         1.0       0.77      0.28      0.41       401

    accuracy                           0.94      5000
   macro avg       0.85      0.64      0.69      5000
weighted avg       0.93      0.94      0.92      5000
```

Feature Importance



```
                                  feature  importance
13                               uti_card    0.600532
5   non_mtg_acc_past_due_12_months_num    0.486923
1                            avg_card_debt    0.373371
9                          inq_12_month_num    0.326570
7           mortgages_past_due_6_months_num    0.250613
6        non_mtg_acc_past_due_6_months_num    0.163904
11                    card_open_36_month_num    0.058644
14                           uti_50plus_pct    0.050333
16                      uti_card_50plus_pct    0.043170
25                                States_SC    0.031465
23                                States_MS    0.027461
15                      uti_max_credit_line    0.024704
12                  auto_open_ 36_month_num    0.024074
19                                States_AL    0.007439
22                                States_LA    0.005689
10                     card_inq_24_month_num   -0.005082
24                                States_NC   -0.007792
20                                States_FL   -0.027506
3                           credit_good_age   -0.029955
21                                States_GA   -0.036445
18                               rep_income   -0.045393
8                    credit_past_due_amount   -0.090878
17                              ind_acc_XYZ   -0.098635
4                                  card_age   -0.159770
2                                credit_age   -0.234333
0                            tot_credit_debt   -0.301289
```

Logistic Regression Model Confusion Matrix

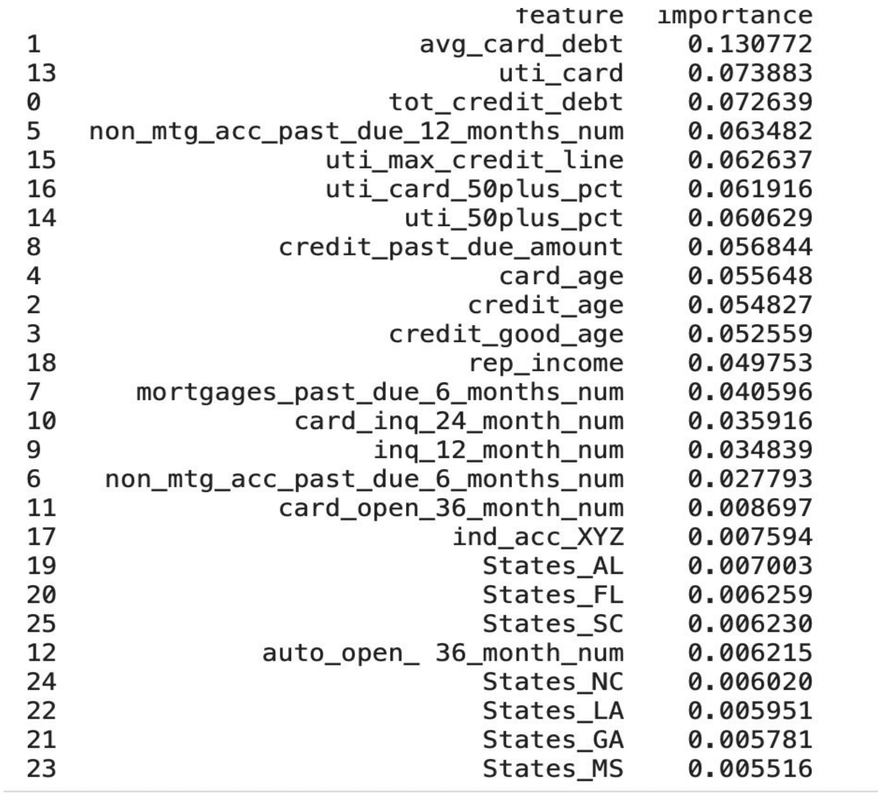


```
Test set accuracy:
0.9354
Validation set accuracy:
0.942
```

Random Forest Model

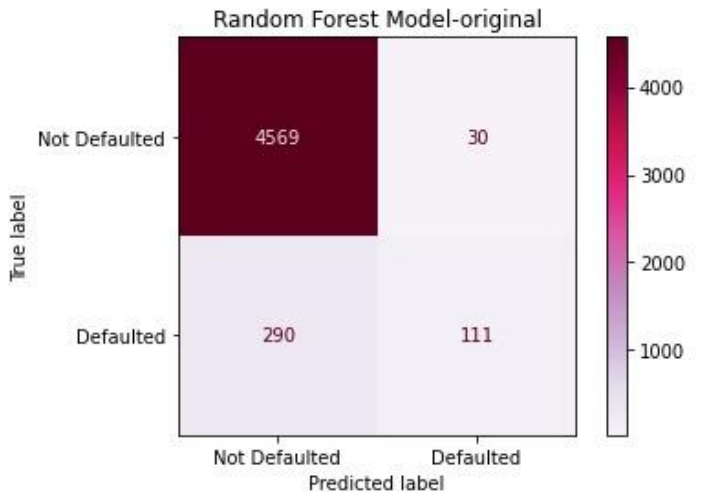
- Random Forest models combine the output of multiple decision trees in order to arrive at the final predicted outcome. A benefit of these models is that there is a reduced risk of overfitting the data, and it is easy to determine feature importance.
  - Random Forest can handle large amounts of training data.
- We used the RandomForestClassifier from sklearn.ensemble with 64 trees
- The model was first trained on the dataset of all 20 features, giving a test set accuracy of 0.936 and a validation set accuracy of 0.939
- Exploring the feature importance showed that the ‘auto_open_ 36_month_num’ and ‘ind_acc_XYZ’ features were the two least important features, so we removed and retrained the model to get a very slight increase in accuracy: test set accuracy of 0.9362 and a validation set accuracy of 0.942

Feature Importance

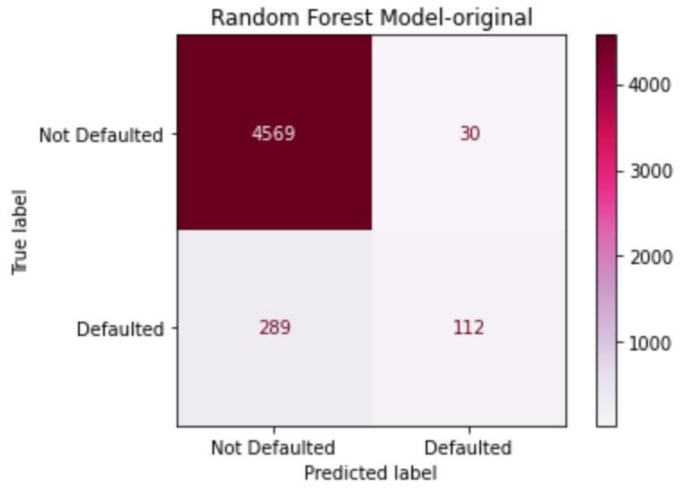


```
                                   feature  importance
1                             avg_card_debt    0.130772
13                                 uti_card    0.073883
0                            tot_credit_debt   0.072639
5   non_mtg_acc_past_due_12_months_num    0.063482
15                       uti_max_credit_line   0.062637
16                       uti_card_50plus_pct   0.061916
14                            uti_50plus_pct   0.060629
8                     credit_past_due_amount   0.056844
4                                  card_age    0.055648
2                                credit_age    0.054827
3                           credit_good_age    0.052559
18                               rep_income    0.049753
7        mortgages_past_due_6_months_num    0.040596
10                    card_inq_24_month_num    0.035916
9                          inq_12_month_num    0.034839
6     non_mtg_acc_past_due_6_months_num    0.027793
11                    card_open_36_month_num   0.008697
17                              ind_acc_XYZ    0.007594
19                                States_AL    0.007003
20                                States_FL    0.006259
25                                States_SC    0.006230
12                  auto_open_ 36_month_num    0.006215
24                                States_NC    0.006020
22                                States_LA    0.005951
21                                States_GA    0.005781
23                                States_MS    0.005516
```

Random Forest Confusion Matrices

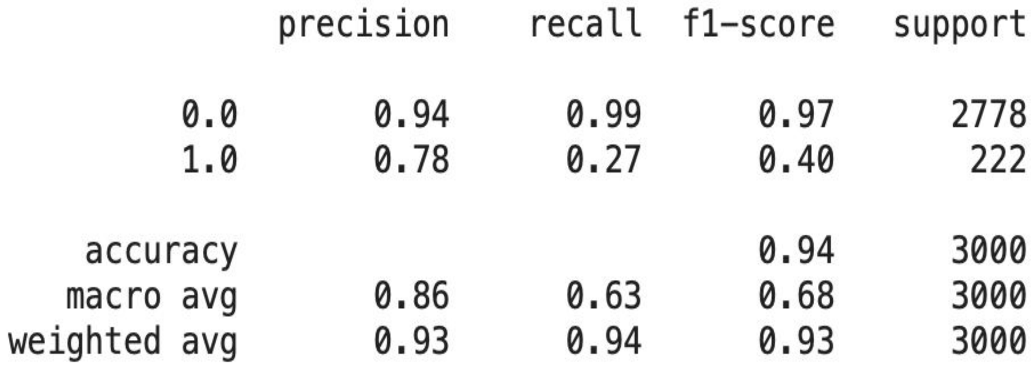


```
Test set accuracy:
0.936
Validation set accuracy:
0.9393333333333334
```



```
Test set accuracy:
0.9362
Validation set accuracy:
0.942
```

Classification Report

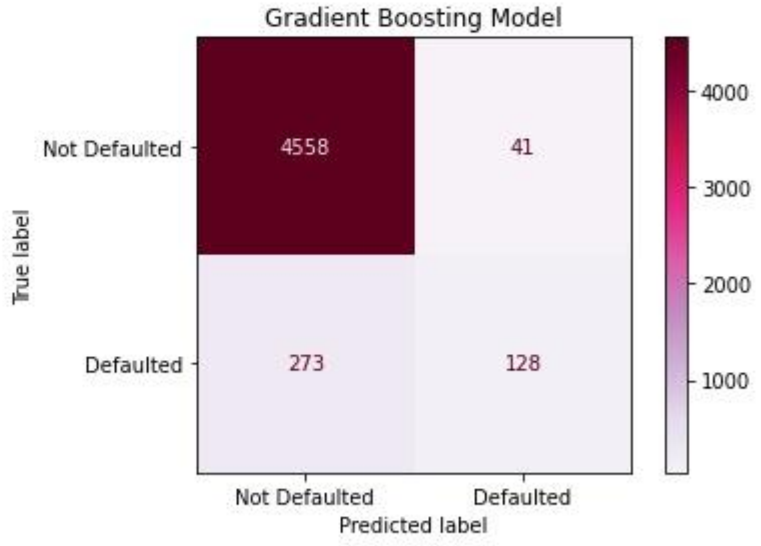


```
              precision    recall  f1-score   support

         0.0       0.94      0.99      0.97      2778
         1.0       0.78      0.27      0.40       222

    accuracy                           0.94      3000
   macro avg       0.86      0.63      0.68      3000
weighted avg       0.93      0.94      0.93      3000
```

Gradient Boosting

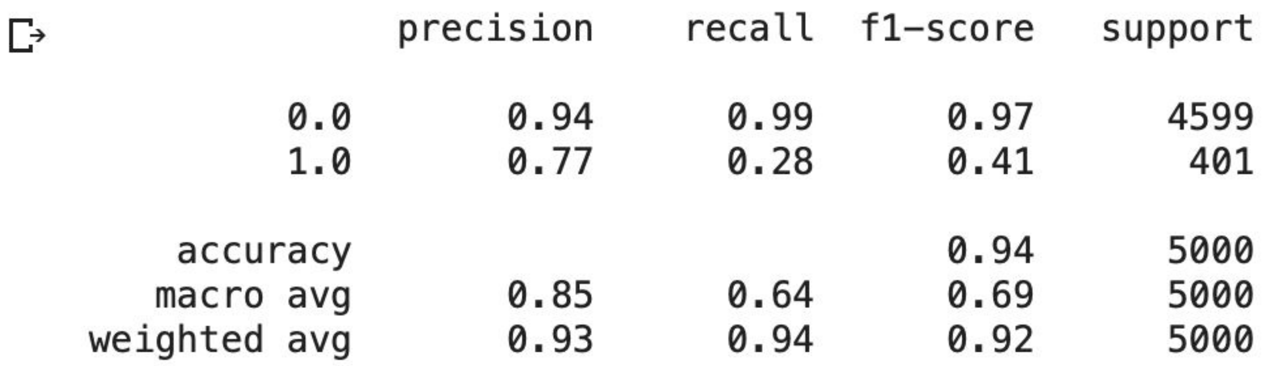
- Gradient Boosting builds decision trees one at a time, and each tree helps to correct the errors made by the previously trained tree . Gradient boosting(GBM) focuses step by step, which works well with a unbalanced data set.
- We used the GradientBoostingClassifier from sklearn.ensemble
- The Gradient Boosting Model was created to test the classifier’s performance at different learning levels. The best performance came at a learning rate of 0.5, with a test set accuracy of 0.937 and a validation set accuracy of 0.936
- We were able to find this optimal learning rate by iterating through learning rates between .05 and 1.

Gradient Boosting Confusion Matrix



```
Test set accuracy:
0.9372
Validation set accuracy:
0.9363333333333334
```

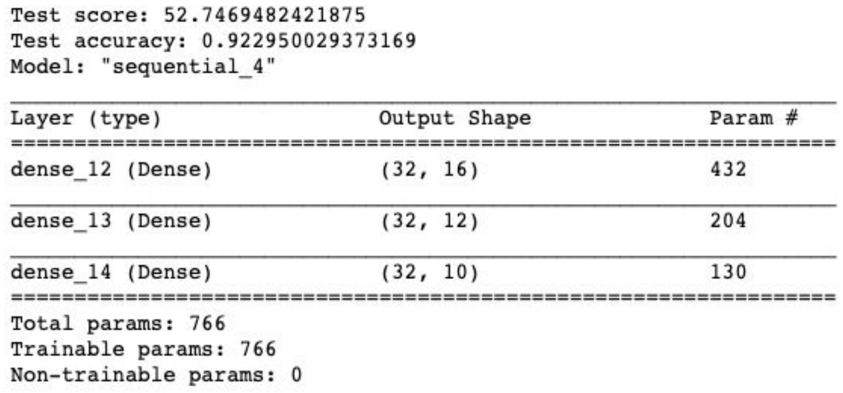
Gradient Boosting



```
              precision    recall  f1-score   support

         0.0       0.94      0.99      0.97      4599
         1.0       0.77      0.28      0.41       401

    accuracy                           0.94      5000
   macro avg       0.85      0.64      0.69      5000
weighted avg       0.93      0.94      0.92      5000
```

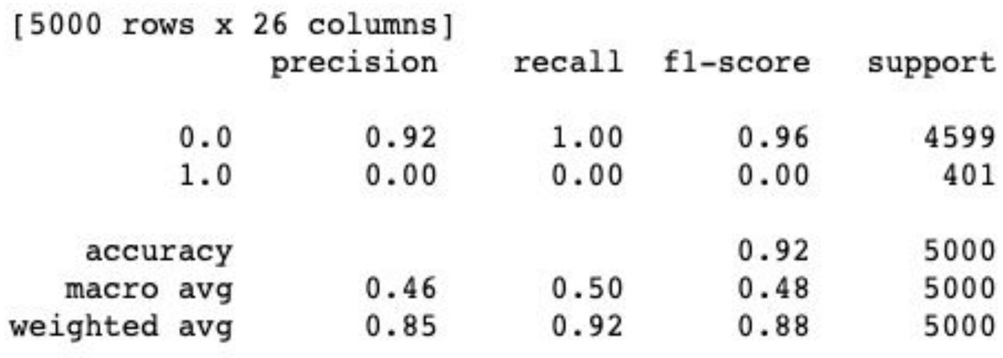
Neural Nets

- Neural Networks are models that are typically used for complex data sets and work by running data through multiple layers of interconnected units in order to process and analyze information.
- For this model we created a 3 Layer Neural Net using Keras in python and ran it with 30 epochs. We had accurate results, although still not quite as accurate as the other models.



```
Test score: 52.7469482421875
Test accuracy: 0.922950029373169
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (32, 16)                  432
_________________________________________________________________
dense_13 (Dense)             (32, 12)                  204
_________________________________________________________________
dense_14 (Dense)             (32, 10)                  130
=================================================================
Total params: 766
Trainable params: 766
Non-trainable params: 0
```

Neural Nets



```
[5000 rows x 26 columns]
              precision    recall  f1-score   support

         0.0       0.92      1.00      0.96      4599
         1.0       0.00      0.00      0.00       401

    accuracy                           0.92      5000
   macro avg       0.46      0.50      0.48      5000
weighted avg       0.85      0.92      0.88      5000
```

Overall Results

Logistic Regression: test accuracy-0.9354

Random Forest: test accuracy-0.9362

Naive Bayes Classifier: test accuracy - 0.9354

Gradient Boosting: test accuracy-0.9372

Neural Net: Test accuracy - 0.929

Though all models had similar accuracies, the Gradient Boosting is the most accurate when run on the testing data.

Real World Application

When a customer is looking to apply for a credit card, they should give the bank as many of the features as they have, marking the rest as null. Then, their profile would be run through our Gradient Boosting model, predicting either a 0 or 1. A 1 means the customer is likely to default their account in the first 18 months, meaning they do not make payments for three consecutive months. We would be more inclined to deny these customers. A 0 means the customer is not predicted to default their account, so we would be more inclined to accept these customers.

We can assume our model would be 93.72% accurate at predicting if a customer is approved, they will not default their account in the first 18 months.

Conclusion

The most accurate model, Gradient Boosting, only has aa 93.72% accuracy. It is fairly accurate, however there is a chance where a credit card application that deserved a card is rejected. I would recommend, until a better or more accurate model is created, to have in-person jobs that will review the credit card applications for applications that do seem worthy. I would tell my customer, the model stated that your credit card application has been denied, but someone will look into and determine if you will be granted a credit card.