**Predicting Loan Default using Regression**

**Final Report**

**Frankline Ononiwu**

1. Problem Statement
2. The Data
3. Data Wrangling
4. Linear Classification Data Wrangling
5. Modeling
6. Logistic Regression Model
7. Random Forest Classifier Model
8. Support Vector Classifier Model
9. AdaBoost Classifier Model
10. KNearest Neighbors Model
11. Works Cited

**Problem Statement**

The project is a hackathon competition organized by company called Risklab. Risklab is a research company that conducts financial risk management research in partnership with Companies in the industry. Risklab organized a hackathon in 2018 to model credit card default risk using curated banking customer data. In this case study, we are analyzing dataset containing customer attributes and customer historical credit card repayment information. We want to use these variables for analysis and for building predictive models used to classify risk of repayment default.

**The Data**

The dataset consists of 25 variables with 24 independent variables and 1 dependent variable (July\_Repayment Status). It has over 30,000 records and all categorical variables have already been encoded to numerical classes. Dataset has also been anonymized, removing customer identification in order to protect their privacy. To proceed with the modelling after some exploratory analyses, we will be using the sklearn standard scaler to standardize the dataset, and avoiding bias resulting from high data range. All data wrangling, exploration and modeling was performed in the python environment with packages.

**Data Wrangling**

The data was in the csv format and was loaded as a pandas dataframe format on the python environment. The dataset came in as almost completely cleaned except for a few missing values which did not have any noticeable effect on the data statistics after we cleaned them up. We are using five different models for the data, but all classification models we used have the same dataset requirements so there was no attempt to wrangle the data to suit each of the models. The missing values were replaced with the median and mean values. A data.describe() function before and after cleaning confirmed that the missing values did not have noticeable effect on dataset statistics, thereby justifying the use of only mean and median for replacing the missing values.

**Modeling**

We will try five different approaches to classify whether a customer will default on his credit repayment: logistic regression, random forest, support vector machine, knearest neighbor and boosting.

Before we run any of our models, we need to do some pre-processing on the data. The details of these steps can be found in this notebook.

**Logistic Regression Model**

Here, we will use a logistic regression model to classify whether someone will default in loan repayment for the month of July. In this modelling, we will use standard scaling. While this will not have an impact on our results, it will make our regularization more efficient (part of regularization involves calculating norms, so this will be a less costly calculation if we have data with smaller scales). Then, we will test our parameters by using 5-fold cross validation.

Logistic Regression Model with Random Under-sampling - Training Results

Optimized Parameters:

C 0.006

Accuracy Score: 69.6%

Classification Report:

Precision Recall 0 0.93 0.72 1 0.18 0.51 avg/total 0.85 0.70

Confusion Matrix: 7003 2767 556 588

AUC Score: 0.653

We chose the regularization parameter C such that the area under the ROC curve is maximized.

We have a low C value, which means that we are doing a great deal of L2 regularization. That is, we are shrinking the coefficients of many of the features. As we can see from the results, our AUC score is not particularly high, but our precision is relatively high. We don’t get a poor recall score either. Our accuracy is about 70%, which falls in response to the increased recall rate.

We do not have a great model, but let’s try using our model on the test data.

Logistic Regression Model with Random Under-sampling - Testing Results

Accuracy Score: 68.3%

Classification Report:

Precision Recall 0 0.92 0.71 1 0.17 0.50 avg/total 0.84 0.68

Confusion Matrix:

2283 954 200 202

AUC Score:

0.640

We only get slightly worse results on our test data than we did our training data. We may not have a highly accurate model, but we certainly get one that does not overfit.

In addition to the model from scikit learn, we designed an additional logistic regression model for with the data to classify if customers will repay their loan.

**Random Forest Model**

Now, we will implement a random forest model to help solve our classification problem. A big advantage of using a random forest model is that we can view how important our features compared to others. Moreover, we can prevent overfitting by adding more trees to our forest: it is crucial that we do not overfit because we are dealing with a very delicate issue and we cannot afford to have unexpected misclassifications.

Similar to the logistic regression model, we will use 5-fold cross-validation. Like our previous model, we will also try out both random under-sampling and balanced trees. Here are our hyperparameters we are testing out for our random forest model: Hyperparameter Values Tested Number of Trees 100, 250, 500, 1000, 1500, 2000 Max Depth 1, 2, 3, 4, 5, No limit on max Max Features ‘log2’, ‘sqrt’ Criterion ‘gini’, ‘entropy’

As we can see, we are optimizing the number of trees in our forest, the maximum depth of our trees, the maximum number of features we are sampling to split a node into two leaves, and what the criterion is for a split. Here are the results from fitting our model and validating against the training data:

**Boosting**

In the final model, we will try two boosting approaches: AdaBoost and Gradient Boosting. Keep in mind that AdaBoost focuses on the observations with the most error and adds Decision Trees with particular weights according to how well they perform. Learners that have high performance will be given a higher weight. On the other hand, gradient boosting does not focus on specific observations: rather, the Decision Tree trains on the remaining errors of the strong learner.

Since we do not have any features that interact with class weights in this model, we will need to use a random under-sampler to make the data more balanced. That is, we will need to randomly under-sample the majority class without replacement. The hyperparameters we are optimizing are the number of learners and the learning rate. Furthermore, we will used RandomizedSearchCV again to reduce computational time.

Let’s look at the results of our AdaBoost model.

AdaBoost with Random Under-Sampling - Training Results

Optimized Parameters: Number of Trees 1000 Learning Rate 0.1

Accuracy Score: 67.2%

Classification Report: Precision Recall 0 0.93 0.69 1 0.17 0.54 avg/total 0.85 0.67

Confusion Matrix: 6711 3040 530 633

AUC Score: 0.665

We get similar results to those of our logistic regression model: we have higher recall on 0’s and lower recall on 1’s. Like with our random forest, our optimized number of trees is 1000, indicating that we need a fair amount of learners to get our results. We have a learning rate of 0.1, which implies we shrink the contribution of each classifier by 0.1, which is not an insignificant amount. Thus, we need a lot of “information” but the contributions of those pieces of “information” is shrunk. Let’s see our results on the test data:

AdaBoost with Random Under-Sampling - Testing Results

Accuracy Score: 67.6%

Classification Report: Precision Recall 0 0.93 0.69 1 0.17 0.55 avg/total 0.85 0.68

Confusion Matrix: 2250 1006 172 211

AUC Score: 0.646

Our results are nearly identical to those of our training results. Our model is not overfit. The details of this model can be found here.

Let’s look at the results we get from Gradient Boosting with random under-sampling.

Gradient Boosting with Random Under-sampling - Training Results

Optimized Parameters: Number of Trees 500 Learning Rate 1

Accuracy Score: 69.7%

Classification Report: Precision Recall 0 0.93 0.71 1 0.19 0.55 avg/total 0.85 0.70

Confusion Matrix: 6963 2777 529 645

AUC Score: 0.680

Our results are better than those of our AdaBoost training results. We have a higher recall and precision. Accuracy, as well as AUC score, is slightly higher. We require less trees, although our learning rate is slightly higher. So far, we have a more efficient (AdaBoost model took about 5 minutes to train, and the Gradient Boosting model took about 1.5 minutes to train) and higher performing solution than the AdaBoost model. Let’s see the results on the test data.

Gradient Boosting with Random Under-sampling - Testing Results

Accuracy Score: 67.9%

Classification Report: Precision Recall 0 0.92 0.70 1 0.16 0.50 avg/total 0.85 0.68

Confusion Matrix: 2287 980 187 185

AUC Score: 0.621

Unfortunately, our gradient boosting model was slightly overfit. All of our metrics were slightly worse than with our training results. So, we may want to use our AdaBoost model instead.

**Conclusion**

**Works Cited**

*Schwoebel, J. (2018). An Introduction to Voice Computing in Python. Boston; Seattle, Atlanta:*

*NeuroLex Laboratories.* [*https://github.com/jim-schwoebel/voicebook*](https://github.com/jim-schwoebel/voicebook)

*Saxena, Shruti. “Precision vs Recall – Towards Data Science.” Towards Data Science*