STAT3160 FINAL REPORT

Credit Data Classification Machine Learning Model Analysis

Students:
Zihui (Aurora) Weng,
Devin O'Toole,
Michael Straus

Advisor:
Professor Alex Pijyan
ap4347@columbia.edu

May 2, 2024

Contents

1 Abstract

This paper details the comprehensive analysis of the Credit Card Fraud Detection dataset from Kaggle [1]. We address significant aspects of data preprocessing, explore relationships within the data, and describe the application and performance of various machine learning models including Random Forests, Artificial Neural Networks, Gradient Boosting Machines, and an ensemble model.

2 Introduction

The focus of our analysis is the Credit Card Fraud Detection dataset, provided by Kaggle user MISHRA5001.[1] The primary variable of interest, labeled "TARGET", identifies clients experiencing payment difficulties (value = 1) and those without such difficulties (value = 0). Our dataset comprises 122 features, including the TARGET variable, with a significant amount of missing and non-numeric data.

3 Data Preprocessing

3.1 Initial Data Inspection

Our preliminary data review revealed that most features are integers or numeric. We observed that the majority of the 122 features are of type integer or numeric.

Var1	Freq
character	16
integer	41
numeric	65

Figure 1: Table of feature data types.

Of these features, six appeared to have a wider range of values than the others:

- **AMT_INCOME_TOTAL**: Client's income

- **AMT_CREDIT**: Credit amount of the loan
- **AMT_GOODS_PRICE**: Price of goods for which the loan was issued
- **SK_ID_CURR**: Loan ID
- **DAYS_EMPLOYED**: Employment duration before the loan application
- **AMT_ANNUITY**: Annuity of the previous application

This variability necessitated centering and scaling to normalize these predictors.

Variable_Name	Variable_Type	Sample_n	Missing_Count	Per_of_Missing	No_of_distinct_values	min	median	max
AMT_INCOME_TOTAL	numeric	307511	0	0.000	2548	25650.00	147150.00	117000000.00
AMT_CREDIT	numeric	307511	0	0.000	5603	45000.00	513531.00	4050000.00
AMT_GOODS_PRICE	numeric	307233	278	0.001	1002	40500.00	450000.00	4050000.00
SK_ID_CURR	integer	307511	0	0.000	307511	100002.00	278202.00	456255.00
DAYS_EMPLOYED	integer	307511	0	0.000	12574	-17912.00	-1213.00	365243.00
AMT_ANNUITY	numeric	307499	12	0.000	13672	1615.50	24903.00	258025.50
OBS_30_CNT_SOCIAL_CIRCLE	numeric	306490	1021	0.003	33	0.00	0.00	348.00
OBS_60_CNT_SOCIAL_CIRCLE	numeric	306490	1021	0.003	33	0.00	0.00	344.00
AMT_REQ_CREDIT_BUREAU_QRT	numeric	265992	41519	0.135	11	0.00	0.00	261.00
OWN_CAR_AGE	numeric	104582	202929	0.660	62	0.00	9.00	91.00
DEF_30_CNT_SOCIAL_CIRCLE	numeric	306490	1021	0.003	10	0.00	0.00	34.00
AMT_REQ_CREDIT_BUREAU_MON	numeric	265992	41519	0.135	24	0.00	0.00	27.00
AMT_REQ_CREDIT_BUREAU_YEAR	numeric	265992	41519	0.135	25	0.00	1.00	25.00
DEF_60_CNT_SOCIAL_CIRCLE	numeric	306490	1021	0.003	9	0.00	0.00	24.00
HOUR_APPR_PROCESS_START	integer	307511	0	0.000	24	0.00	12.00	23.00
CNT_FAM_MEMBERS	numeric	307509	2	0.000	17	1.00	2.00	20.00
CNT_CHILDREN	integer	307511	0	0.000	15	0.00	0.00	19.00
AMT_REQ_CREDIT_BUREAU_DAY	numeric	265992	41519	0.135	9	0.00	0.00	9.00
AMT_REQ_CREDIT_BUREAU_WEEK	numeric	265992	41519	0.135	9	0.00	0.00	8.00
AMT_REQ_CREDIT_BUREAU_HOUR	numeric	265992	41519	0.135	5	0.00	0.00	4.00
REGION_RATING_CLIENT	integer	307511	0	0.000	3	1.00	2.00	3.00
REGION_RATING_CLIENT_W_CITY	integer	307511	0	0.000	3	1.00	2.00	3.00
TARGET	integer	307511	0	0.000	2	0.00	0.00	1.00

Figure 2: Table of feature summaries, sorted in descending order of maximum value.

3.2 Handling Missing Values

Many features appeared to have a high percentage of missing values (NAs in the data), with 17 features seeing greater than 60% of observations have missing values. We replaced undefined entries ("XNA" and empty strings)

with NA values and eliminated features with over 30,000 missing entries (approximately 10% of the 307,511 total observations in the dataset). For the remaining data, k-nearest neighbors imputation was employed.

Variable_Name	Variable_Type	Sample_n	Missing_Count	Per_of_Missing	No_of_distinct_values	min	median	max
COMMONAREA_AVG	numeric	92646	214865	0.699	3181	0.00	0.02	1.00
COMMONAREA_MODE	numeric	92646	214865	0.699	3128	0.00	0.02	1.00
COMMONAREA_MEDI	numeric	92646	214865	0.699	3202	0.00	0.02	1.00
NONLIVINGAPARTMENTS_AVG	numeric	93997	213514	0.694	386	0.00	0.00	1.00
NONLIVINGAPARTMENTS_MODE	numeric	93997	213514	0.694	167	0.00	0.00	1.00
NONLIVINGAPARTMENTS_MEDI	numeric	93997	213514	0.694	214	0.00	0.00	1.00
LIVINGAPARTMENTS_AVG	numeric	97312	210199	0.684	1868	0.00	0.08	1.00
LIVINGAPARTMENTS_MODE	numeric	97312	210199	0.684	736	0.00	0.08	1.00
LIVINGAPARTMENTS_MEDI	numeric	97312	210199	0.684	1097	0.00	0.08	1.00
FONDKAPREMONT_MODE	character	97216	210295	0.684	5	*NA*	*NA*	*NA*
FLOORSMIN_AVG	numeric	98869	208642	0.678	305	0.00	0.21	1.00
FLOORSMIN_MODE	numeric	98869	208642	0.678	25	0.00	0.21	1.00
FLOORSMIN_MEDI	numeric	98869	208642	0.678	47	0.00	0.21	1.00
YEARS_BUILD_AVG	numeric	103023	204488	0.665	149	0.00	0.76	1.00
YEARS_BUILD_MODE	numeric	103023	204488	0.665	154	0.00	0.76	1.00
YEARS_BUILD_MEDI	numeric	103023	204488	0.665	151	0.00	0.76	1.00
OWN_CAR_AGE	numeric	104582	202929	0.660	62	0.00	9.00	91.00
LANDAREA_AVG	numeric	124921	182590	0.594	3527	0.00	0.05	1.00
LANDAREA_MODE	numeric	124921	182590	0.594	3563	0.00	0.05	1.00
LANDAREA_MEDI	numeric	124921	182590	0.594	3560	0.00	0.05	1.00
BASEMENTAREA_AVG	numeric	127568	179943	0.585	3780	0.00	0.08	1.00
BASEMENTAREA_MODE	numeric	127568	179943	0.585	3841	0.00	0.07	1.00
BASEMENTAREA_MEDI	numeric	127568	179943	0.585	3772	0.00	0.08	1.00

Figure 3: Table of feature summaries, sorted in descending order of percentage of missing values.

3.3 Feature Selection and Transformation

We removed 34 features exhibiting near-zero variance and transformed all character features into factors. Categorical features were dummy encoded. Additionally, infrequently occurring values were pooled into a new "other" category in three features (`NAME_TYPE_SUITE`, `NAME_INCOME_TYPE`, `NAME_HOUSING_TYPE`), which affected values appearing in fewer than 1% of observations.

4 Exploratory Data Analysis

Our investigation delved into the associations between various predictors and the target variable to enrich our understanding of the dataset prior to the selection of a predictive model. Figure 4 elucidates the relationship between loan type and loan credit amount, revealing that revolving loans generally involve lower credit amounts compared to cash loans.

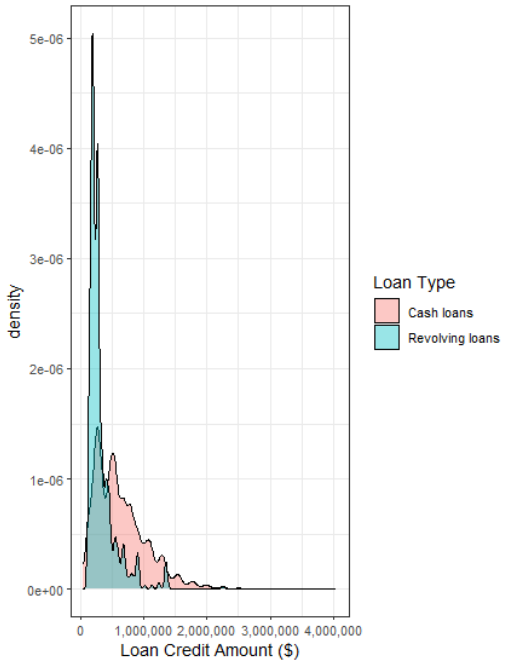

Figure 4: Density plot of loan credit amount in $ by loan type.

In Figure 5, the correlation between payment difficulties and loan credit amount is presented. It is observed that larger credit amounts tend to correspond with fewer payment difficulties, although this relationship is somewhat obscured by the scarcity of instances where the target value is equal to 1.

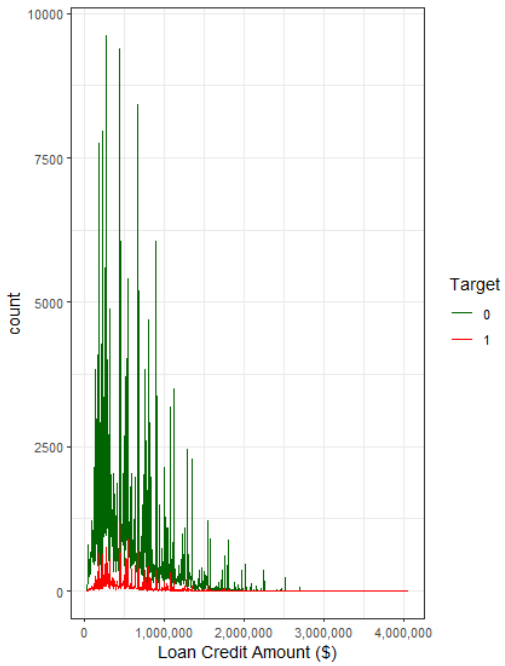

Figure 5: Loan credit amount frequency vs. target.

Figure 6 examines the distribution of client income by gender, highlighting that while the income distribution at the lower spectrum is fairly uniform between genders, males more frequently earn between approximately $10,000,000 and $30,000,000. Notably, the highest income recorded was from a female client.

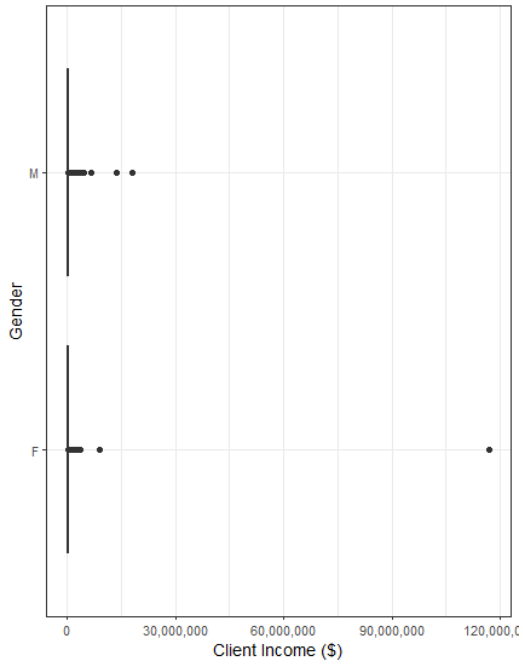

Figure 6: Client income distribution by gender.

Figure 7 explores the impact of client education level on the likelihood of payment difficulties. It is evident that clients with only lower secondary education are more prone to encounter payment difficulties. Furthermore,

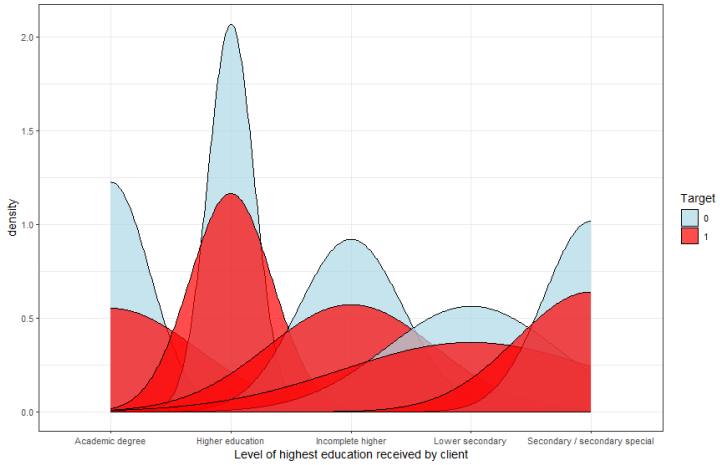

Figure 7: Density plot of client education level and target.

figure 8 investigates the relationship between loan amount and client region rating, illustrating that higher region ratings often correlate with lower loan amounts compared to regions with lower ratings.

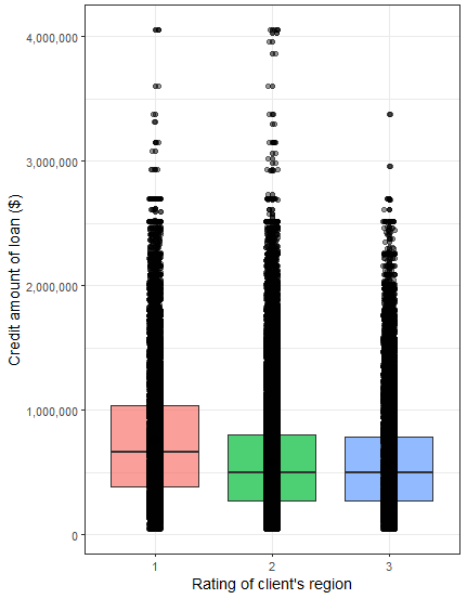

Figure 8: Boxplot of client region rating and loan credit amount.

Lastly, Figure 9 confirms that regions with higher ratings tend to experience, on average, more payment difficulties.

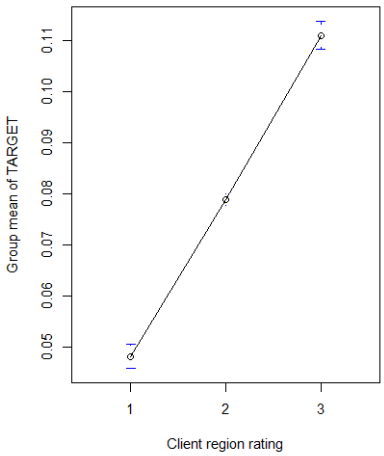

Figure 9: Group means of target by client region rating.

These insights guided our subsequent model selection.

5 Model Selection Rationale

5.1 Justification Per Algorithm

5.1.1 Support Vector Machines

For our selection of models, we wanted to make choices that could be scaled to our dataset with many features and observations. We learned that, after some cursory analysis with a subset of the data, that SVM was not up to the task. That algorithm took about 30 times longer to run than equivalent-accuracy Random Forest and Artificial Neural Network processes, and we wanted to produce results which could be run repeatedly to get the best hyperparameters possible without fear of taking up too much time. As such, we focused our efforts into Artificial Neural Networks, Random Forests, Gradient Boosting Machines, and an ensemble which combined all three.

5.1.2 Random Forests

Looking at our data, we noticed that we have many more samples (300k) than in any previous project. Thus, creating robust models that are easily scalable to greater numbers of instances is key. This is the perfect job for random forests, as they aggregate the results from many different decision trees, which is itself a hyperparameter that can be fitted as needed. Even if training time is too great, we can always strip down the complexity of the model with easily understandable hyperparameters. We can also ignore the issue of normalizing our data, as random forests are distribution free, and may be especially useful in our case because many of our categorical variables have non-sortable classes. Therefore, we will not have to dummy encode and strip out magnitude to get results. Many columns have missing values, such as wall color, and Random Forest can deal with this situation as it is very robust. We do not need to impute, and instead can squeeze every last bit of information out of the features.

5.1.3 Gradient Boosting Machines

We also chose XGBoost due to its versatility. With the great complexity of our dataset, it may be best to have a wide range of hyperparameters to choose from, such as decay rate, subsample frequency, and number of iterations. Similar to Random Forest, it also handles missing values well, allowing us to minimize data loss during preprocessing. Although XGBoost is often prone to overfitting, the saving grace we have is our 300k instances which will give a lot of variation to minimize bias. However, when we use this approach, we will make sure to remove outliers in the dataset, which do exist in income variables, as then we could have high variance in our resulting model.

5.1.4 Artifical Neural Networks

Artificial neural networks in our case helped find nonlinear patterns in the data which would have been more difficult to find with the above two approaches. Using several layers and a ReLU activation function, we are able to find correlations in our credit card bank that we wouldn't be able to otherwise. For example, how the skewed right feature which records the number of children can relate to credit difficulties, even taking into account confounding factors such as job type which may have complex interdependencies with

it. We also do not know what features may be useful in this dataset, so we will rely partially on ANN's ability to robustly build new ones. Some linear combinations of other variables such as the number of identity documents readily available from all the FLAG_DOCUMENT variables, may be useful, and Artificial Neural Networks will be able to realize these useful association with little coding needed.

5.1.5 Stacked Models

Finally, as a way to increase our predictive power even further and reduce bias, we will design a superlearner with weighted estimates of a tuned linear SVM, Extreme Gradient Boosting tree, and a Random Forest. We decided to use SVM here but not for earlier model evaluation because we believe that even if features are nonlinear and this approach yields low accuracy, it should be outvoted by more accurate models. If we get great results, though, we would investigate SVM further. We decided to go with a superlearner because we found that the weighted estimates of models of the same type were not enough to get the results we wanted - as all instances of the same type made similar errors. We also liked the idea of our algorithm using cross-validation to develop the model, as our best hyperparameter combinations led to high complexity on each individual instance, especially with Random Forest, so we did not want those estimates to be biased by overfitting to one chunk of training data. We also wanted a model which used a variety of different metrics to evaluate the best model, not just ROC, and make predictions accordingly. Fortunately in R we get a wide range of choices of parameters to weigh estimates together, such as Kappa, Accuracy, and Sensitivity. With all these approaches in mind, we started to train our models.

5.2 Approaches to Challenges

5.2.1 Downsampling

Immediately, we were beset by issues. The dataset was imbalanced, with about 90% negative observations and only 10% positive. We assumed this is because people historically have been able to pay off their loans without financial trouble a high percentage of the time. That's great for the world at large, but for our purposes, it presented an issue of monotone predictions. On the first pass, all models produced accuracy of about 90% - but at the

undesirable cost of specificity, which hovered at around 0.1% on Random Forest, XGBoost, and Artificial Neural Network models (Figure 10). Essentially, the models were not able to distinguish between positive and negative instances in either the training or testing datasets, which defeated the whole purpose of performing model building in the first place.

```
Confusion Matrix and Statistics

          Reference
Prediction     n     p
         n 84840  7392
         p    15     7

               Accuracy : 0.9197
                 95% CI : (0.9179, 0.9215)
    No Information Rate : 0.9198
    P-Value [Acc > NIR] : 0.5417

                  Kappa : 0.0014

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.9998232
            Specificity : 0.0009461
         Pos Pred Value : 0.9198543
         Neg Pred Value : 0.3181818
             Prevalence : 0.9197975
         Detection Rate : 0.9196349
   Detection Prevalence : 0.9997615
      Balanced Accuracy : 0.5003847

       'Positive' Class : n
```

Figure 10: Random Forest confusion matrix with tuned hyperparameters on the initial unbalanced test set.

So, we decided to employ downsampling which would ideally create a model with more differential power. We now had a balanced dataset with equal numbers of positive and negative instances. After making this executive decision the models behaved as predicted. Accuracy dropped to around 65%, but specificity and sensitivity were essentially aligned. This finding was more pronounced on our Random Forest algorithm, which now had the highest accuracy of 70% but also severe overfitting issues. In addition, we saw that decreasing our already downsampled dataset to around 10000 from

our available 30000 observations did not decrease accuracy significantly, but ran in a much more reasonable time interval, allowing us to create a tiered system for tuning our parameters. We would first produce a large grid with about 100 combinations on a smaller dataset of 1000 observations. Although we only got about 60% accuracy on the best models with that large of a grid, we found that we could find many local minima in the error function with such an approach. Then, when we scaled up the dataset gradually while keeping positive and negative observations equal, we could be more precise in our selection of hyperparameters to hopefully achieve better results in a reasonable amount of time (Figure 11). This process took hours and not days, and we deemed that any gains in our evaluation metrics from using the whole dataset to tune were simply infeasible to attain given time constraints.

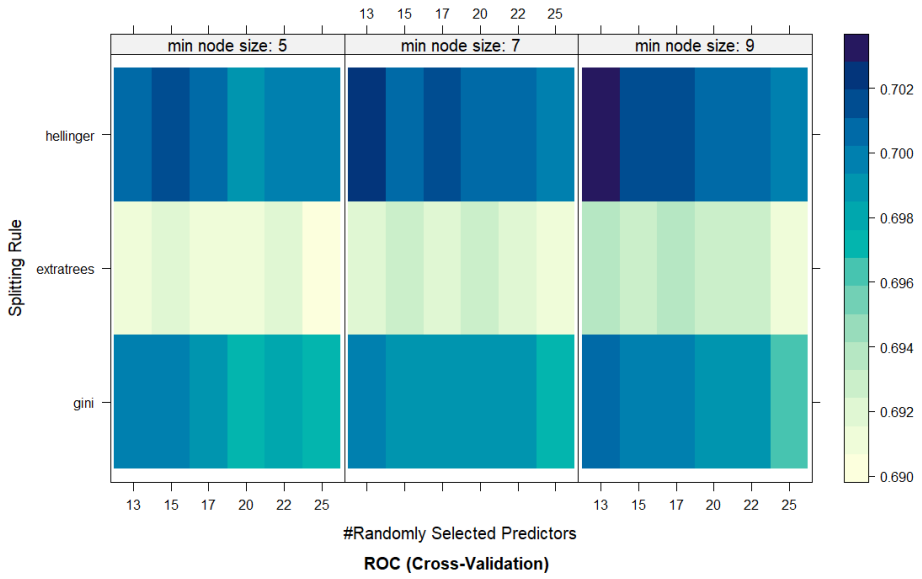

Figure 11: A fine grid used on a 1000-sized subset of our 30000 datapoints, validated on an unseen test set, again for our Random Forest model.

5.2.2 Principal Component Analysis

In looking for ways to improve the model above the 70 percent mark, we first attempted to determine what features were significantly impacting the model. As we did dummy encoding for several variables, we found that

many categorical variables had sparse presences in their respective columns. This led to mainly quantitative features appearing in our permutation based importance analyses (Figure 12).

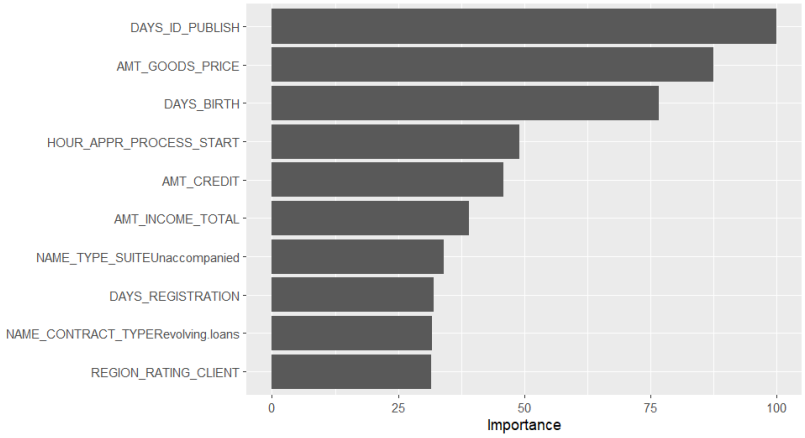

Figure 12: Our feature importance via permutation diagram for our final tuned ANN model on a balanced dataset.

However, in an ideal model, all features passed are relevant and show deeper relations to either a positive or negative response in terms of credit security to the data analyst. If other variables are unimportant, it may be better to downweight them so they do not take up precious bandwidth for the algorithms we are applying to our problem. So, we ran all preprocessing and feature fitting steps again with Principal Component Analysis, and picked out only the first ten features, which we found explained about 90% of variation within the model from our scree plot. We made sure to scale all numeric variables so that they played equal parts in the eigenvalue decomposition process of PCA. In addition, since we found that downsampling made predictions which were not entirely of the negative class, we decided to apply that to our PCA model as well. Unfortunately, our results were underwhelming. Even after extensively tuning all three models on over 50 combinations of hyperparameters each, we were not able to get great gains in accuracy.

In fact, it dropped to 63%, with sensitivity and specificity in that ballpark for all our models (Figure 13 versus Figure 14).

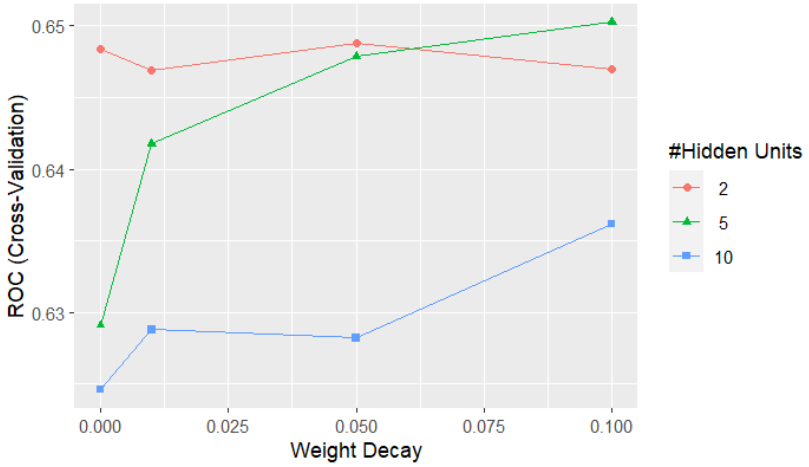

Figure 13: Tuning plots for a balanced dataset with PCA on ANN.

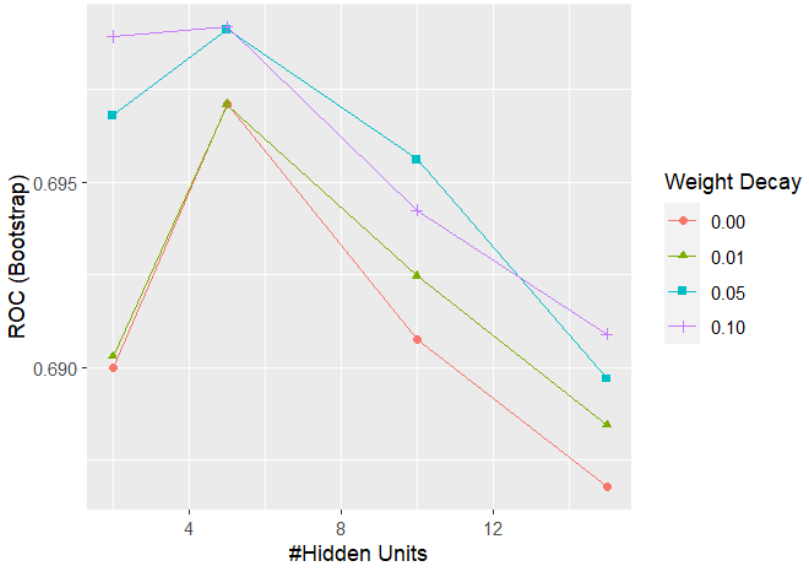

Figure 14: Tuning plots without PCA.

This may be because we were removing too much variance within the model, even with only 10 percent purportedly being dropped. In addition, it may be difficult to extrapolate this approach to new credit holders as we

would need to adjust the correlation matrix each time we got new data, adding complexities to real world deployment.

5.2.3 Threshold Variation

Looking for inspiration to answer why all our approaches were producing suboptimal results, we turned to the Kaggle code section. They are a repository of not only data, but notebooks other people have designed to wrangle it. In particular, we found one which was able to get 94% accuracy and 50% specificity on those elusive positive cases [2]. They used models similar to us in Python, such as XGBoost and Random Forest, that were likely implemented in similar ways between the two languages. Two noticeable preprocessing steps which varied between ours and their approaches was that they used upsampling to balance out positive and negative classes, and also they toggled the threshold for which a class would be considered positive. The default value is .5, as for most models the predictive power for positive and negative classes is the same. However, because it is easy for the Random Forest model to overfit to the upsampled data, which has artificial variation that may not produce as many relationships between variables, the notebook creators got better results after adjusting the bar downward. They tested many values on a held-out validation set, and landed on the surprisingly low number of 0.15. The fact that even their professional, published model had such difficulty recognizing positive classes is a testament to the unruliness of this dataset. In order to see what a reasonable value would be for ours, we produced the probability of a correct prediction for both our positive and negative instances on a tuned Random Forest model with upsampling (Figure 15-16). We note the probability value is much higher for predicting absences versus presences, even with upsampling which balanced instances of each class during training. We ended up using a slightly lower threshold (0.14), after which one saw an accuracy of 80% and a specificity of around 35% on an unbalanced test set.

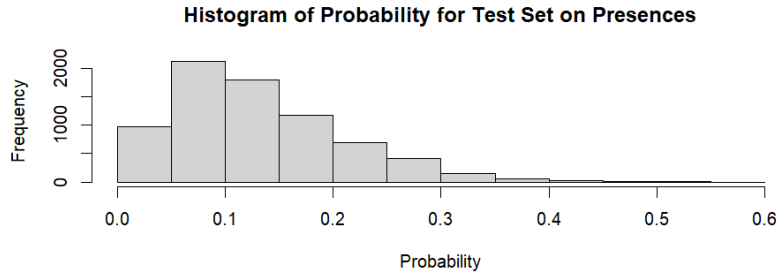

Figure 15: Histograms of correct values for positive prediction.

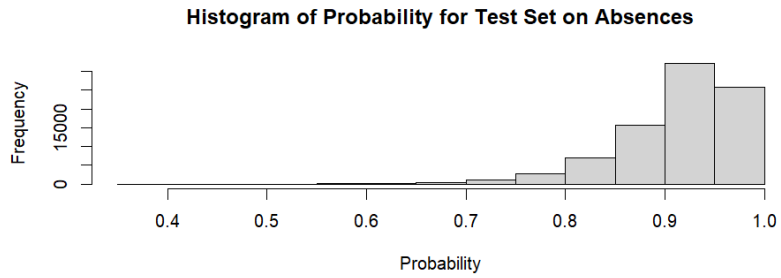

Figure 16: Histograms of correct values for nagative prediction.

This was our best predictive model yet in terms of predicting positive occurrences without having to make the test set 50-50. However, it does not solve all our problems, as we do pay in accuracy when we decrease this threshold as negative instances get classified incorrectly. This approach also took a long time to run, as it almost doubled the size of our dataset. As such, we were not able to get a stacked model to run on this data, as we spent 2-3 days running it with no results.

5.2.4 Other Methods

The aforementioned processes are not the only ones we used in a vain attempt to extract more predictive power from our dataset. As detailed in the divided_model_code folder in our Github repository, we also tried not scaling our predictors, not dropping but instead just using features with high proportions of missing values, aggregating the flagged document columns, and

attempting various kernels with SVM. No approaches stood out in terms of giving us better results, though - assumedly because none actually yield much more correlations within the data, and instead shuffle around information.

5.3 Variable Feature Importance

As for feature importance, we found similar trends across models, indicating some variables are more relevant than others. Looking at Figure 17-19, we find that AMT_GOODS_PRICE and AMT_CREDIT, which both relate to the size of the credit card debt with a customer, makes it onto the top 10 list of predictor variables each time. This makes sense, as larger credit card payments are harder to meet. I would expect banks to adjust their risk accordingly and only give those loans to more financially stable customers. However, our results indicate that is not always the case. Perhaps less surprising is the AMT_ANNUITY variable showing up in all three models. We assume that there may not be a causal relationship, as annuities usually do not give you any more money than plain old investments. However, people who have annuities are likely more forward thinking, as they are willing to put off short term happiness for long term rewards, which is similar to the type of person that would pay off their loans on time. Taking this into account, we removed other variables besides these top 10, and produced models. However, similar to the PCA findings, removing data decreased accuracy to 60%, with sensitivity, ROC, and specificity aligned there as well. It is still good to know what gives our model its results, even if we cannot directly apply this information to produce better models.

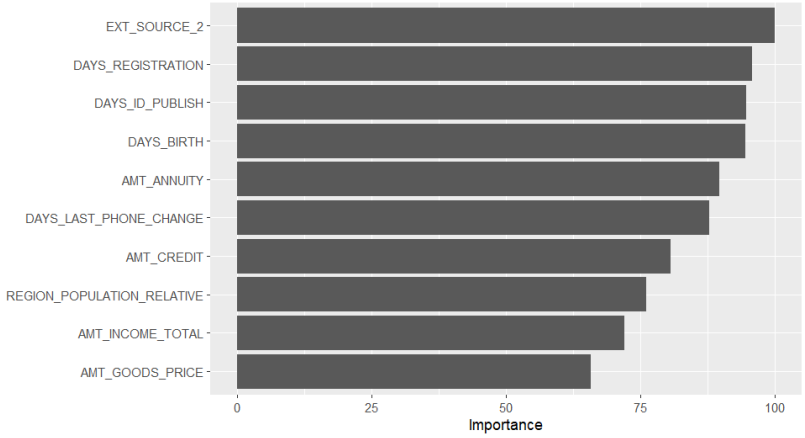

Figure 17: Variable Importance of feature (VIP) scores, obtained by permuting columns of data from Random Forest trained on the unbalanced dataset

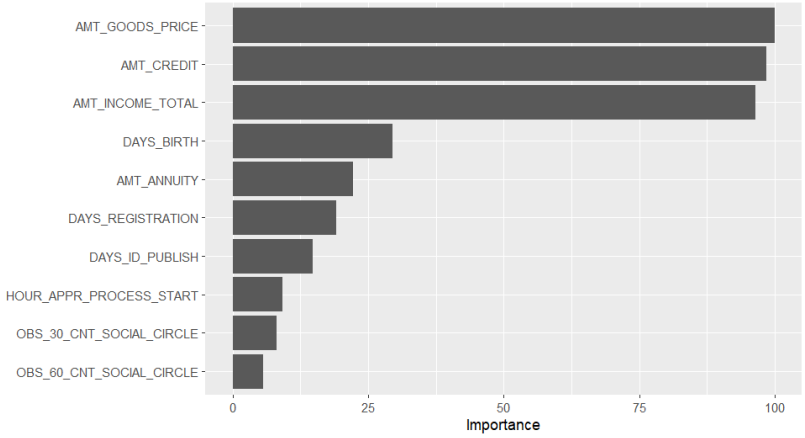

Figure 18: VIP for ANN

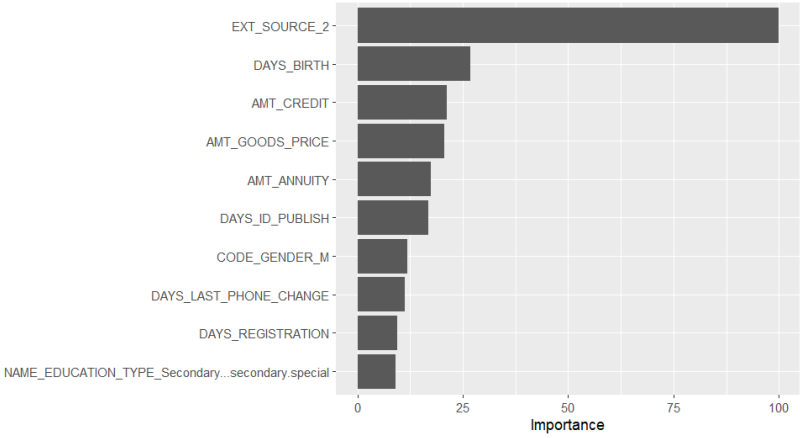

Figure 19: VIP for XGBoost

6 Conclusion

We applied all the methods we learned in class and more to our problem of credit card loan defaulting. Although our sensitivity was low on our approaches, we were confident that we gave this problem our best shot. As mentioned before, even advanced models show metrics with values around the 50% range [2]. We used a variety of approaches to resolve the issue, even trying to replicate the results of those who came before us and build upon them with methods such as downsampling and threshold modification. And while preparing all necessary materials for our model, we built a functioning website and got a chance to write our own code in R that used functions applicable to a wide variety of situations - from PCA, to balancing datasets, to producing informative visualizations. In that regard, we consider our project a success.

References

[1] Mishra5001, "Credit card fraud detection," Jul 2019. [Online]. Available: https://www.kaggle.com/datasets/mishra5001/credit-card

[2] thedat1001020, "Xgboost, random forest," Dec 2023. [Online]. Available: https://www.kaggle.com/code/thedat1001020/xgboost-random-forest