PREDICTING CREDIT CARD DEFAULT

A PREPRINT

Group L
Ben Brahim, Abdelhamid
Carretero Alvarez, Pablo
De Mariscal Perez, Patrik
Gu, Chen
Li, Fred

January 7, 2020

1 Introduction

The aim of this report is to explain in detail the code developed to perform machine learning algorithms for prediction of Credit Card Default. There will be a description of the first steps taken to analyze and understand the provided 'Credit Card' Data, as well as every manipulation and transformation applied before undertaking a series of model approaches. Descriptions on these models, as well as their pros and cons, will be discussed. A section regarding the rationale behind the validation techniques performed will follow, continued by a summary of the results obtained, their limitations and possible improvement techniques. Finally, the best performing machine learning models will be used on the test data set, which provided successful 'recall' scores of around 35-40%. This is one of the oldest and most useful machine learning finance applications, as significant risk can be identified early on and reduced by banks, governments and any other entity subject to capital loaning.

2 Data Transformation and Exploration

Prior to training, we applied several transformations to the data and explored it in multiple ways. Initially, we explored the data with simple functions such as head(), info(), and describe(), through which we obtained a general description of the data. We also checked for null values and found there to be none. Thus, we found data cleaning to be unnecessary.

We used the value_counts() function to spot unusual values present in the following input features: LIMIT_BAL, SEX, EDUCATION, MARRIAGE, AGE, and PAY_1 through to PAY_6. We found unusual unique values in EDUCATION (0, 5, 6), MARRIAGE (0), and PAY_1 through to PAY_6 features (-2, 0). For EDUCATION, we substituted these values for a 4 (indicating "Others" category). For MARRIAGE, we substituted the '0' value for 3 (also indicating "Others" category). We found that a high proportion of the values of PAY_1 feature were '-2' or '0', implying that they are unlikely to be anomalous data. After further investigation, we found that the unique values '-2' and '0' from PAY_ features, although not mentioned in the coursework's brief, have an actual meaning: no consumption and revolving credit, respectively. Thus, we decided to keep these unique values in our dataset.

We also obtained the split of Default vs. Non-Default in our dataset (77.62% vs. 22.38%), which showed us that we were working with a reasonably balanced data set, especially considering that for most default-prediction cases, datasets are heavily imbalanced.

We performed several techniques of exploration, including visualization and feature selection. We had a preliminary look at the distribution of each feature of the data set via the plotting of histograms. We confirmed that there were no remaining anomalous unique values in any of the features and that none of the features had been capped. We found that the distribution of Age was concentrated between 20-50. Furthermore, we found that all BILL_AMT had similar distributions when compared to one another, as was the case for PAY_2 through to PAY_6 and PAY_AMT features. It makes sense that the former and latter attributes present similar distributions as credit card holders need to pay the amount of the bill. It was found, however, that the scale of these features was different, suggesting that feature scaling was required.

We also looked at the distribution and relationship between different continuous variables with respect to Defaults. Looking at PAY_AMT1 vs. Age, we were not able to identify any meaningful relationship between these and Defaults.

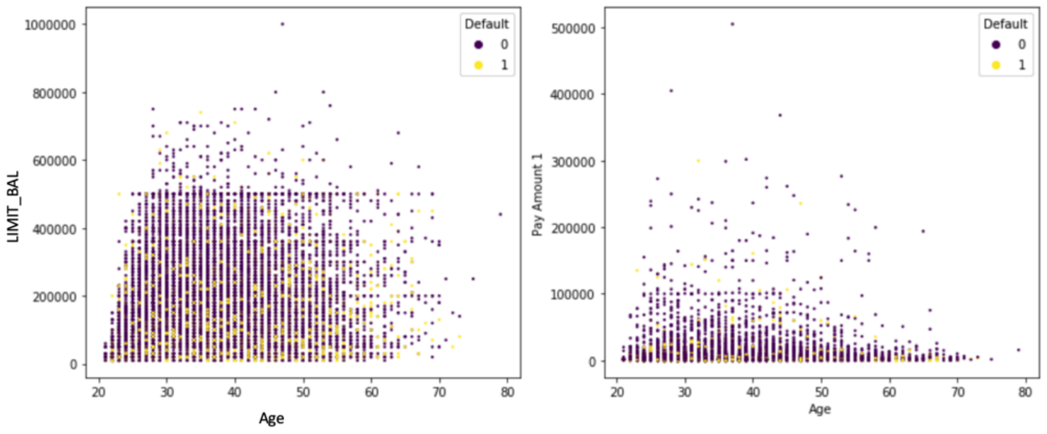


Figure 1: LIMIT_BAL vs Age and Pay Amount 1 vs Age

We also found this to be the case with PAY_AMT3 vs. LIMIT_BAL, PAY_AMT3 vs. BILL_AMT3, and LIMIT_BAL vs. Age. From the apparently random spread and distribution of these scatter plots, we determined that 2D data is not ideal for separating and identifying relationships, and thus resorted to using multiple variables and Kernels.

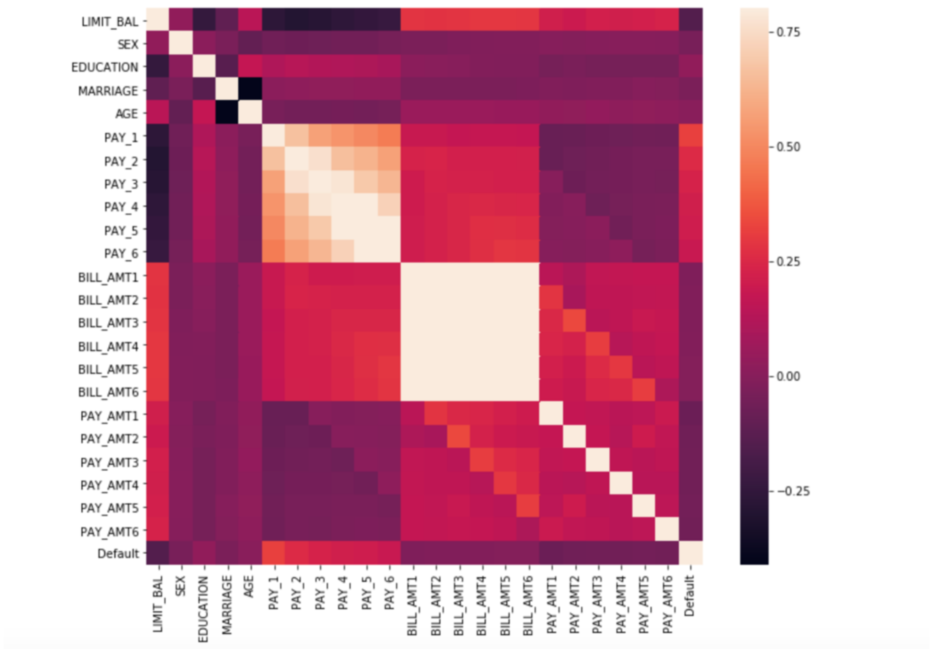


Figure 2: Correlation Matrix

We then computed the correlation matrix to look for linear correlations between features. This is a matrix showing the correlation coefficient for each pair of features, with values between –1 and 1, where –1 indicates a strong negative correlation and 1 indicates a strong positive correlation. A value of 0 indicates no correlation. We found, for example, that LIMIT_BAL had a very weak correlation with Sex. We found there to be a very high correlation between all BILL_AMT variables (>0.80) and a relatively high correlation between all PAY_ variables (>0.40). This is useful information as it indicates that the use of all the latter variables is likely not ideal as it would bring the same information to the models. Thus, in training models, some of these features will be removed.

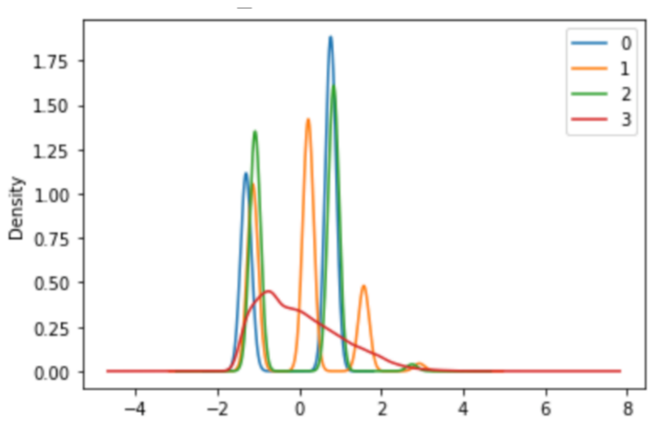


Figure 3: Demographical Features

We also grouped similar types of variables to visualize and analyze them together. We sought to further identify correlations between the variables and compare their distributions. We grouped them into demographic, payment_status, bill_amount, and paid_amount group variables. We did not identify meaningful similarities between demographical features (0:Sex, 1:Education, 2:Marriage, 3:Age), and thus these were treated individually. Payment_status variables were shown to be very similar in distribution, with PAY_1 being the only variable to include a peak uncommon to the other variables. This suggests the importance of maintaining PAY_1 in model training as it includes information absent in other variables. All BILL_AMT variables were shown to have near-identical distributions, and LIMIT_BAL would have a relatively similar distribution to these. All PAY_AMT variables also showed near-identical distributions.

We also separated the Default and Non_Default cases for each feature and plotted the two resultant distributions together. For LIMIT_BAL, we found that the proportion of Non-Default cases increased with increased LIMIT_BAL. This makes sense as clients with a high amount of credit given are likely clients with higher networth which are more unlikely to default. On the other hand, we find that there is a much higher proportion of Default cases for lower LIMIT_BAL, suggesting clients with a lower credit given (i.e. customers likely to have a lower networth) are much more likely to default.

We also looked at the distribution of 3 categories within past payment history (pay duly, 1 month delay, 2 months delay, and 3 months delay) with respect to age. We found that higher delay distributions were slightly shifted towards the lower age area, suggesting longer delays are more associated with younger people.

Additionally, we looked at the distribution of defaults and non-defaults with respect to age. In general, these distributions are similar, with the default distribution being slightly shifted towards the left, indicating a slightly higher proportion of defaults for younger people. This makes sense as younger people will have less income, probably lower job stability and will thus be more prone to delaying their payments. We decided to create bins of the age variable rather than using the given age variable data values as we believed it could have a greater predictive power. Thus, we stratified the age data into 6 different bins, of between 10 and 12 years.

Upon further research, we found that it would be interesting to perform feature combination to see if any combination of variables could unlock further information that could be useful for our models. We decided to combine the Sex and Marriage variables by multiplying these together. Thus we obtained unique values of 1, 2, 3, 4, and 6 for this variable, representing married man, single man or married woman, others man, single woman, and others woman, respectively.

We split the provided train dataset, 25% of which went towards the validation set and 75% of which went towards the training set. Thus, including the 6k data rows from the provided test set, we obtain a 18k, 6k, 6k data rows split between training, validation, and test sets (i.e. a 60% 20% 20% split, which is generally recommended in machine learning).

As mentioned earlier, feature scaling is required given the massive differences between different features. We performed scaling of the data via the StandardScaler() function. This makes each feature normally distributed (Gaussian with mean of 0 and unit variance) and helps remove the effect of outliers. Standardization is required for many machine learning models/classifiers (e.g. RBF kernel of Support vector machines, which assumes all features are centered around 0 and have variance in the same order), which don't perform well if input variables have different scales.

3 Methodology Overview

Before initiating the methodology description, it is key to mention that codes, approaches and models have been inspired by a diversified set of sources: DataCamp online courses and various YouTube videos on Supervised Learning have been studied to obtain a first insight on these type of machine learning problems. Furthermore, multiple online sources and machine learning books have been consulted to obtain final versions of the code and report. These are mentioned throughout the report and referenced at the end. The following section will discuss the approach followed to obtain the final algorithms and their corresponding results.

For our analysis, we wanted to identify what features were important in making predictions for our dataset. Feature selection is an important machine learning technique for a few reasons. Firstly, many features within a given dataset do not provide much, if any, signal when it comes to predicting a variable - these features simply add to the noise of the data which leads to overfitting. Of course, the issue then becomes which features are important and which are not - if we remove too many features or too important ones, we then lose some of our signal which leads to underfitting. A second important reason for feature selection is that it makes models less "black box" and more interpretable whilst also reducing the computational price of our models [Kaushik, 2016].

There are many different feature selection and dimensionality reduction techniques in machine learning each with their own pros and cons. Our strategy is to apply multiple different strategies (discussed below) and models on our data set and use the information in combination to identify which of the features are important when it comes to predicting defaults. Then we can identify which variables are consistently of most and least importance using different feature selection methods.

3.1 Analyzing the importance of different variables

We started by first attaining the importance of each variable using an Extra-Trees Classifier. This is a tree-based embedded selection method which gives the importance of each variable in the forming of the model. In general, in classification trees variables that are higher up are more important when it comes to splitting the data. This then tells us which variables are most important when it comes to finding the boundaries.

To obtain these, two steps were taken: firstly, an optimal random state was obtained by creating a loop and assessing the accuracy and recall performances of an Extra-Trees Classifier. Then, the importance of our model features were obtained using the built-in function '.feature_importances_', displaying a major importance of the PAY_1 variable. The mentioned classifier is a random forest technique that implements a meta-estimator that fits a number of randomized decision trees (this is the reason for the name) on a set of sub-samples of the dataset and then averages them in order to improve predictive accuracy and avoid overfitting. Each tree is provided with a random sample k of features, to then use the Gini index to assess the information gain and form each node of the tree.

Our decision of choosing Extra-Trees Classifier instead of a Random-Tree Classifier was based on research: it can be found that the latter performs worse when there is more noise in the data set (it produces a slower run and it is more computationally expensive). Empirical research shows an optimal value of the K parameter to be equal to the square root of the number of variables; however, as it can be seen in our graph, for our particular situation the optimal value was approximatively 19 [The Kernel Trip, 2018].

A disadvantage of this technique is deciding where the cut-off point should be, as there is no mathematical way of choosing an optimal number of features. Additionally, this method only shows you the importance of each variable, not which variables are completely unimportant [Gupta 2019].

3.2 Feature Selection using Recursive Feature Elimination (RFE)

Recursive Feature Elimination is a feature selection technique that works backwards using a model to assess the importance of each variable and removes the least important variable at each iteration.

Recursive Feature Selection was applied on a Random Forrest model, which is run over the whole data set. A score is in terms of how much they add to the model performance based on our scoring method 'recall'. By using cross-validation, it provides substantial ground to eliminate 'poorly' additive features. As it can be observed on the code, features with a value of '1' provide significant recall changes while features with any other values will not be significant towards the overall scoring metric of our model, hence our decision of eliminating them [Weston et al. 2002].

From our analysis, we saw that RFE was consistent with our prior importance analysis, as it kept many of the variables that ranked highly in our importance analysis, such as 'LIMIT_BAL', 'AGE', and 'PAY_1'. This consistency helps support our claims. We also wanted to apply a different embedded feature selection method and thus we used the LASSO.

3.3 Feature selection using Lasso

Lasso regression optimizes a complex squared error loss function, reducing some weight to an insignificant value. It is similar to Ridge Regression but has a major difference: this complex function will use the Ridge regression penalty utilizing the absolute value of the slope instead of the square value, making our predictions less sensitive to the training data set. Without deeply immersing into the mathematics behind it, the main difference between these two approaches is that Ridge regression will shrink the slope of a feature asymptotically close to zero, while Lasso regression can achieve this value. Therefore, the latter will eliminate the weight that ‘useless’ variables have on our model.

When performing feature selection using the LASSO, we again find similar results: only the PAY_6 was shrunk to 0 while many of the variables were reduced very close to 0. If we look at the top 8 features [’MARRIAGE’, ’AGE’, ’PAY_1’, ’BILL_AMT1’, ’BILL_AMT2’, ’BILL_AMT3’, ’PAY_AMT1’, ’PAY_AMT2’] we again see that it is quite consistent with our previous feature selection techniques.

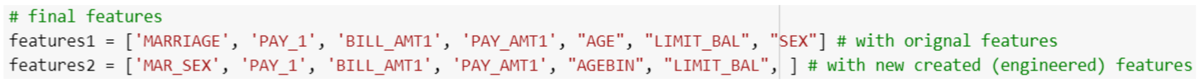
3.4 Feature filtering using variable correlation

Working with highly correlated variables carries out a high risk: the gain in information may be very low and multicollinearity can happen (as some features will be providing redundant knowledge), affecting the performance of our models.

Previously, we visually observed highly correlated variables in our ‘Correlation Matrix’. In this section, features with a correlation coefficient higher than 0.6 are computed and listed. High correlations between Bill and Pay variables can be observed, reinforcing the graphical conclusions previously assumed. Removing these variables could be a potential enhancement for the final execution of our model [Badr, 2019].

3.5 Conclusion

Combining the previously studied feature selection methods, we draw a few important conclusions. From the decision-tree feature importance study, we observe that variables PAY_1, LIMIT_BAL and AGE are key when it comes to implementing these models, and features such as BILL_AMT and PAY_AMT have a lot of predictive power. We also identified correlated variables and decided to remove most of them, as they do not provide much information and are expected to increase the ‘noise’ of our data. The results of the Recursive Feature Selection support this decision and from the Lasso regression, we obtained an insight of which variables do not provide sufficiently useful information for the aim of our study. Therefore, we decided to continue the study with the following two sets of features:



```
# final features
features1 = ['MARRIAGE', 'PAY_1', 'BILL_AMT1', 'PAY_AMT1', "AGE", "LIMIT_BAL", "SEX"] # with orignal features
features2 = ['MAR_SEX', 'PAY_1', 'BILL_AMT1', 'PAY_AMT1', "AGEBIN", "LIMIT_BAL", ] # with new created (engineered) features
```

4 Model Training and Validation

A set of high-performing classification models were studied in this section. The hyperparameters of these models were exhaustively analyzed to find the values which provide an optimal recall score. To do so, the method ‘GridSearchCV’ was used. This method searches over a ‘grid’ consisting of all hyperparameters and then finds the most appropriate ones by cross-validating over the training data set. Cross-validation is a technique which generates many non-overlapping train/test splits on training data and reports the average test set performance. Though this method’s high efficiency is not well understood, it has been chosen due to its excellent empirical results. The scoring method used was ‘recall’, which is the fraction of predicted default cases out of all actual default cases. This decision was made considering credit giver’s main concern, which is to correctly predict default cases to reduce their risk. The following models were evaluated: Logistic Regression, Gaussian Naïve Bayes, Random Forest Classifier, Support Vector Machine and Extreme Gradient Boosting. Each of these are described below.

4.1 Logistic Regression

Logistic Regression is a machine learning technique commonly used for binary classification problems, where the output set contains two classes. Logistic regression can handle both discrete (e.g. gender) and continuous data (e.g.

age). The key function used in logistic regression is the logistic sigmoid, S:

$$S(x) = \frac{1}{(1 + e^{-x})}$$

This function is used so that the model's output is squashed to between 0 and 1 (inclusive), given it is a binary classification problem (0: No Default, 1: Default). If we were to use a linear function for the output, we would obtain an unbounded one. In logistic regression, we model $p_y(y = 1|x)$ (i.e. the probability of 1: Default given an input x) as:

$$\frac{1}{1 + e^{-w \cdot x}}$$

There are several motivations for using logistic regression in machine learning. It leads to an optimisation problem that is smooth and convex, and thus gradient descent can be used. Logistic regression requires that the data has no outliers and that all the values of all features are desired and not anomalous. Thus, we made sure to, for example, remove the undesired unique values from the education and marriage input variables. Additionally, there is the assumption that a linear relationship exists between the input variables and the output, which is not necessarily the case. Overfitting is a common issue with the model, which requires one to remove strongly correlated input variables. Hyperparameter tuning is focused on parameters C and penalty. C is the inverse of regularization strength. The lower the value, the stronger the regularization. It is a parameter aimed at reducing overfitting. The penalty parameter is used to specify the norm to be used in penalization. By introducing the l1 (i.e. Lasso) penalty we obtain higher sparsity (when compared to other penalty norms such as l2) in the solution [Brownlee, 2019].

4.2 Naïve Bayes

Naïve Bayes is commonly used for binary problems, as is the case of the Credit Default problem. It makes use of Bayes' Theorem (shown below) to determine the probability of a hypothesis given prior data that is known. In the problem's context, A represents Default and B represents the data.

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Naïve Bayes assumes the input attributes are conditionally independent. This heavily simplifies the calculation of probabilities and makes the problem tractable. It is an assumption that is unlikely to be fully realistic, but despite this Naïve Bayes models tend to output good prediction results. We chose to model through Gaussian Naïve Bayes as it is commonly used when there are continuous input variables (as is the case in the Credit Default problem). It assumes a Gaussian distribution, which is a strong assumption that does not always hold true. The mean and standard deviation of the input variables are calculated, as well as the probabilities of Default and of Non–Default in the training dataset and the conditional probabilities of each input value given Default and given Non-Default. As no coefficients must be fitted via optimization, training of a Naïve Bayes model is fast.

Naïve Bayes places more restrictions on the set parameters than Logistic Regression - it explicitly defines a dependence between w0 and wi. Naïve Bayes is a generative classifier, and as such, requires more model assumptions than Logistic Regression, which is a discriminative classifier. This makes Naïve Bayes less robust and more sensitive to modelling choices than Logistic Regression. But, due to the higher parameter restriction in Naïve Bayes, if modelling assumptions are appropriate, then Naïve Bayes will require a lower amount of data than Logistic Regression for the same level of convergence in parameter estimates. Assuming independence (which is not necessarily the case), Naïve Bayes tends to perform better than other models [Brownlee, 2019].

4.3 Random Tree Classifier

This classification algorithm consists of the aggregation of a large number (n) of different decision trees which operate as an ensemble. Each individual tree provides a class prediction and the class with a higher number of votes becomes the model prediction (this is often referred as 'bagging'). The addition of all these individual trees gives much more meaningful results than that of any induvial prediction (this is the principal mechanism of ensemble methods). This aggregation creates an uncorrelated forest of trees which cancel out some errors made by individual trees, thus preventing overfitting the data [Gupta, 2019].

4.4 Support Vector Machine (SVM)

Support Vector Machine is a machine learning algorithm frequently used in classification problems. It seeks to find a hyperplane that optimally separates data into two classes. When dealing with big amounts of data, it is often very complicated to separate it in this way. To avoid this, SVM uses kernel methods: this enables the operations in higher dimensional feature spaces, heavily simplifying variable separation. As we had previously mentioned, data will hardly be linearly separated, hence our decision of adopting either Gaussian (RBF) or polynomial kernels to map our SVM algorithm into higher dimensions. Hyperparameter tuning was focused on two parameters: ‘C’ (measure for the complexity of the model) and ‘Gamma’ (spread of the kernel). It is important to mention that very high values of ‘C’ will lead to overfitting, as the model will try to avoid misclassification at all cost. Very high ‘Gamma’ values will cause over-fitting as the model will try to separate data as perfectly as possible, while very low values will provide a high bias. Thus, underfitting will occur due to the ‘linearity’ of the boundary [Albon, 2017].

4.5 Extreme Gradient Boosting (XGB)

Extreme Gradient Boosting Classifier is another ensemble method which intends to convert a set of ‘weak learner’ models into a single much stronger one. The underlying mechanism is the following: it iteratively learns a set of weak models on subsets of the data and then weights each prediction according to the learner’s performance. The result is a single prediction consisting of the combination of all weak learner’s weighted performances. As we are facing a classification problem where we are intending to obtain a category (‘Default’ or ‘No Default’), ‘reg:logistic’ will be the loss function computed on our XGB model. The usage of a tree-based learner provides the same advantages ‘Random Tree Classifiers’ has: an uncorrelated forest with its overfitting prevention.

A loop was created, where all these algorithms were passed by. Here, hyperparameter tuning was performed using a gridsearch. It is important to note that XGB was fitted and transformed the data within a pipeline. Though this could have been applied to all other models, we decided not to do it for the sake of clarity. The optimal hyperparameters obtained were:

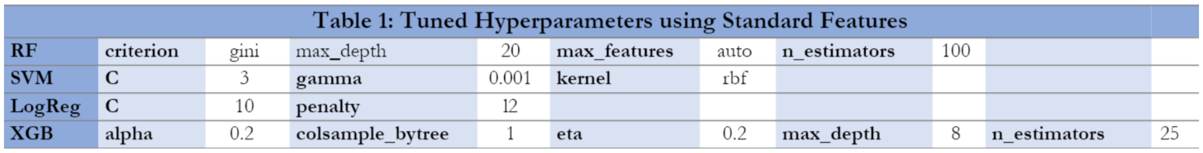


Table 1: Tuned Hyperparameters using Standard Features

RF	criterion	gini	max_depth	20	max_features	auto	n_estimators	100		
SVM	C	3	gamma	0.001	kernel	rbf				
LogReg	C	10	penalty	l2						
XGB	alpha	0.2	colsample_bytree	1	eta	0.2	max_depth	8	n_estimators	25

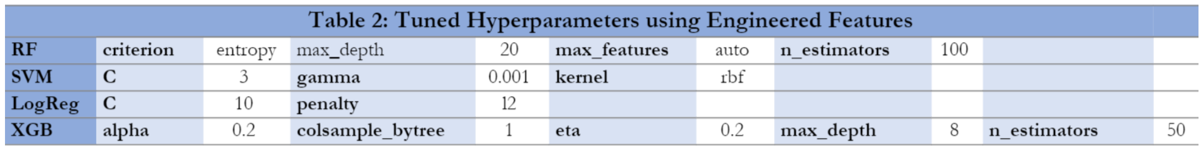


Table 2: Tuned Hyperparameters using Engineered Features

RF	criterion	entropy	max_depth	20	max_features	auto	n_estimators	100		
SVM	C	3	gamma	0.001	kernel	rbf				
LogReg	C	10	penalty	l2						
XGB	alpha	0.2	colsample_bytree	1	eta	0.2	max_depth	8	n_estimators	50

4.6 Ensemble Model (Stacking)

We chose to use a stacking ensemble method to combine our different model’s predictive power and try to form a better model based on recall score. We chose to use stacking, which is a powerful ensemble technique that combines multiple classification models using a meta-classifier. A meta-classifier works on the outputs of the base classifiers. For example, in our model the base algorithms are the LR, SVM and GNB and the chosen meta-classifier is the RF. The base classifies are first run on our training data and then the meta-classifier is run on the output of the base algorithms.

The stacking algorithm works by the following way. We have our training data D and we have T base classifiers, and for each one we train them on D. Then we create a new data set for all the predictions 1 to ‘m’ for each of our base classifiers. Then we train our meta classifier on this new data set to obtain our ensemble classifier. Intuitively, what stacking does is it helps blend the decisions boundary: simple logistic regression would have a linear boundary, and other classifiers would have an exponential one, but stacking brings these together. Using uncorrelated algorithms allows us to reduce the variance. Each model will be affected by some degree of variance, and thus will likely be fitting some amount of noise. However, when you combine the different uncorrelated algorithms you eliminate some of the individual variances of different algorithms leading to an algorithm that better predicts the data.

We tried all our algorithms as meta-classifiers and found that the best meta-classifier (in terms of recall) for our data and our combination of algorithms was the random forest. The random forest is a commonly used meta classifier as it is an ensemble method [Vadim, 2019].

5 Results

After we used cross-validation to tune our hyperparameters, we used these tuned models and applied them to our data using cross-validation, with k = 5. Due to the reasons explained earlier, the evaluation metric we chose to apply is Recall.

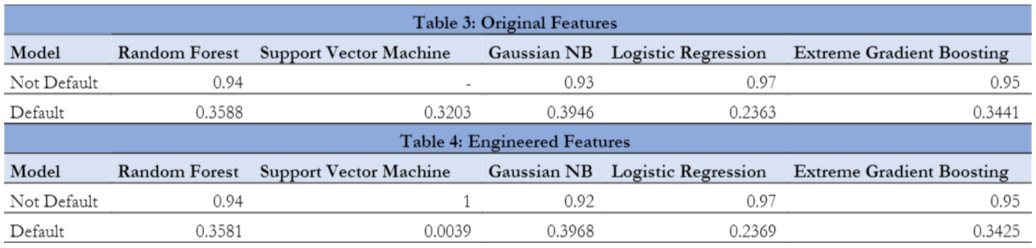


Table 3: Original Features

Model	Random Forest	Support Vector Machine	Gaussian NB	Logistic Regression	Extreme Gradient Boosting
Not Default	0.94	-	0.93	0.97	0.95
Default	0.3588	0.3203	0.3946	0.2363	0.3441

Table 4: Engineered Features

Model	Random Forest	Support Vector Machine	Gaussian NB	Logistic Regression	Extreme Gradient Boosting
Not Default	0.94	1	0.92	0.97	0.95
Default	0.3581	0.0039	0.3968	0.2369	0.3425

We first applied each tuned model to the original feature set and then did the same with engineered features to compare both. There were minimal differences between the two feature sets for most of the models. When we compare the different models, we notice that the best performer in terms of recalling defaults was the Gaussian NB. A potential reason for this could be that the features chosen are conditionally independent (as we have removed correlated features, this is more likely to be the case). When features are less dependent the Naive Bayes outperforms other models. It is important to observe that SVM gave surprisingly inaccurate results with the tuned hyperparameter C=3 (see Table 4). This model's performance was studied individually with C=50, obtaining the recall values shown in Table 3. The recall value for SVM-Not Default could not be obtained as 'C' is a very computationally expensive value and the code took an excessive amount of time to run. Thus, 'C' was not tuned on a GridSearchCV.

5.1 Precision/Recall Graph

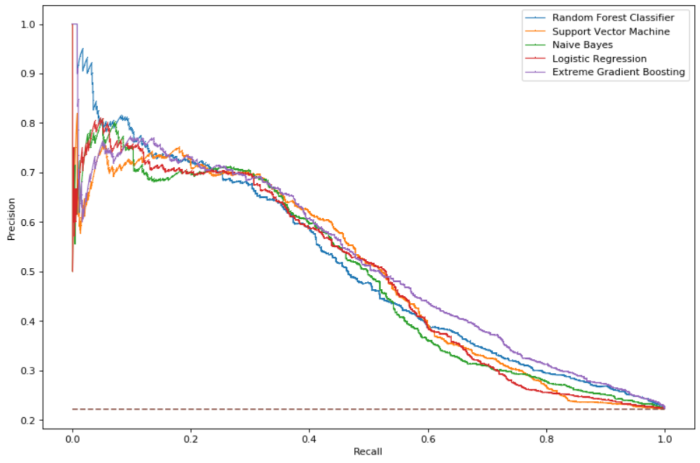


Figure 4: Precision/Recall Graph

$$Precission = \frac{TP}{TP + FP} \quad Recall = \frac{TP}{TP + FN}$$

The precision-recall graph tells us the trade-off between the precision and recall of our model. The recall is the performance measure of the whole positive subset of the data, whereas precision is the performance based on the positive predictions. That means recall is the proportion of correctly predicted positives out of the total number of actual positives, whereas precision is the proportion of correctly predicted positives out of the total number of predicted positives. There is a trade-off between this in general as we can see from the graph above. Higher recall means a lower precision and vice-versa [Opex Analytics, 2019]. If one considers the extreme where all data points are predicted to be positive, this results in a recall of 100% as FN = 0 so TP / (TP + FN) becomes TP/TP = 1, but it also means that precision will be lower, as false positives will be higher since one is assuming every datapoint is a positive.

From the graph above we see that XBG seems to outperform other models in terms of the PRC. This is because it is the model that is the furthest to the top right meaning that the trade-off between precision and recall is smaller. We also see that the GNB and the LR are some of the worst performers when it comes to this trade-off as they generally have the lowest curves. We also see that for the XGB we can have a recall of 60% whilst maintaining a respectable precision of 40%, demonstrating its dominance in this respect.

5.2 Receiver Operator Characteristic (ROC) Curve

$$TPR = \frac{TP}{TP + FN} \quad FPR = \frac{FP}{TN + FP}$$

The Receiver Operator Characteristic (ROC) compares the relationship between the true positive rate and the false positive rate. It is created by plotting the proportion of observations that were correctly predicted positive (1) out of all the positive observations, against the proportion of observations that are falsely predicted positive out of the total number of negative observations (0). The yellow-dashed line shows the baseline: this is the point where True Positive Rate (TPR) = False Positive Rate (FPR). The further up and to the left a classifier's curve is the better the classifier is [Narkhede, 2019]. For example, a classifier tending towards the point (0,1) would have a TPR of almost 1, meaning that when it predicts 1 it is almost always right, and thus it would have a very low FPR as it rarely predicts 1 incorrectly. This is important in the context of our problem, as from the banks perspective they don't want to incorrectly predict that a customer will default as this would mean a loss of potential revenue and even of the customer itself.

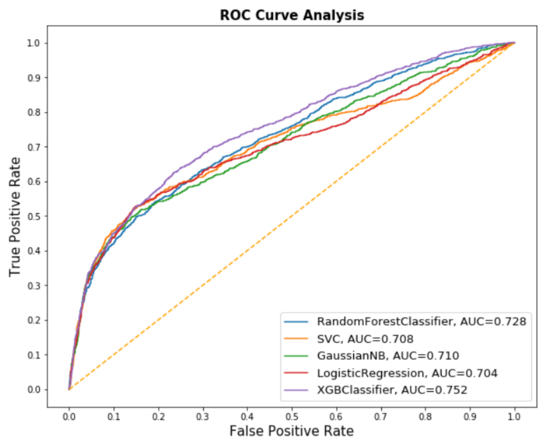


Figure 5: ROC Curve Analysis

From the ROC graph, we see that the XBG is the best performer with an Area Under the Curve (AUC) of 0.75, followed closely by the RF. However, the algorithms seem to perform quite similarly when comparing their TPR to their FPR. As the XGB performs best in this area, this implies it is the best model out of our set of chosen models when it comes to distinguishing between the distributions of true positives and true negatives.

6 Final Predictions on Test Set

The best performing models (Gaussian Naïve Bayes and Extreme Gradient Boosting) were chosen to perform our final predictions on the provided test data set. As it can be observed in Table 5, the final models provide satisfactory results.

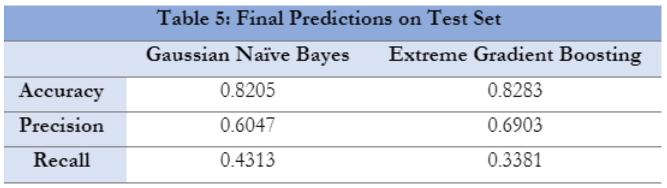


Table 5: Final Predictions on Test Set

	Gaussian Naïve Bayes	Extreme Gradient Boosting
Accuracy	0.8205	0.8283
Precision	0.6047	0.6903
Recall	0.4313	0.3381

Gaussian Naïve Bayes obtains a recall value of 43% (improving the performance obtained when testing the validation set) and Extreme Gradient Boosting achieves a 34% recall score. None of these values is significantly different than those obtained on the training and validation section, showing that these models do not present under- nor overfitting characteristics.

7 Conclusion

As described in section 6, our final results show good prediction results in terms of accuracy, precision, and recall. Nevertheless, these values have room for improvement. A potential source of improvement could be the investigation of further models and comparison of these, as well as further hyperparameter tuning. However, the greatest improvement could potentially come from the use of deep learning (a subset of machine learning) to solve the Credit Card Default problem. In deep learning, Artificial Neural Networks (ANN's) learn how to perform a task from labeled data without the need to program explicit rules of how to do so. ANN's have exhibited great empirical success and are a continuously growing area of research within machine learning, having been successfully applied in image and speech recognition as well as patient diagnosis [Chauhan, 2019].

References

[1] Albon, Chris. "SVC Parameters When Using RBF Kernel." Chris Albon, 20 Dec. 2017, chrisalbon.com/machine_learning/support_vector_machines/svc_parameters_using_rbf_kernel/.

[2] AlindGuptaCheck. "ML: Extra Tree Classifier for Feature Selection." GeeksforGeeks, 26 July 2019, www.geeksforgeeks.org/ml-extra-tree-classifier-for-feature-selection/.

[3] Analytics, Opex. "Precision and Recall: Understanding the Trade-Off." Medium, The Opex Analytics Blog, 24 July 2019, medium.com/opex-analytics/why-you-need-to-understand-the-trade-off-between-precision-and-recall-525a33919942.

[4] Badr, Will. "Why Feature Correlation Matters .... A Lot!" Medium, Towards Data Science, 22 Jan. 2019, towardsdatascience.com/why-feature-correlation-matters-a-lot-847e8ba439c4.

[5] Brownlee, Jason. "Logistic Regression for Machine Learning." Machine Learning Mastery, 12 Aug. 2019, machinelearningmastery.com/logistic-regression-for-machine-learning/.

[6] Brownlee, Jason. "Naive Bayes for Machine Learning." Machine Learning Mastery, 12 Aug. 2019, machinelearningmastery.com/naive-bayes-for-machine-learning/.

[7] Chauhan, Nagesh Singh. "Introduction to Artificial Neural Networks(ANN)." Medium, Towards Data Science, 10 Oct. 2019, towardsdatascience.com/introduction-to-artificial-neural-networks-ann-1aea15775ef9.

[8] Geron Aurelien. Hands-on Machine Learning with Scikit-Learn and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, 2017.

[9] Geurts, Pierre, et al. "Extremely Randomized Trees." Machine Learning, vol. 63, no. 1, 2006, pp. 3–42., doi:10.1007/s10994-006-6226-1.

[10] "L1 Penalty and Sparsity in Logistic Regression." Scikit, scikit-learn.org/stable/auto_examples/linear_model/plot_logistic_l1_l2_sparsity.html.

[11] Malik, Usman. "Applying Filter Methods in Python for Feature Selection." Stack Abuse, Stack Abuse, 31 Oct. 2018, stackabuse.com/applying-filter-methods-in-python-for-feature-selection/.

[12] Narkhede, Sarang. "Understanding AUC - ROC Curve." Medium, Towards Data Science, 26 May 2019, towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

[13] "Random Forest vs Extra Trees." The Kernel Trip, 2 Sept. 2018, www.thekerneltrip.com/statistics/random-forest-vs-extra-tree/.

[14] Shaikh, Raheel. "Feature Selection Techniques in Machine Learning with Python." Medium, Towards Data Science, 28 Oct. 2018, towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e.

[15] "Sklearn.linear_model.LogisticRegression." Scikit, scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[16] Smolyakov, Vadim. "Ensemble Learning to Improve Machine Learning Results." Medium, Stats and Bots, 7 Mar. 2019, blog.statsbot.co/ensemble-learning-d1dcd548e936.

[17] "Support Vector Machines: A Simple Explanation." KDnuggets, www.kdnuggets.com/2016/07/support-vector-machines-simple-explanation.html.

[18] Wolpert, David H. "Stacked Generalization." Neural Networks, Pergamon, 18 Oct. 2005, www.sciencedirect.com/science/article/pii/S0893608005800231.

[19] Yiu, Tony. "Understanding Random Forest." Medium, Towards Data Science, 14 Aug. 2019, towardsdatascience.com/understanding-random-forest-58381e0602d2.

[20] YouTube, YouTube, www.youtube.com/watch?v=m2a2K4lprQw.