BT2103: Optimization Methods in Business Analytics

AY 2022/23 Semester 1

Group Project

Prepared by:

Name	Matriculation Number
Calvin Septyanto	A0244491A
Koh Zhe Wei	A0234382H
Lim Yi Han	A0239065B
Varsha Ramkumar	A0237298N

Prepared for: Dr. Rudy Setiono

TABLE OF CONTENTS

1. Introduction of data set and data modelling

1.1 Background

The banking and credit card industry faces the biggest risk of credit risk, which is usually an area that requires the most amount of capital. As a result, detection and management of default risk is the key factor to reducing loss and generating revenue in this industry Thus, it is important to note that as compared to the current applications of machine learning and big data that are mainly focused on credit score predicting - which may cause banks to lose out on a valuable client if we when considering risk management, the result of predictive accuracy would be much more useful.

This data set in particular is based on research on a case of customers' default payments in Taiwan, which helps to compare the default probability among 9 data mining methods. Our aim is to conduct a quantitative analysis of the credit card default risk in this data set using various machine learning models without using information such as credit score and credit history and find interpretable data to help the bank with this risk. Information on the attributes of the dataset has been provided in the Appendix.

1.2 Method Overview

- Data: The UCI repository was used to retrieve the dataset "card.csv".
- Models: We applied several classification machine learning models in our analysis - SVM, Logistic Regression, Neural Network, Naive Bayes, Random Forest, Decision Tree, XGBoost, ADABoost, and LightGBM.
- Tools: The programming is done in R. R libraries are utilised for machine learning models as well as data analysis and visualisation.

2. Exploratory Data Analysis

"card.csv" contains data on 30,000 credit card holders' payment information that was taken from a Taiwan bank from April 2005 to September 2005. 23 feature attributes describe each data sample (columns *LIMIT_BAL* to *PAY_AMT6*).

Exploratory Data Analysis is the crucial process of doing preliminary investigations on data in order to uncover patterns, spot anomalies, test hypotheses, and validate assumptions using summary statistics and graphical representations. (Patil, 2018)

2.1 Missing Values and Distribution

No missing values were found in the dataset as shown in Figure 1, so we did not use any missing values handling techniques.

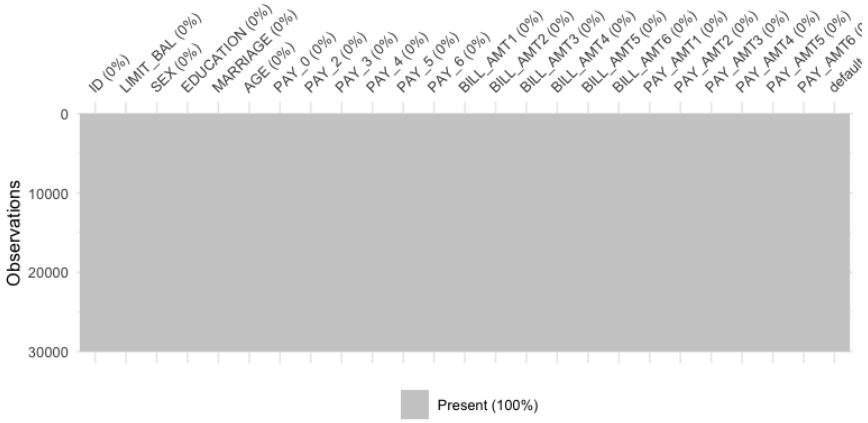


Figure 1: Analysis of Missing Values

"Card.csv" is an imbalanced dataset, with a much lower proportion of credit card holders who defaulted as compared to the proportion who did not default.

Firstly, we examine the overall distribution of defaults and non-defaults. Figure 2 shows that the target class has an uneven distribution of observations as non-defaults comprise 77.88% of the observations whereas defaults only make up 22.12%. This results in an imbalanced dataset.

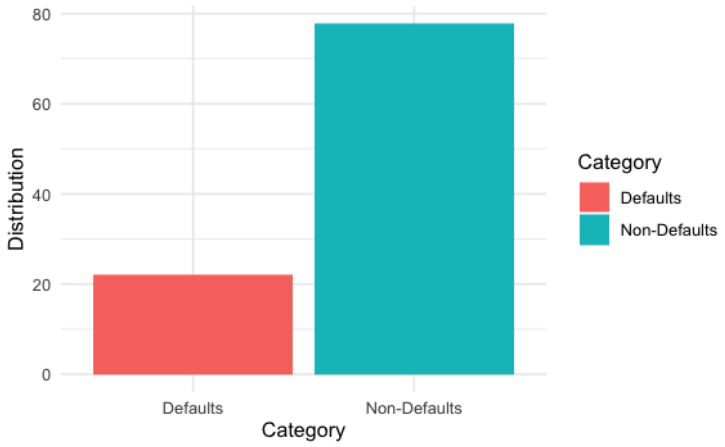


Figure 2: Distribution of defaults and non-defaults in dataset

2.2 Gender Variable

Males tend to default more on their credit card debts than females in this dataset.

Using proportion to gauge each gender's default rate, we find that 24.17% of males defaulted on their debt as compared to 20.78% of females.

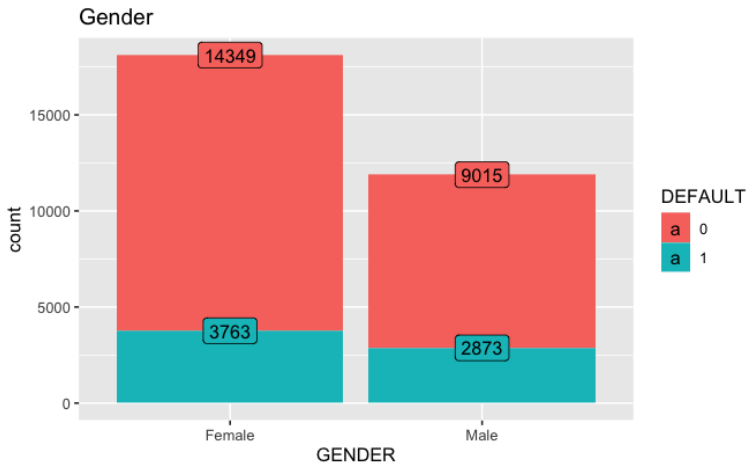


Figure 3: Number of defaults based on gender

2.3 Education Variable

Customers with higher academic qualifications are less likely to default on credit card payments.

Figure 4 indicates that customers with lower levels of education tend to default more, with high school level (3) at 25.16%, followed by university level (2) at 23.73% and graduate school (1) at 19.23%. There is another group (4) named "Others" which is not defined in the dataset with only 7.05% of defaulting customers.

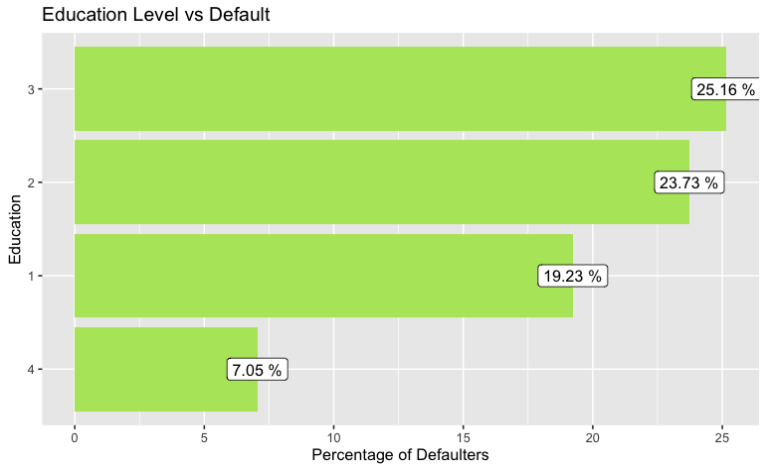


Figure 4: Percentage of defaults based on Education Level

Similarly, customers with higher academic qualifications receive higher credit limits.

Figure 5 indicates that the higher the education level, customers have a higher median, higher upper quartile and higher maximum limits.

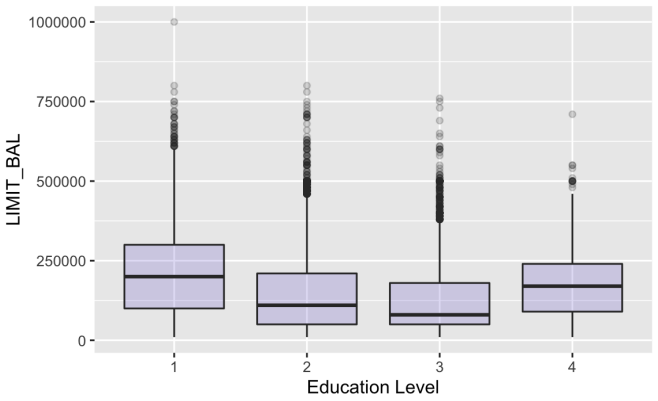


Figure 5: Credit Limits based on Education Level

2.4 Marriage Variable

Customers who are married tend to default more than customers who are not married.

Figures 6 and 7 indicate that a higher proportion of customers who are married (23.61%) tend to default as compared to the proportion of customers who are not married (20.93%).

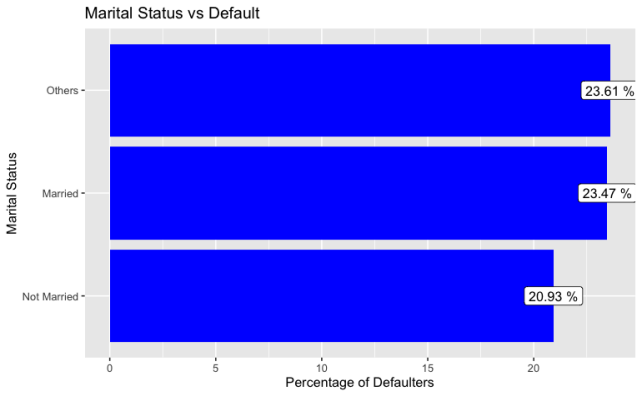


Figure 6: Percentage of defaulters based on Marital Status

2.5 Age Variable

Customers aged 20 to 50 tend to have less defaults than older customers aged 50 to 80.

Figure 8 indicates that the older age groups generally tend to default more than the younger age groups. The group aged 70-80 have the highest proportion

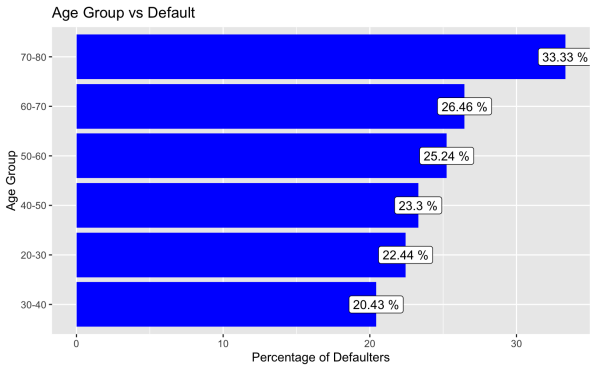
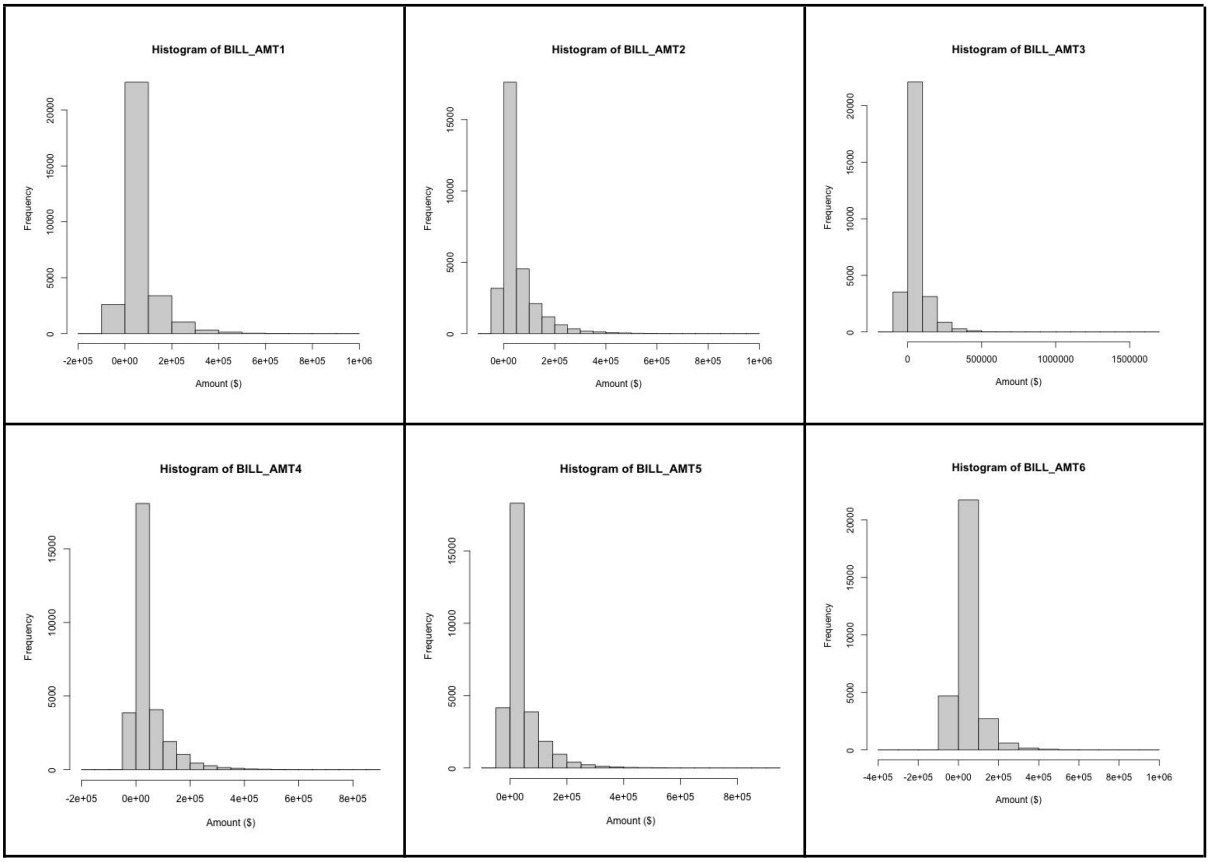


Figure 7: Percentage of Defaults based on Age Groups

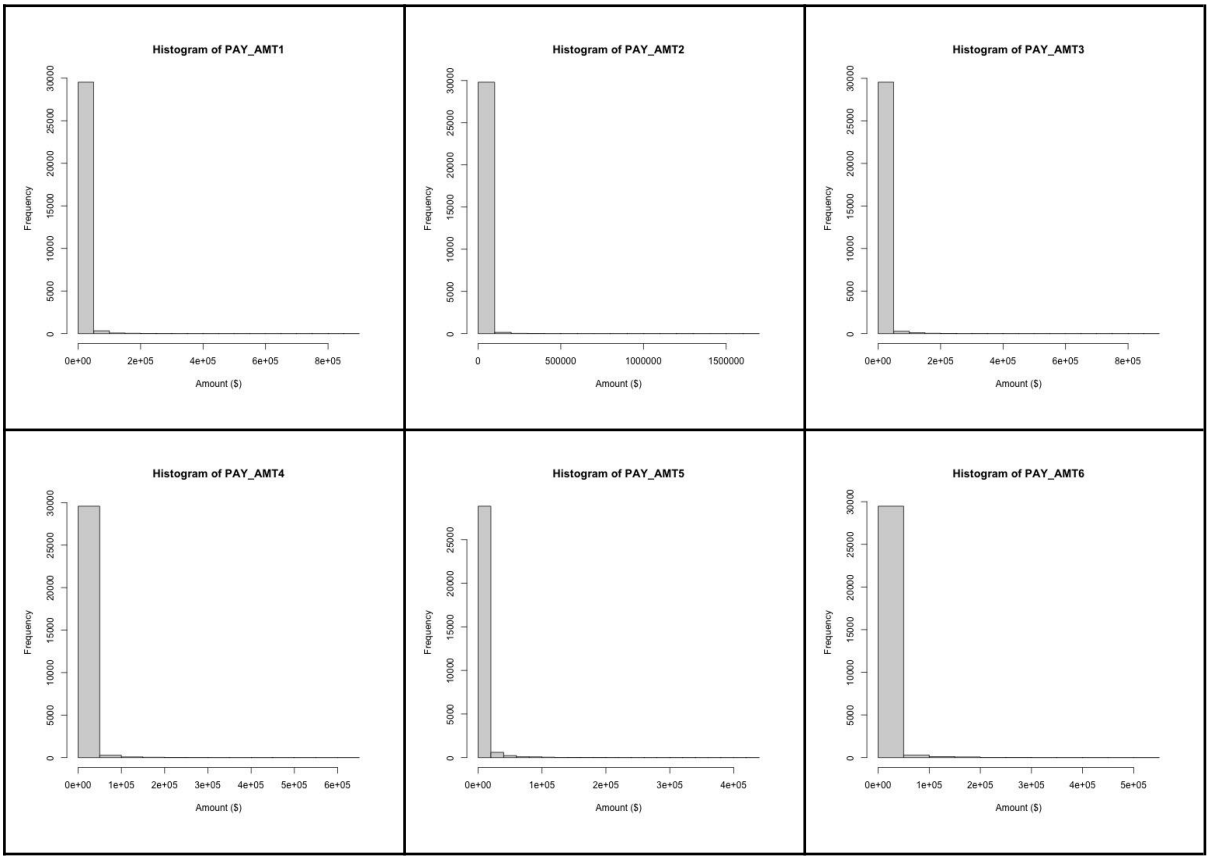
Analysing Credit Behaviour

2.6 Histogram of BILL_AMT



Based on the histograms above, BILL_AMT1 to BILL_AMT6 are right-skewed. So, we need to standardise the values in our data pre-processing.

2.7 Histogram of PAY_AMT



Based on the histogram above, PAY_AMT1 to PAY_AMT6 are right-skewed. So, we need to standardise the values in our data pre-processing.

2.8 Anomalies

Based on on our initial analysis of the dataset, we have realised there are some contradicting data as follows:

1) **Clients who have "Paid in Full" in the recent month marked "Default" (eg. ID24)**
    - There are multiple IDs where they have paid in full for all months, which means they owe nothing to the bank but are marked as default.
2) **Clients with a "Negative Bill" but were still marked as "Default" (eg. ID27)**
    - There are multiple IDs where BILL_AMT_1 < 0 yet default = 1.
    - However a customer with a negative balance does not owe anything to the bank but rather they have paid more than they should so the bank owes them
3) **Inactive clients are marked as "Default" (eg.ID46)**
    - There are multiple IDs with PAY_0...6 all = -2 and BILL_AMT_1...6 all = 0, yet default = 1.
    - However if a customer did not use credit for the last 6 months should owe nothing

4) **Some clients "Bill Amount" is much higher than their "Credit Limit"**
    - This is not realistic as it is a very rare occurrence in real life.
5) **Illogical timeline for delay in payment (eg. ID32)**
    - Multiple IDs with PAY_0 >1 but PAY_1...6=0.
    - However, that is not logical. For eg. there cannot be a 2-month delay in September if there was no delay in August.
    - Eg. ID90 is a similar phenomenon just shifted in time (PAY_2=2 but PAY_3=0, which is, again, logically impossible)

These anomalies could be due to an human input error when gathering the data or due to an incomplete definition of the variables. However, we have chosen not to remove the rows affected from these anomalies as there might be unobservable factors or attributes that affect such anomalies.

2.9 Outliers

In our report, we are not disregarding any of the outliers. During our preliminary analysis, we found that some values of attributes are too centred around the median of the dataset causing an indication of "outliers" from the boxplot of every numerical attributes. Furthermore, we found anomalies explained above as we believe there might be some unobservable variables (variables related to credit cards clients that are omitted from the data set) or conditions (maybe random sampling is not involved, or measurement error from the data set) that cause such anomalies. Considering we don't have much expertise on the credit card domain, we felt it is best to keep as much of the data as possible to maximise the data quality.

3. Data pre-processing

For our data-pre-processing to the Card dataset, we did eight transformations to make our dataset in usable format.

1) **Drop 'ID' attributes.**
   We dropped 'ID' attributes since 'ID' is not necessary to our machine learning models' variables.

2) **Change the naming convention of "PAY_0" to "PAY_1".**
   Based on colnames(card) inspection, we noticed that BILL_AMT and PAY_AMT have structured number naming from 1 to 6. Meanwhile, the number naming for PAY is a bit jumpy from "PAY_0" to "PAY_2". For the purpose of naming consistency, we changed "PAY_0" to "PAY_1".

3) **Correcting for data entry error on "EDUCATION" column.**
   Based on the official data set documentation on UCI, education can only have four values as follows (1 = graduate school; 2 = university; 3 = high school; 4 = others). However, when

we did a preliminary check on "EDUCATION", we found the inconsistency given in the image below.

0	1	2	3	4	5	6
14	10585	14030	4917	123	280	51

We got some unknown values of 0, 5, and 6. Since we got values for others (denoted by values of 4), we would just group 0, 4, 5, and 6 as 4.

4) **Correcting for data entry error on "MARRIAGE" column.**
Similar to the "Education" column, based on the official data set documentation on UCI, "MARRIAGE" column can only have three values (1 = married; 2 = single; 3 = others). However, based on our check, we found the inconsistency given in the image below.

0	1	2	3
54	13659	15964	323

We got 0 as an unknown value. Since we got values for others (denoted by values of 3), we would just group 0 and 3 together as 3.

5) **Change the naming convention of "default.payment.next.month" to "DEFAULT_PAYMENT".**
The naming of "default.payment.next.month" is a bit too long winded. To make the naming more understandable, we would just rename "default.payment.next.month" to "DEFAULT_PAYMENT".

6) **Split the training:test ratio to be 75:25.**
We decided to use k-fold cross validation, that's why we won't split the validation ratio.

7) **For each training set and test set, mutate all categorical attributes to as.factor().**

8) **For each training set and test set, standardise all numerical attributes.**
We need to do feature scaling for training and test sets separately to avoid information leakage. For all numerical attributes in the dataset except for "AGE", the majority of the values are in thousands. However, age is only represented in tens. Therefore, when we run the model, it will make the "AGE" attribute less dominant compared to other numerical attributes. So, it's best practice to standardise all numerical attributes.

4. Feature selection

For the feature selection, we use the Boruta package, which is a feature selection algorithm. We selected Boruta as after doing a comparison with other feature selection method, i.e. LASSO, Boruta produces the best results in accuracy. Boruta itself is a wrapper method, in which the

algorithm checks the importance of each attribute by creating shadow features, based on the core of the *random forest algorithm*.

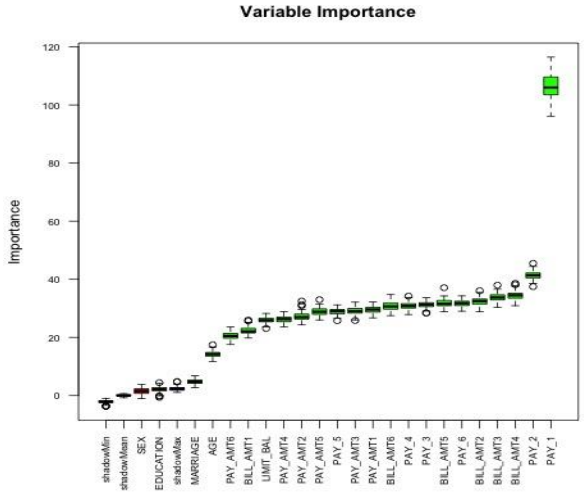


Figure 8: Variable importance (Boruta)

As we can see above, we plot the variable importance with its importance for each attribute (both categorical and continuous). We have set the threshold cut of >= 25 for the importance value. So, for our feature selection, based on Boruta algorithm's results, we will select PAY_1, PAY_2, BILL_AMT4, BILL_AMT3, BILL_AMT2, BILL_AMT5, PAY_6, PAY_3, PAY_4, BILL_AMT6, PAY_AMT1, PAY_AMT3, PAY_5, PAY_AMT5, PAY_AMT2, PAY_AMT4, and LIMIT_BAL.

5. Model selection & evaluation

Before we jump into the model selection, we need to inform you that our model will be run through all the selected attributes from the Boruta model. We did this due to the long computation time if we were to use all the attributes. For the validation method, we will be using ***5-fold cross validation.*** To support the 5-fold, we will be using the *CARET package* in R to automatically apply 5-fold cross validation (via *trainControl*) to our training set. Moreover, we will be using *arithmetic and harmonic average class accuracy, precision, recall, F1-score, and AUC score* to evaluate our model.

For our machine learning models, we will run our models to three datasets (*vanilla, random undersampling,* and *synthetic datasets)*. Our team decided not to run random oversampling as doing oversampling will increase the number of CLASS 1 ("DEFAULT") to three times, which might make the model prone to overfitting.

1. Undersampling
   This works by randomly eliminating observations from the majority class until the class distribution of "DEFAULT_PAYMENT" is balanced. For our undersampling dataset, we have the distribution of 4,981 non-default samples and 4,968 default samples.

2. Synthetic
   We realised that oversampling has the potential of overfitting, and undersampling has the potential of losing significant information on the dataset. One way to overcome this is to create artificial data. Our synthetic data was generated through an R package called ROSE. For the distribution of our synthetic dataset, we have 11,152 non-default samples and 11,348 default samples.

5.1 Model selection

- SVM
  SVM works by finding a hyperplane that maximises the distance of two points from different classes. The classification of each data point will be determined by where the data points lie on the side of the hyperplane (decision boundary).

  a) Original

Kernel	**Validation Accuracy**	**Test Set Accuracy**	**Average Class Accuracy (Arithmetic)**	**Average Class Accuracy (Harmonic)**	**Precision**	**Recall**	**F1-score**
Linear	82.09%	81.41%	63.70%	47.70%	67.43%	31.77%	43.20%
Poly-nomial	78.01%	77.87%	50.47%	2.25%	63.33%	1.14%	2.24%
Radial	82.06%	81.25%	63.20%	46.48%	67.19%	30.69%	42.14%
Sigmoid	77.29%	79.53%	61.20%	43.38%	58.24%	28.18%	37.98%

Based on the overall accuracy above, we will select a *linear* kernel for our SVM prediction on the *original dataset*.

b) Undersampling

Kernel	**Validation Accuracy**	**Test Set Accuracy**	**Average Class Accuracy (Arithmetic)**	**Average Class Accuracy (Harmonic)**	**Precision**	**Recall**	**F1-score**
Linear	69.32%	77.48%	68.23%	64.15%	49.40%	51.56%	50.45%
Poly-nomial	56.40%	40.73%	59.04%	40.62%	26.25%	92.03%	40.85%

Radial	69.47%	77.67%	68.69%	64.88%	49.80%	52.52%	51.12%
Sigmoid	64.64%	63.8%	64.80%	64.75%	33.99%	66.61%	45.01%

Based on the overall accuracy above, we will select a *radial* kernel for our SVM prediction on the *undersampling dataset*.

c) Synthetic

Kernel	**Validation Accuracy**	**Test Set Accuracy**	**Average Class Accuracy (Arithmetic)**	**Average Class Accuracy (Harmonic)**	**Precision**	**Recall**	**F1-score**
Linear	68.75%	77.6%	68.26%	64.12%	49.65%	51.44%	50.53%
Poly-nomial	66.8%	60.05%	65.18%	63.87%	32.57%	74.40%	45.30%
Radial	70.44%	76.73%	69.24%	66.62%	48.01%	55.76%	51.60%
Sigmoid	62.88%	66.61%	62.91%	62.20%	34.59%	56.24%	42.83%

Based on the overall accuracy above, we will select a *radial* kernel for our SVM prediction on the *synthetic dataset*.

- Logistic regression
  Logistic regression is a classification algorithm that uses supervised learning to predict the likelihood of a target variable.

Dataset	**Validation Accuracy**	**Test Set Accuracy**	**Average Class Accuracy (Arithmetic)**	**Average Class Accuracy (Harmonic)**	**Precision**	**Recall**	**F1-score**
Original	81.98%	81.40%	64.82%	51.06%	65.29%	34.95%	45.53%
Under-sampling	70.41%	76.57%	69.08%	66.44%	47.71%	55.58%	51.34%
Synthetic	69.82%	76.93%	69.16%	66.32%	48.37%	55.16%	51.54%

- Neural network

Neural networks are a series of algorithms with interconnected nodes that identify relationships in a dataset. It is mainly used for pattern recognition.

Dataset	Validation Accuracy	Test Set Accuracy	Average Class Accuracy (Arithmetic)	Average Class Accuracy (Harmonic)	Precision	Recall	F1-score
Original	81.90%	81.41%	65.00%	51.55%	65.09%	35.43%	45.89%
Under-sampling	70.61%	76.17%	69.89%	68.06%	47.13%	58.57%	52.23%
Synthetic	73.03%	39.77%	58.71%	38.90%	26.04%	92.81%	40.67%

- Naive Bayes
  Naive Bayes is a classification algorithm that works under the core of Naive Bayes theorem. This model has assumptions of attributes being independent to each other.

Dataset	Validation Accuracy	Test Set Accuracy	Average Class Accuracy (Arithmetic)	Average Class Accuracy (Harmonic)	Precision	Recall	F1-score
Original	76.27%	79.79%	65.04%	54.20%	56.71%	38.49%	45.86%
Under-sampling	64.28%	72.95%	68.16%	67.07%	42.31%	59.53%	49.46%
Synthetic	73.94%	54.96%	64.04%	59.86%	30.53%	80.40%	44.26%

- Random Forest
  Random forest is a classification algorithm with the core of using multiple decision trees. The predictions used in Random forest comes from average output from different decision trees.

Dataset	Validation Accuracy	Test Set Accuracy	Average Class Accuracy (Arithmetic)	Average Class Accuracy (Harmonic)	Precision	Recall	F1-score
Original	80.96%	81.09%	64.24%	49.35%	64.20%	33.87%	44.35%

Under-sampling	69.20%	74.84%	68.95%	67.31%	44.94%	58.33%	50.77%
Synthetic	75.52%	37.85%	57.64%	35.60%	25.49%	93.29%	40.04%

- Decision Tree
  Decision tree uses a random model decision that looks like a tree to determine the possible outcomes from each decision.

Dataset	**Validation Accuracy**	**Test Set Accuracy**	**Average Class Accuracy (Arithmetic)**	**Average Class Accuracy (Harmonic)**	**Precision**	**Recall**	**F1-score**
Original	81.14%	81.09%	62.72%	45.25%	66.94%	29.62%	41.06%
Under-sampling	62.72%	80.04%	66.70%	58.06%	56.82%	42.69%	48.75%
Synthetic	63.53%	28.35%	52.90%	15.93%	23.32%	97.12%	37.61%

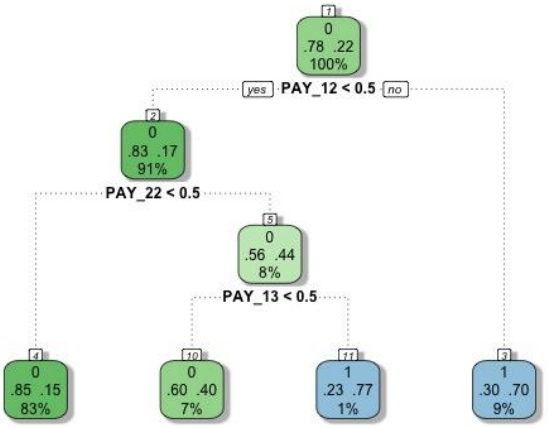


Figure 9: Decision tree for vanilla dataset

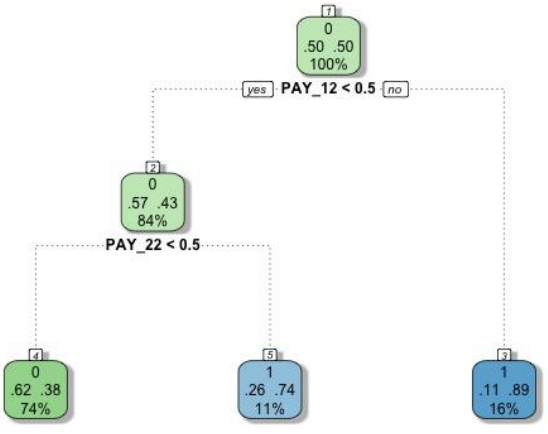


Figure 10: Decision tree for undersampling dataset

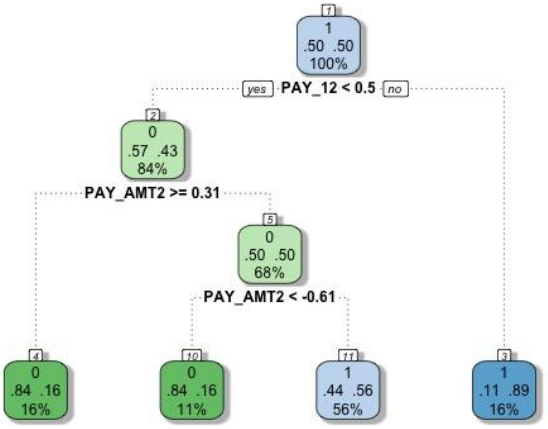


Figure 11: Decision tree for synthetic dataset

Models	Pros	Cons
SVM	1) Works well with a clear organised data 2) Effective in high dimensional spaces 3) Memory Efficient 4) Effective when no.of dimensions greater than the no. of samples	1) Training time for large data sets or noisy data is higher 2) Does not provide probability estimates, expensive to calculate using cross-validation
Logistic Regression	1) Simplest machine learning algorithm 2) Adaption to changes in input data 3) If dataset has features that is easily separable, it is very efficient 4) Outputs well calibrated probabilities	1) May lead to overfitting on high dimensional data 2) Nonlinear problems cant' be solved 3) High Data maintenance 4) Cant handle missing data
Neural Networks	1) Flexible - can be used for both classification and regression 2) Good to model nonlinear data with a large amount of inputs 3) Operating speed 4) Wide range of applications 5) Adaption to changes in input data 6) Effective noise filtering of data	1) Depend a lot on training data - lead to overfitting and generalisation 2) Duration of development 3) "Black-Box" problem 4) Computationally expensive
Naive Bayes	1) Algorithm is very fast due to independent features assumption 2) Works well with high-dimensional data such as text-classification, spam detection, etc 3) Performs well if input variables are categorical instead of numerical	1) Independent feature assumption usually does not hold in real life 2) Limited applicability in real-world use cases 3) Less accurate due to violation of assumption
Random Forest	1) Performs with categorical and numerical data - no data scaling/transformation required 2) Performs implicit feature selection and generates uncorrelated decision trees 3) Works well with dataset with high number of features	1) Not easily interpretable - does not provide visibility into coefficients like linear regression, for e.g 2) Computationally intensive for large datasets 3) Very little control over what model does since it is a black box

		algorithm
Decision Tree	1) Little to no data preprocessing (e.g one-hot encoding, dummy variables) required 2) Fast and efficient compared to KNN and other classification algorithms 3) Non-parametric - no assumptions are made about the spatial distribution and classifier structure 4) Scale invariant - feature scaling and normalisation not required	1) Training requires more time and is relatively expensive compared to other algorithms 2) Can easily lead to overfitting of the data 3) Relatively unstable as tiny variations in the data might cause a completely different tree being generated 4) Limited prediction powers, especially when dealing with complex, large datasets

Figure 12: Analysis of pros & cons of ML models

ADABoost, LightGBM and XGBoost are variations of tree-based ensemble algorithms, which build models in sequence using the entire dataset, with each model building on the previous model's error. Since these algorithms use Ensemble Learning, which averages the predictions across multiple models to get a prediction that would be more stable and generalise better, risk of overfitting is reduced while maintaining strong prediction performance.

	ADABoost	**LightGBM**	**XGBoost**
Framework	Tree-based ensemble algorithms for supervised ML problems.		
Tree Symmetry	Asymmetric	Asymmetric (tree growth is leaf-wise)	Asymmetric (tree growth is level-wise)
Splitting Method	Forest of decision stumps	Gradient-based one-side sampling (GOSS)	Pre-sorted, histogram-based algorithm
Type of Boosting	Adaptive	Gradient	Gradient
Pros	1) Less prone to overfitting 2) Accuracy of weak classifiers can be improved 3) Better used to classify text and images over binary classification	1) Higher Efficiency 2) Faster Training Speed 3) Low memory usage which leads to better accuracy 4) Compatible with large and complex datasets	1) Highly flexible 2) Supports regularisation 3) Uses the power of parallel processing
Cons	1) Sensitive to outliers 2) Not compatible with noisy data	1) Sensitive to overfitting	1) Not compatible with outliers and unstructured data

Figure 13: Characteristics of ADABoost, LightGBM, XGBoost

- XGBoost

Extreme Gradient Boosting (XGBoost) provides a gradient boosting framework for the decision tree algorithm.

Dataset	Validation Accuracy	Test Set Accuracy	Average Class Accuracy (Arithmetic)	Average Class Accuracy (Harmonic)	Precision	Recall	F1-score
Original	81.94%	81.41%	64.48%	50.07%	65.93%	33.99%	44.86%
Under-sampling	70.54%	77.57%	70.47%	68.15%	49.64%	57.67%	53.36%
Synthetic	77.93%	34.27%	56.02%	28.61%	24.67%	95.20%	39.18%

- *ADABoost*
  Adaptive Boosting (AdaBoost) is a modelling technique that combines numerous weak attributes into a single strong classifier attribute.

Dataset	Validation Accuracy	Test Set Accuracy	Average Class Accuracy (Arithmetic)	Average Class Accuracy (Harmonic)	Precision	Recall	F1-score
Original	81.57%	81.28%	63.86%	48.45%	66.10%	32.49%	43.57%
Under-sampling	70.15%	78.08%	69.51%	66.08%	50.67%	54.08%	52.32%
Synthetic	74.25%	34.30%	55.43%	29.32%	24.45%	93.47%	38.76%

- *LightGBM*
  Light Gradient Boosted Machine (LightGBM) is a GB framework based on a decision tree algorithm, mainly used for classification, ranking and other ML tasks.

Dataset	Validation Accuracy	Test Set Accuracy	Average Class Accuracy (Arithmetic)	Average Class Accuracy (Harmonic)	Precision	Recall	F1-score
Original	81.97%	81.32%	64.38%	49.92%	65.47%	33.87%	44.65%

Under-sampling	71.17%	78.23%	69.31%	65.58%	51.01%	53.24%	52.10%
Synthetic	76.75%	33.80%	55.76%	27.70%	24.55%	95.32%	39.04%

5.2 AUC–ROC Curve

For the plot of AUC–ROC, we will be using the CARET package to optimise for ROC, and we will plot the curve with the help of the *MLEval* package using the *evalm() function*. We generated the AUC – ROC curve for the original, undersampling, and synthetic data ***to the test set.***

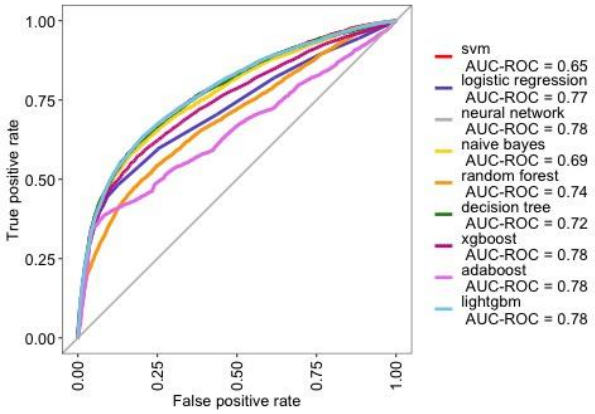


Figure 14: AUC–ROC Curve of models from original set

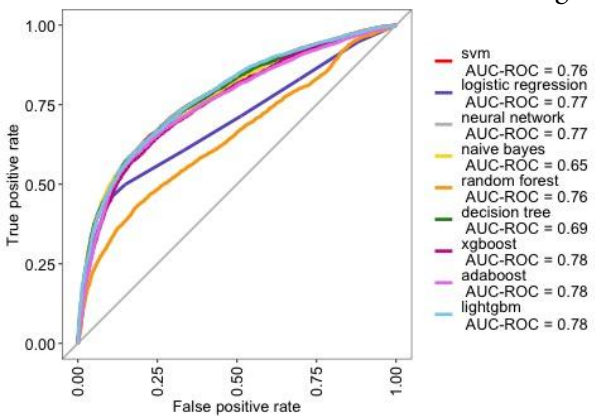


Figure 15: AUC–ROC Curve of models from undersampling set

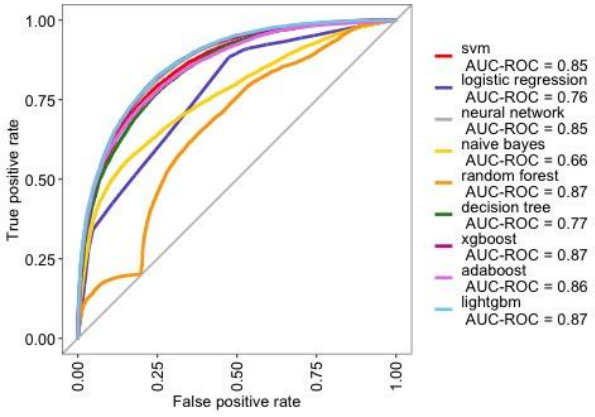


Figure 16: AUC–ROC Curve of models from synthetic set

6. Room for improvements

- **Handle Overfitting**

  For our ***synthetic*** models evaluation, we noticed there is a case of overfitting as we produced a high accuracy for validation, but a low accuracy for the test set. Three ways that our team could possibly do to overcome overfitting are by:

  1) Increase the number of folds in cross-validation. We can use 10-fold cross validation instead when running our models. However, due to the number of models that we picked, along with each model computation time, our initial pick was to choose 5-fold cross validation.
  2) Reduce the number of feature selection. On our current model, we selected at least 15 attributes to run every model. However, this could potentially raise the issue of overfitting. So, we would produce a better model if we were to select fewer but significant attributes.
  3) Implement regularization. We can implement *L1/L2 regularization* to give a "penalty" term to our model, that way we can overcome the case of overfitting.

- **Logistic Regression Cutoff**

  We would like to lower the false negatives in our case study as much as feasible; thus, we can set a threshold number that boosts our TPR while decreasing our FPR. For instance, we can select a threshold of 0.3, generate a confusion matrix, and test the model's accuracy to check for any improvement.

- **SVM Grid search / Hyperparameter tuning**

  To further improve our SVM model, we'll be applying grid search to find the best gamma and cost parameters. The simplest approach for hyperparameter tuning is grid search. With grid search, we try every possible combination of the values in this grid to calculate some performance measures using cross-validation. The best set of hyperparameters for the grid is where the value of cross-average validation is at its highest. For both original and undersampling dataset, we are using linear as the kernel, therefore we can tune for the best Cost parameter. Meanwhile, for synthetic dataset, we have the best model when we use radial as the kernel, so we can use grid search to find the best Cost and Gamma parameter.

- **Tree-based model tuning**

  Since models such as random forest, decision tree, XGBoost, ADABoost, and LightGBM use tree-based algorithms for the classification, we can proceed to do similar tuning to these models. We can do grid search to find the optimal number and depth of the tree, then the number of splits in the tree. For models like XGBoost, ADABoost, and LightGBM, we can further tune the gamma parameter, learning rate, minimum instances in the child node, and other necessary parameters by randomised search.

- **Anomalies justification**

Based on our findings, we have numerous anomalies inside the dataset. However, due to our limited knowledge in the credit card domain, we couldn't justify the anomalies. Moreover, from the official repository of the dataset, we could not also find any justification on the anomalies. Therefore, this justification will impact our models accuracy overall. Moreover, we also found some data entry errors that could significantly affect our prediction. For example, we have some negative values on BILL_AMT1. Upon checking, the negative value is supposed to indicate refund. However, since we don't have any data on the baseline of the correct amount, we could not correct the data such that the BILL_AMT1 becomes "correct payment + refund". So, one room for improvement is for the author(s) of the dataset to clarify and justify some of the anomalies or entry errors, so that we are able to process or correct it during our data pre-processing, further improving our models.

- **Quality of data**

  We believe the current dataset itself is not enough to build our classification of credit card's default. Therefore, we need to obtain different datasets to further improve our data set quality.

  1. We need to have another credit card client's data set that focuses specifically on the client's credit history, client's type of credits, length of the credit history, etc.
  2. We need to have a longer window period of the dataset. This will help us in streamlining the patterns of default credit card clients.
  3. Adding income from the client's dataset might improve our model as it could help in identifying the client's spending pattern that could potentially lead them to default.
  4. We need to have more recent data, as this data originated from 2005, the data might not be representative anymore.

7. Summary

We will use F1-score and ROC index metric to pick the best model for original, undersampling, and synthetic data sets respectively.

a) Original

Model	F1-score	ROC Index
SVM (*Linear*)	43.20%	0.65
Logistic Regression	45.53%	0.77
Neural Network	**45.89%**	**0.78**
Naive Bayes	45.86%	0.69

Model	F1-score	ROC Index
Decision Tree	41.06%	0.72
XGBoost	44.86%	0.78
ADABoost	43.57%	0.78
LightGBM	44.65%	0.78

Random Forest	44.35%	0.74

Based on the F1-score and ROC index above, we will use ***Neural Network*** as our best model to train the original training set over the test set.

b) Undersampling

Model	**F1-score**	**ROC Index**
SVM (*Radial*)	51.12%	0.76
Logistic Regression	51.34%	0.77
Neural Network	52.23%	0.77
Naive Bayes	49.46%	0.65
Random Forest	50.77%	0.76

Model	**F1-score**	**ROC Index**
Decision Tree	48.75%	0.69
XGBoost	**53.36%**	**0.78**
ADABoost	52.32%	0.78
LightGBM	52.10%	0.78

Based on the F1-score and ROC index above, we will use ***XGBoost*** as our best model to train the undersampling training set over the test set.

c) Synthetic

Model	**F1-score**	**ROC Index**
SVM (*Radial*)	**51.60%**	**0.85**
Logistic Regression	51.54%	0.76
Neural Network	40.67%	0.85
Naive Bayes	44.26%	0.66
Random Forest	40.04%	0.87

Model	**F1-score**	**ROC Index**
Decision Tree	37.61%	0.77
XGBoost	39.18%	0.87
ADABoost	38.76%	0.86
LightGBM	39.04%	0.87

Based on the F1-score and ROC index above, we will use ***SVM*** as our best model to train the synthetic training set over the test set.

8. References

AdaBoost implementation and tuning for high dimensional feature space in R. (2017, August 4). Data Science Stack Exchange. Retrieved November 11, 2022, from https://datascience.stackexchange.com/questions/21959/adaboost-implementation-and-tuning-for-high-dimensional-feature-space-in-r

Brownlee, J. (2016, February 5). *Tune Machine Learning Algorithms in R (random forest case study)*. Machine Learning Mastery. Retrieved November 11, 2022, from https://machinelearningmastery.com/tune-machine-learning-algorithms-in-r/

caret: Classification and Regression Training. (2022, August 9). The Comprehensive R Archive Network. Retrieved November 11, 2022, from https://cran.r-project.org/web/packages/caret/caret.pdf

default of credit card clients Data Set. (2016, January 26). UCI Machine Learning Repository. Retrieved November 11, 2022, from https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients

Exploratory Data Analysis in R Programming. (2022, July 1). GeeksforGeeks. Retrieved November 11, 2022, from https://www.geeksforgeeks.org/exploratory-data-analysis-in-r-programming/

Fraj, B. (2017, December 24). *In Depth: Parameter tuning for Gradient Boosting | by Mohtadi Ben Fraj | All things AI*. Medium. Retrieved November 11, 2022, from https://medium.com/all-things-ai/in-depth-parameter-tuning-for-gradient-boosting-3363992e9bae

Gradient Boosting and Parameter Tuning in R. (n.d.). Kaggle. Retrieved November 11, 2022, from https://www.kaggle.com/code/camnugent/gradient-boosting-and-parameter-tuning-in-r/notebook

Gradient Boosting and Parameter Tuning in R. (n.d.). Kaggle. Retrieved November 11, 2022, from https://www.kaggle.com/code/camnugent/gradient-boosting-and-parameter-tuning-in-r/notebook

John, C. R. (2020, February 12). *MLeval: Machine Learning Model Evaluation*. CRAN. Retrieved November 11, 2022, from https://cran.r-project.org/web/packages/MLeval/MLeval.pdf

Joseph, R. (2018, December 29). *Grid Search for model tuning. A model hyperparameter is a... | by Rohan Joseph*. Towards Data Science. Retrieved November 11, 2022, from https://towardsdatascience.com/grid-search-for-model-tuning-3319b259367e

Kursa, M. B. (2020, May 21). Boruta: Wrapper Algorithm for All Relevant Feature Selection. Retrieved November 11, 2022, from https://cran.r-project.org/web/packages/Boruta/Boruta.pdf

Logistic Regression Pros & Cons | HolyPython.com. (n.d.). Holy Python. Retrieved November 1, 2022, from https://holypython.com/log-reg/logistic-regression-pros-cons/

Losing Paradise. (2012, October 29). YouTube. Retrieved November 11, 2022, from https://towardsdatascience.com/essential-things-you-need-to-know-about-f1-score-dbd973bf1a3/

Vadapalli, P. (2022, September 30). *Naive Bayes Classifier: Pros & Cons, Applications & Types Explained*. upGrad. Retrieved November 11, 2022, from https://www.upgrad.com/blog/naive-bayes-classifier/

Singh, J. (n.d.). *Random Forest: Pros and Cons. Random forest is a supervised learning... | by Jagandeep Singh*. DataDrivenInvestor. Retrieved November 11, 2022, from https://medium.datadriveninvestor.com/random-forest-pros-and-cons-c1c42fb64f04

Top 5 advantages and disadvantages of Decision Tree Algorithm. (n.d.). Dhiraj K. Retrieved November 11, 2022, from https://dhirajkumarblog.medium.com/top-5-advantages-and-disadvantages-of-decision-tree-algorithm-428ebd199d9a

Pros and Cons of Decision Trees. (2018, October 8). LinkedIn. Retrieved November 11, 2022, from https://www.linkedin.com/pulse/pros-cons-decision-trees-aashima-yuthika/

Nikulski, J. (2020, March 16). *The Ultimate Guide to AdaBoost, random forests and XGBoost*. Towards Data Science. Retrieved November 11, 2022, from https://towardsdatascience.com/the-ultimate-guide-to-adaboost-random-forests-and-xgboost-7f9327061c4f

Gera, D. (2020, September 9). *Boosting Algorithms: AdaBoost, Gradient Boosting, XGB, Light GBM and CatBoost*. Medium. Retrieved November 11, 2022, from https://medium.com/@divyagera2402/boosting-algorithms-adaboost-gradient-boosting-xgb-light-gbm-and-catboost-e7d2dbc4e4ca

Patil, P. (2018, March 24). *What is Exploratory Data Analysis? | by Prasad Patil*. Towards Data Science. Retrieved November 17, 2022, from https://towardsdatascience.com/exploratory-data-analysis-8fc1cb20fd15

9. Appendix

Attribute Information:

- **ID**: ID of each client
- **LIMIT_BAL**: Amount of given credit in NT dollars (includes individual and family/supplementary credit)
- **SEX**: Gender (1 = male, 2 = female)
- **EDUCATION**: (1 = graduate school, 2 = university, 3 = high school, 0,4,5,6 = others)
- **MARRIAGE**: Marital status (0 = others, 1 = married, 2 = single, 3 = divorce)
- **AGE**: Age in years
- **PAY_1**: Repayment status in September, 2005
  (-2 = No consumption, -1 = paid in full, 0 = use of revolving credit (paid minimum only), 1 = payment delay for one month, 2 = payment delay for two months, ... 8 = payment delay for eight months, 9 = payment delay for nine months and above)
- **PAY_2**: Repayment status in August, 2005 (scale same as above)
- **PAY_3**: Repayment status in July, 2005 (scale same as above)
- **PAY_4**: Repayment status in June, 2005 (scale same as above)
- **PAY_5**: Repayment status in May, 2005 (scale same as above)
- **PAY_6**: Repayment status in April, 2005 (scale same as above)
- **BILL_AMT1**: Amount of bill statement in September, 2005 (NT dollar)
- **BILL_AMT2**: Amount of bill statement in August, 2005 (NT dollar)
- **BILL_AMT3**: Amount of bill statement in July, 2005 (NT dollar)
- **BILL_AMT4**: Amount of bill statement in June, 2005 (NT dollar)
- **BILL_AMT5**: Amount of bill statement in May, 2005 (NT dollar)
- **BILL_AMT6**: Amount of bill statement in April, 2005 (NT dollar)
- **PAY_AMT1**: Amount of previous payment in September, 2005 (NT dollar)
- **PAY_AMT2**: Amount of previous payment in August, 2005 (NT dollar)
- **PAY_AMT3**: Amount of previous payment in July, 2005 (NT dollar)
- **PAY_AMT4**: Amount of previous payment in June, 2005 (NT dollar)
- **PAY_AMT5**: Amount of previous payment in May, 2005 (NT dollar)
- **PAY_AMT6**: Amount of previous payment in April, 2005 (NT dollar)
- **default.payment.next.month**: Default payment (1=yes, 0=no)

BT2103ProjectCode

November 11, 2022

[ ]:
```
library(dplyr)
library(ROSE) # Random OS, US, SMOTE
library(mlbench)
library(caret)
library(Boruta)
library(varImp)
library(tidyverse)
library(reshape2)
library(ggplot2)
library(e1071)
library(caTools)
library(ROCR)
library(naniar) # for missing values
library(plyr)
library(corrplot)
```

[ ]:
```
set.seed(1234)
card <- read.csv("~/Downloads/card.csv", skip = 1)
```

0.0.1 Exploratory Data Analysis

[ ]:
```
data <- read.table("card.csv", sep = ",", skip = 1, header=T)
#view first 6 rows of dataset
head(data)
#vis_miss(data)
```

[ ]:
```
data$DEFAULT = as.factor(data$default.payment.next.month)
data$GENDER <- ifelse(data$SEX==1, "Male", "Female")
ggplot(data = data, mapping = aes(x = GENDER, fill = DEFAULT)) +
  geom_bar() +
  ggtitle("Gender") +
  stat_count(aes(label = ..count..), geom = "label")
```

[ ]:
```
#Merging 0, 5 and 6 to 4(others)
data$EDUCATION = ifelse(data$EDUCATION == 0 |data$EDUCATION == 5 |␣
 ↪data$EDUCATION == 6, 4, data$EDUCATION)
#Converting EDUCATION to a categorical variable
data$EDUCATION = as.factor(data$EDUCATION)
```

```
#Plotting Bar graph for EDUCATION
ggplot(data = data, mapping = aes(x = EDUCATION, fill = DEFAULT)) +
  geom_bar() +
  ggtitle("EDUCATION") +
  stat_count(aes(label = ..count..), geom = "label")
```

[ ]:
```
calDefaultPerc = function(x, colInd, data.level){
  default_count = c()
  default_perc = c()

  for (i in 1:length(data.level)) {
    print("start of for...")
    total_count = count(x[x[colInd] == data.level[i], ], vars = "DEFAULT")
    count_def = total_count[total_count$DEFAULT == 1, 2]
    total = total_count[total_count$DEFAULT == 1, 2] +␣
 ↪total_count[total_count$DEFAULT == 0, 2]
    print(i)
    perc = round(count_def/total * 100, digits = 2)

    default_count = c(default_count, count_def)
    default_perc = c(default_perc, perc)
    print("end of for...")
  }
  print("outer for....")
  #data.level
  length(default_count)
  length(default_perc)

  paste("data.level: ", length(data.level))
  paste("default_count: ", length(default_count))
  paste("default_perc: ", length(default_perc))

  default_perc_df = data.frame(level = data.level, default.count =␣
 ↪default_count,
                               percentage = default_perc)
  print("data frame created....")
  return(default_perc_df)
}
```

[ ]:
```
default_count_perc = calDefaultPerc(x = data, colInd = 4,
                                    data.level = levels(data$EDUCATION))
# Bar Graph
ggplot(data = default_count_perc, mapping = aes(x = reorder(level, percentage),␣
 ↪y = percentage)) +
  geom_bar(stat = "identity", fill = "#b3e569") +
  coord_flip() +
  xlab("Education") +
```

```
  ylab("Percentage of Defaulters") +
  ggtitle("Education Level vs Default") +
  geom_label(label = paste(default_count_perc$percentage, "%"))
```

[ ]:
```
ggplot(data, aes(x=as.factor(EDUCATION), y=LIMIT_BAL)) +␣
↪geom_boxplot(fill="slateblue", alpha=0.2) + xlab("Education Level")
```

[ ]:
```
# Adding one feature marital status
addMaritalStatus = function(x, colInd, newColInd){
  for (i in 1: nrow(x)) {
    if(x[i, colInd] == 1){
      x[i, newColInd] = "Married"
    }
    else if (x[i, colInd] == 2) {
      x[i, newColInd] = "Not Married"
    }
    else{
      x[i, newColInd] = "Others"
    }

  }
  return(x)
}
```

[ ]:
```
# Merging 0 to 3(others)
data$MARRIAGE = ifelse(data$MARRIAGE == 3, 0, data$MARRIAGE)
data$MARRIAGE = as.factor(data$MARRIAGE)
table(data$MARRIAGE)
data$MARITALSTATUS = NA
data = addMaritalStatus(data, which(colnames(data) == "MARRIAGE"),
                            which(colnames(data) == "MARITALSTATUS"))
# convert to categorical data
data$MARITALSTATUS = as.factor(data$MARITALSTATUS)

# Bar graph for marital status
ggplot(data = data, mapping = aes(x = MARITALSTATUS, fill = DEFAULT)) +
  geom_bar() +
  xlab("Marital status") +
  ggtitle(" Defaulters on Marital Status") +
  stat_count(aes(label = ..count..), geom = "label")
```

[ ]:
```
# Explore Marital status vs default
marital_default_per_df = calDefaultPerc(x = data,
                                        colInd = which(colnames(data) ==␣
↪"MARITALSTATUS"),
                                        data.level =␣
↪levels(data$MARITALSTATUS))
```

```
marital_default_per_df

# Bar Graph
ggplot(data = marital_default_per_df, mapping = aes(x = reorder(level,␣
 ↪percentage), y = percentage)) +
  geom_bar(stat = "identity", fill = "#0000FF") +
  coord_flip() +
  xlab("Marital Status") +
  ylab("Percentage of Defaulters") +
  ggtitle("Marital Status vs Default") +
  geom_label(label = paste(marital_default_per_df$percentage, "%"))
```

[ ]:
```
data1 = data %>%
  mutate(
    # Create categories
    age_group = dplyr::case_when(
      AGE <= 30             ~ "20-30",
      AGE > 30 & AGE <= 40 ~ "30-40",
      AGE > 40 & AGE <= 50 ~ "40-50",
      AGE > 50 & AGE <= 60 ~ "50-60",
      AGE > 60 & AGE <= 70 ~ "60-70",
      AGE > 70 & AGE <= 80 ~ "70-80"
    ),
    # Convert to factor
    age_group = factor(
      age_group,
      level = c("20-30", "30-40","40-50", "50-60", "60-70", "70-80")
    )
  )
data1$age_group = as.factor(data1$age_group)
levels(data1$age_group)
age_default_per_df = calDefaultPerc(x = data1,
                                    colInd = which(colnames(data1) ==␣
 ↪"age_group"),
                                    data.level = levels(data1$age_group))
age_default_per_df
# Bar Graph
ggplot(data = age_default_per_df, mapping = aes(x = reorder(level, percentage),␣
 ↪y = percentage)) +
  geom_bar(stat = "identity", fill = "#0000FF") +
  coord_flip() +
  xlab("Age Group") +
  ylab("Percentage of Defaulters") +
  ggtitle("Age Group vs Default") +
  geom_label(label = paste(age_default_per_df$percentage, "%"))
```

[ ]:
```
dataset = data = mutate_all(testset, function(x) as.numeric(as.character(x)))
data.cor = cor(dataset)
table(data.cor)
corrplot(data.cor, tl.srt = 60,tl.cex = 0.5,method = "number",number.cex=0.3)

pay1 = cor.test(dataset$DEFAULT_PAYMENT,dataset$PAY_1,method ="pearson")
pay2 = cor.test(dataset$DEFAULT_PAYMENT,dataset$PAY_2,method ="pearson")
pay3 = cor.test(dataset$DEFAULT_PAYMENT,dataset$PAY_3,method ="pearson")
pay4 = cor.test(dataset$DEFAULT_PAYMENT,dataset$PAY_4,method ="pearson")
pay5 = cor.test(dataset$DEFAULT_PAYMENT,dataset$PAY_5,method ="pearson")
pay6 = cor.test(dataset$DEFAULT_PAYMENT,dataset$PAY_6,method ="pearson")

tab <-␣
 ↪matrix(c(pay1$estimate,pay2$estimate,pay3$estimate,pay4$estimate,pay5$estimate,pay6$estimat
 ↪nrow = 1)
colnames(tab) <- c("Pay_1", "Pay_2","Pay_3","Pay_4","Pay_5","Pay_6")
rownames(tab) <- c("Default Payment")
tab.final <- as.table(tab)
kable(tab.final) %>% kable_styling(bootstrap_options = c("striped", "hover"))␣
 ↪%>%
  kable_styling(latex_options = "HOLD_position")
```

0.0.2 Data pre-processing

[ ]:
```
card <- card[, c(-1)]
```

Drop the 'ID' column as it is not needed inside our prediction.

[ ]:
```
colnames(card)[colnames(card) == "PAY_0"] ="PAY_1"
```

We will change the column name of "PAY_O" to "PAY_1" to make the naming convention consistent with BILL_AMT and PAY_AMT.

[ ]:
```
table(card$EDUCATION)
card[card$EDUCATION %in% c(0, 5, 6), "EDUCATION"] <- 4
```

Based on the documentation of the data set, the values of 0, 5, 6 in EDUCATION column are unknown / undocumented, so we can move it to "OTHERS" (denoted by 4).

[ ]:
```
table(card$MARRIAGE)
card[card$MARRIAGE == 0, "MARRIAGE"] <- 3
```

Based on the documentation of the data set, the values of 0 in MARRIAGE column are unknown / undocumented, so we can move it to "OTHERS" (denoted by 3).

[ ]:
```
colnames(card)[colnames(card) == "default.payment.next.month"]␣
 ↪="DEFAULT_PAYMENT"
```

We change the column name for readability purpose.

0.0.3 Split train:test to 75:25

[ ]:
```
set.seed(1234)
index <- 1:nrow(card)
n = length(card$LIMIT_BAL)
testindex <- sample(index, trunc(n)/4)
testset <- card[testindex,]
trainset <- card[-testindex,]
```

[ ]:
```
# Convert the categorical as factor
trainset <- trainset %>% mutate_at(c("SEX", "EDUCATION", "MARRIAGE", "PAY_1",␣
 ↪"PAY_2", "PAY_3", "PAY_4", "PAY_5", "PAY_6", "DEFAULT_PAYMENT"), as.factor)
```

Change all categorical variables into factor variables.

[ ]:
```
trainset <- trainset %>%
    mutate_at(c("LIMIT_BAL", "AGE", "BILL_AMT1", "BILL_AMT2", "BILL_AMT3",␣
 ↪"BILL_AMT4", "BILL_AMT5", "BILL_AMT6", "PAY_AMT1", "PAY_AMT2", "PAY_AMT3",␣
 ↪"PAY_AMT4", "PAY_AMT5", "PAY_AMT6"), scale)

trainset <- trainset %>%
    mutate_at(c("LIMIT_BAL", "AGE", "BILL_AMT1", "BILL_AMT2", "BILL_AMT3",␣
 ↪"BILL_AMT4", "BILL_AMT5", "BILL_AMT6", "PAY_AMT1", "PAY_AMT2", "PAY_AMT3",␣
 ↪"PAY_AMT4", "PAY_AMT5", "PAY_AMT6"), as.numeric)
```

[ ]:
```
# Convert the categorical as factor
testset <- testset %>% mutate_at(c("SEX", "EDUCATION", "MARRIAGE", "PAY_1",␣
 ↪"PAY_2", "PAY_3", "PAY_4", "PAY_5", "PAY_6", "DEFAULT_PAYMENT"), as.factor)
```

Change all categorical variables into factor variables.

[ ]:
```
testset <- testset %>%
    mutate_at(c("LIMIT_BAL", "AGE", "BILL_AMT1", "BILL_AMT2", "BILL_AMT3",␣
 ↪"BILL_AMT4", "BILL_AMT5", "BILL_AMT6", "PAY_AMT1", "PAY_AMT2", "PAY_AMT3",␣
 ↪"PAY_AMT4", "PAY_AMT5", "PAY_AMT6"), scale)

testset <- testset %>%
    mutate_at(c("LIMIT_BAL", "AGE", "BILL_AMT1", "BILL_AMT2", "BILL_AMT3",␣
 ↪"BILL_AMT4", "BILL_AMT5", "BILL_AMT6", "PAY_AMT1", "PAY_AMT2", "PAY_AMT3",␣
 ↪"PAY_AMT4", "PAY_AMT5", "PAY_AMT6"), as.numeric)
```

[ ]:
```
jpeg("~/Documents/rplot.jpg")
hist(card$BILL_AMT1, main = "Histogram of BILL_AMT1", xlab = "Amount ($)")
dev.off()
jpeg("~/Documents/rplot1.jpg")
hist(card$BILL_AMT2, main = "Histogram of BILL_AMT2", xlab = "Amount ($)")
dev.off()
jpeg("~/Documents/rplot2.jpg")
```

```
hist(card$BILL_AMT3, main = "Histogram of BILL_AMT3", xlab = "Amount ($)")
dev.off()
jpeg("~/Documents/rplot3.jpg")
hist(card$BILL_AMT4, main = "Histogram of BILL_AMT4", xlab = "Amount ($)")
dev.off()
jpeg("~/Documents/rplot4.jpg")
hist(card$BILL_AMT5, main = "Histogram of BILL_AMT5", xlab = "Amount ($)")
dev.off()
jpeg("~/Documents/rplot5.jpg")
hist(card$BILL_AMT6, main = "Histogram of BILL_AMT6", xlab = "Amount ($)")
dev.off()
```

[ ]:
```
jpeg("~/Documents/rplot.jpg")
hist(card$PAY_AMT1, main = "Histogram of PAY_AMT1", xlab = "Amount ($)")
dev.off()
jpeg("~/Documents/rplot1.jpg")
hist(card$PAY_AMT2, main = "Histogram of PAY_AMT2", xlab = "Amount ($)")
dev.off()
jpeg("~/Documents/rplot2.jpg")
hist(card$PAY_AMT3, main = "Histogram of PAY_AMT3", xlab = "Amount ($)")
dev.off()
jpeg("~/Documents/rplot3.jpg")
hist(card$PAY_AMT4, main = "Histogram of PAY_AMT4", xlab = "Amount ($)")
dev.off()
jpeg("~/Documents/rplot4.jpg")
hist(card$PAY_AMT5, main = "Histogram of PAY_AMT5", xlab = "Amount ($)")
dev.off()
jpeg("~/Documents/rplot5.jpg")
hist(card$PAY_AMT6, main = "Histogram of PAY_AMT6", xlab = "Amount ($)")
dev.off()
```

[ ]:
```
levels(trainset$PAY_1) <- c("-2", "-1", "0",  "1",  "2",  "3",  "4",  "5", ␣
 ↪"6",  "7",  "8" )
levels(trainset$PAY_2) <- c("-2", "-1", "0",  "1",  "2",  "3",  "4",  "5", ␣
 ↪"6",  "7",  "8" )
levels(trainset$PAY_3) <- c("-2", "-1", "0",  "1",  "2",  "3",  "4",  "5", ␣
 ↪"6",  "7",  "8" )
levels(trainset$PAY_4) <- c("-2", "-1", "0",  "1",  "2",  "3",  "4",  "5", ␣
 ↪"6",  "7",  "8" )
levels(trainset$PAY_5) <- c("-2", "-1", "0",  "1",  "2",  "3",  "4",  "5", ␣
 ↪"6",  "7",  "8" )
levels(trainset$PAY_6) <- c("-2", "-1", "0",  "1",  "2",  "3",  "4",  "5", ␣
 ↪"6",  "7",  "8" )


levels(testset$PAY_1) <- c("-2", "-1", "0",  "1",  "2",  "3",  "4",  "5",  "6",␣
 ↪ "7",  "8" )
```

```
levels(testset$PAY_2) <- c("-2", "-1", "0",  "1",  "2",  "3",  "4",  "5",  "6",␣
 ↪ "7",  "8" )
levels(testset$PAY_3) <- c("-2", "-1", "0",  "1",  "2",  "3",  "4",  "5",  "6",␣
 ↪ "7",  "8" )
levels(testset$PAY_4) <- c("-2", "-1", "0",  "1",  "2",  "3",  "4",  "5",  "6",␣
 ↪ "7",  "8" )
levels(testset$PAY_5) <- c("-2", "-1", "0",  "1",  "2",  "3",  "4",  "5",  "6",␣
 ↪ "7",  "8" )
levels(testset$PAY_6) <- c("-2", "-1", "0",  "1",  "2",  "3",  "4",  "5",  "6",␣
 ↪ "7",  "8" )
```

[ ]:
```
boruta_output <- Boruta(DEFAULT_PAYMENT ~ ., data=trainset, doTrace=0)

# Do a tentative rough fix
roughFixMod <- TentativeRoughFix(boruta_output)
boruta_signif <- getSelectedAttributes(roughFixMod)

imps <- attStats(roughFixMod)
imps2 = imps[imps$decision != 'Rejected', c('meanImp', 'decision')]
head(imps2[order(-imps2$meanImp), ])  # descending sort

jpeg("~/Documents/boruta.jpg")
plot(boruta_output, cex.axis=.7, las=2, xlab="", main="Variable Importance")
dev.off
```

Based on the robuta feature selection algorithm, we will implement cutoff of 30. So we would select PAY_1, PAY_2, BILL_AMT4, BILL_AMT3, PAY_3, BILL_AMT2, PAY_4, BILL_AMT6, & PAY_6.

0.0.4 Train data resampling

We do resampling after feature selection as the documentation for SMOTE recommends doing feature selection before resampling.

[ ]:
```
data_balanced_under<- ovun.sample(DEFAULT_PAYMENT ~ ., data = trainset, method␣
 ↪= "under", seed = 1234)$data
table(data_balanced_under$DEFAULT_PAYMENT)
```

[ ]:
```
data.rose <- ROSE(DEFAULT_PAYMENT ~ ., data = trainset, seed = 1234)$data
table(data.rose$DEFAULT_PAYMENT)
```

0.0.5 Machine learning models (SVM)

[ ]:
```
library(e1071)
set.seed(1234)
```

```
svm.model <- svm(as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 +␣
 ↪BILL_AMT3 + BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 +␣
 ↪PAY_AMT1 + PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,␣
 ↪data = trainset, type="C-classification", kernel = "linear", cross = 5)
summary(svm.model)
results_test <- predict(svm.model, testset[,-24] )
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Validation Accuracy", svm.model$tot.accuracy)
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
library(e1071)
set.seed(1234)
svm.model <- svm(as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 +␣
 ↪BILL_AMT3 + BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 +␣
 ↪PAY_AMT1 + PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,␣
 ↪data = data_balanced_under, type="C-classification", kernel = "linear",␣
 ↪cross = 5)
summary(svm.model)
results_test <- predict(svm.model, testset[,-24] )
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Validation Accuracy", svm.model$tot.accuracy)
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
library(e1071)
set.seed(1234)
```

```
svm.model <- svm(as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 +␣
 ↪BILL_AMT3 + BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 +␣
 ↪PAY_AMT1 + PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,␣
 ↪data = data.rose, type="C-classification", kernel = "linear", cross = 5)
summary(svm.model)
results_test <- predict(svm.model, testset[,-24] )
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Validation Accuracy", svm.model$tot.accuracy)
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
library(e1071)
set.seed(1234)
svm.model <- svm(as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 +␣
 ↪BILL_AMT3 + BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 +␣
 ↪PAY_AMT1 + PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,␣
 ↪data = trainset, type="C-classification", kernel = "polynomial", cross = 5)
summary(svm.model)
results_test <- predict(svm.model, testset[,-24] )
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Validation Accuracy", svm.model$tot.accuracy)
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
library(e1071)
set.seed(1234)
```

```
svm.model <- svm(as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 +␣
 ↪BILL_AMT3 + BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 +␣
 ↪PAY_AMT1 + PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,␣
 ↪data = data_balanced_under, type="C-classification", kernel = "polynomial",␣
 ↪cross = 5)
summary(svm.model)
results_test <- predict(svm.model, testset[,-24] )
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Validation Accuracy", svm.model$tot.accuracy)
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
library(e1071)
svm.model <- svm(as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 +␣
 ↪BILL_AMT3 + BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 +␣
 ↪PAY_AMT1 + PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,␣
 ↪data = data.rose, type="C-classification", kernel = "polynomial", cross = 5)
summary(svm.model)
results_test <- predict(svm.model, testset[,-24] )
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Validation Accuracy", svm.model$tot.accuracy)
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
library(e1071)
set.seed(1234)
```

```
svm.model <- svm(as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 +␣
 ↪BILL_AMT3 + BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 +␣
 ↪PAY_AMT1 + PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,␣
 ↪data = trainset, type="C-classification", kernel = "radial", cross = 5)
summary(svm.model)
results_test <- predict(svm.model, testset[,-24] )
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Validation Accuracy", svm.model$tot.accuracy)
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
library(e1071)
set.seed(1234)
svm.model <- svm(as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 +␣
 ↪BILL_AMT3 + BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 +␣
 ↪PAY_AMT1 + PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,␣
 ↪data = data_balanced_under, type="C-classification", kernel = "radial",␣
 ↪cross = 5)
summary(svm.model)
results_test <- predict(svm.model, testset[,-24] )
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Validation Accuracy", svm.model$tot.accuracy)
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
library(e1071)
set.seed(1234)
```

```
svm.model <- svm(as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 +␣
 ↪BILL_AMT3 + BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 +␣
 ↪PAY_AMT1 + PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,␣
 ↪data = data.rose, type="C-classification", kernel = "radial", cross = 5)
summary(svm.model)
results_test <- predict(svm.model, testset[,-24] )
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Validation Accuracy", svm.model$tot.accuracy)
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
library(e1071)
set.seed(1234)
svm.model <- svm(as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 +␣
 ↪BILL_AMT3 + BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 +␣
 ↪PAY_AMT1 + PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,␣
 ↪data = trainset, type="C-classification", kernel = "sigmoid", cross = 5)
summary(svm.model)
results_test <- predict(svm.model, testset[,-24] )
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Validation Accuracy", svm.model$tot.accuracy)
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
library(e1071)
set.seed(1234)
```

```
svm.model <- svm(as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 +␣
 ↪BILL_AMT3 + BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 +␣
 ↪PAY_AMT1 + PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,␣
 ↪data = data_balanced_under, type="C-classification", kernel = "sigmoid",␣
 ↪cross = 5)
summary(svm.model)
results_test <- predict(svm.model, testset[,-24] )
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Validation Accuracy", svm.model$tot.accuracy)
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2])))) / 2 * 100)
```

[ ]:
```
library(e1071)
set.seed(1234)
svm.model <- svm(as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 +␣
 ↪BILL_AMT3 + BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 +␣
 ↪PAY_AMT1 + PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,␣
 ↪data = data.rose, type="C-classification", kernel = "sigmoid", cross = 5)
summary(svm.model)
results_test <- predict(svm.model, testset[,-24] )
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Validation Accuracy", svm.model$tot.accuracy)
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2])))) / 2 * 100)
```

0.0.6 Machine learning models (Logistic Regression)

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
logit <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainset,

        # `rf` method for random forest
        method='glmnet',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        family = 'binomial',
        # Accuracy to measure the performance of the model
        metric='Accuracy')
results_test <- predict(logit, testset[, -24])
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(logit$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
logit <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
```

```
        data=data_balanced_under,

        # `rf` method for random forest
        method='glmnet',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        family = 'binomial',
        # Accuracy to measure the performance of the model
        metric='Accuracy')
results_test <- predict(logit, testset[, -24])
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(logit$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
logit <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=data.rose,

        # `rf` method for random forest
        method='glmnet',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        family = 'binomial',
        # Accuracy to measure the performance of the model
        metric='Accuracy')
results_test <- predict(logit, testset[, -24])
```

```
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(logit$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

0.0.7 Machine learning models (Neural Network)

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
nn <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainset,

        # `rf` method for random forest
        method='nnet',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='Accuracy')
results_test <- predict(nn, testset[,-24])
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(nn$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
```

```
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
nn <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=data_balanced_under,

        # `rf` method for random forest
        method='nnet',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='Accuracy')
results_test <- predict(nn, testset[,-24])

cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(nn$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
nn <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,
```

```
        # Source of data; remove the Species variable
        data=data.rose,

        # `rf` method for random forest
        method='nnet',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='Accuracy')
results_test <- predict(nn, testset[,-24])
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(nn$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

0.0.8 Machine learning models (Naive Bayes)

[ ]:
```
set.seed(1234)
model = train(trainset[c(1, 6:11, 13:17, 18:22)],␣
 ↪trainset$DEFAULT_PAYMENT,'nb',trControl=trainControl(method='cv',number=5))
results_test <- predict(model$finalModel, testset[, -24])
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test$class))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(model$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
set.seed(1234)
model = train(data_balanced_under[c(1, 6:11, 13:17, 18:22)],␣
 ↪data_balanced_under$DEFAULT_PAYMENT,'nb',trControl=trainControl(method='cv',number=5))
results_test <- predict(model$finalModel, testset[, -24])
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test$class))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(model$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
set.seed(1234)
model = train(data.rose[c(1, 6:11, 13:17, 18:22)], data.
 ↪rose$DEFAULT_PAYMENT,'nb',trControl=trainControl(method='cv',number=5))
results_test <- predict(model$finalModel, testset[, -24])
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test$class))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(model$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

0.0.9 Machine learning models (Random Forest)

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
forest <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,
```

```
        # Source of data; remove the Species variable
        data=trainset,

        # `rf` method for random forest
        method='rf',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,

        # Accuracy to measure the performance of the model
        metric='Accuracy')
results_test <- predict(forest, testset[, -24])
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(forest$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
forest <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=data_balanced_under,

        # `rf` method for random forest
        method='rf',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,

        # Accuracy to measure the performance of the model
        metric='Accuracy')
```

```
results_test <- predict(forest, testset[, -24])
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(forest$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
forest <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=data.rose,

        # `rf` method for random forest
        method='rf',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,

        # Accuracy to measure the performance of the model
        metric='Accuracy')
results_test <- predict(forest, testset[, -24])
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(forest$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
```

```
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

0.0.10 Machine learning models (Decision Tree)

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
df <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainset,

        # `rf` method for random forest
        method='rpart',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,

        # Accuracy to measure the performance of the model
        metric='Accuracy')
results_test <- predict(df, testset[, -24])
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(df$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
library(rattle)
jpeg("~/Documents/df_original.jpg")
fancyRpartPlot(df$finalModel)
dev.off()
```

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
df <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=data_balanced_under,

        # `rf` method for random forest
        method='rpart',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,

        # Accuracy to measure the performance of the model
        metric='Accuracy')
results_test <- predict(df, testset[, -24])
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(df$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
library(rattle)
jpeg("~/Documents/df_us.jpg")
fancyRpartPlot(df$finalModel)
dev.off()
```

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
df <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,
```

```
        # Source of data; remove the Species variable
        data=data.rose,

        # `rf` method for random forest
        method='rpart',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,

        # Accuracy to measure the performance of the model
        metric='Accuracy')
results_test <- predict(df, testset[, -24])
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(df$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
jpeg("~/Documents/df_rose.jpg")
fancyRpartPlot(df$finalModel)
dev.off()
```

0.0.11 Machine learning models (XGBoost)

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
xgb <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainset,

        # `rf` method for random forest
        method='xgbTree',
```

```
        # Add repeated cross validation as trControl
        trControl=repeat_cv,

        # Accuracy to measure the performance of the model
        metric='Accuracy', verbose = 0)
results_test <- predict(xgb, testset[, -24])
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(xgb$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
xgb <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=data_balanced_under,

        # `rf` method for random forest
        method='xgbTree',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,

        # Accuracy to measure the performance of the model
        metric='Accuracy', verbose = 0)
results_test <- predict(xgb, testset[, -24])
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(xgb$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
```

```
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
xgb <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=data.rose,

        # `rf` method for random forest
        method='xgbTree',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,

        # Accuracy to measure the performance of the model
        metric='Accuracy', verbose = 0)
results_test <- predict(xgb, testset[, -24])
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(xgb$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

0.0.12 Machine learning models (ADABoost)

[ ]:
```
library(mltools)
library(data.table)
trainset_ada <- trainset[c(1, 6:11, 13:17, 18:22)]
dummy <- dummyVars(" ~ .", data=trainset_ada)

#perform one-hot encoding on data frame
trainset_ada <- data.frame(predict(dummy, newdata=trainset_ada))
trainset_ada$DEFAULT_PAYMENT <- trainset$DEFAULT_PAYMENT

data_balanced_under_ada <- data_balanced_under[c(1, 6:11, 13:17, 18:22)]
dummy <- dummyVars(" ~ .", data=data_balanced_under_ada)

#perform one-hot encoding on data frame
data_balanced_under_ada <- data.frame(predict(dummy,␣
 ↪newdata=data_balanced_under_ada))
data_balanced_under_ada$DEFAULT_PAYMENT <- data_balanced_under$DEFAULT_PAYMENT

data.rose_ada <- data.rose[c(1, 6:11, 13:17, 18:22)]
dummy <- dummyVars(" ~ .", data=data.rose_ada)

#perform one-hot encoding on data frame
data.rose_ada <- data.frame(predict(dummy, newdata=data.rose_ada))
data.rose_ada$DEFAULT_PAYMENT <- data.rose$DEFAULT_PAYMENT

testset_ada <- testset[c(1, 6:11, 13:17, 18:22)]
dummy <- dummyVars(" ~ .", data=testset_ada)

#perform one-hot encoding on data frame
testset_ada <- data.frame(predict(dummy, newdata=testset_ada))
testset_ada$DEFAULT_PAYMENT <- testset$DEFAULT_PAYMENT
```

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
ada <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ .,

        # Source of data; remove the Species variable
        data=trainset_ada,

        # `rf` method for random forest
        method='ada',

        # Add repeated cross validation as trControl
```

```
        trControl=repeat_cv,

        # Accuracy to measure the performance of the model
        metric='Accuracy', verbose = 0)
results_test <- predict(ada, testset_ada)
cm <- as.matrix(table(Actual = testset_ada$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(ada$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
ada <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ .,

        # Source of data; remove the Species variable
        data=data_balanced_under_ada,

        # `rf` method for random forest
        method='ada',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,

        # Accuracy to measure the performance of the model
        metric='Accuracy', verbose = 0)
results_test <- predict(ada, testset_ada)
cm <- as.matrix(table(Actual = testset_ada$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(ada$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
```

```
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
ada <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ .,

        # Source of data; remove the Species variable
        data=data.rose_ada,

        # `rf` method for random forest
        method='ada',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,

        # Accuracy to measure the performance of the model
        metric='Accuracy', verbose = 0)
results_test <- predict(ada, testset_ada)
cm <- as.matrix(table(Actual = testset_ada$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(ada$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

0.0.13 Machine learning models (LightGBM)

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
gbm <- train(

        # Formula. We are using all variables to predict Species
```

```
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainset,

        # `rf` method for random forest
        method="gbm",

        # Add repeated cross validation as trControl
        trControl=repeat_cv,

        # Accuracy to measure the performance of the model
        metric='Accuracy', verbose = 0)
results_test <- predict(gbm, testset[, -24])
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(gbm$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
gbm <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=data_balanced_under,

        # `rf` method for random forest
        method="gbm",

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
```

```
        # Accuracy to measure the performance of the model
        metric='Accuracy', verbose = 0)
results_test <- predict(gbm, testset[, -24])
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(gbm$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

[ ]:
```
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5)
gbm <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=data.rose,

        # `rf` method for random forest
        method="gbm",

        # Add repeated cross validation as trControl
        trControl=repeat_cv,

        # Accuracy to measure the performance of the model
        metric='Accuracy', verbose = 0)
results_test <- predict(gbm, testset[, -24])
cm <- as.matrix(table(Actual = testset$DEFAULT_PAYMENT, Predicted =␣
 ↪results_test))
paste("Accuracy:", sum(diag(cm)) / sum(cm) * 100)
paste("Validation Accuracy:", mean(gbm$results$Accuracy) * 100)
paste("Recall:", cm[2, 2] / (cm[2,1] + cm[2,2]) * 100)
paste("Precision:", cm[2, 2] / (cm[1, 2] + cm[2,2]) * 100)
paste("Arithmetic Avg:", ((cm[1,1] / (cm[1,1] + cm[1, 2])) + (cm[2,2] / (cm[2,␣
 ↪1] + cm[2, 2]))) * 50)
```

```
paste("Harmonic Avg:", 1/(0.5 * (((cm[1, 1] + cm[1, 2]) / cm[1, 1]) + ((cm[2,␣
 ↪1] + cm[2, 2]) / cm[2, 2])))* 100)
paste("F1-Score:", (cm[2, 2] / (cm[2, 2] + 0.5*(cm[1, 2] + cm[2, 1]))) * 100)
paste("ROC Index:", ((cm[2, 2] / (cm[2, 1] + cm[2, 2])) + (cm[1, 1] / (cm[1, 1]␣
 ↪+ cm[1, 2]))) / 2 * 100)
```

0.0.14 ROC

[ ]:
```
trainsets <- trainset
levels(trainsets$SEX) <- c("one",  "two")
levels(trainsets$EDUCATION) <- c("one",  "two",  "three",  "four")
levels(trainsets$MARRIAGE) <- c("one",  "two",  "three")
levels(trainsets$PAY_1) <- c("minustwo", "minusone", "zero",  "one",  "two", ␣
 ↪"three",  "four",  "five",  "six",  "seven",  "eight" )
levels(trainsets$PAY_2) <- c("minustwo", "minusone", "zero",  "one",  "two", ␣
 ↪"three",  "four",  "five",  "six",  "seven",  "eight" )
levels(trainsets$PAY_3) <- c("minustwo", "minusone", "zero",  "one",  "two", ␣
 ↪"three",  "four",  "five",  "six",  "seven",  "eight" )
levels(trainsets$PAY_4) <- c("minustwo", "minusone", "zero",  "one",  "two", ␣
 ↪"three",  "four",  "five",  "six",  "seven",  "eight" )
levels(trainsets$PAY_5) <- c("minustwo", "minusone", "zero",  "one",  "two", ␣
 ↪"three",  "four",  "five",  "six",  "seven",  "eight" )
levels(trainsets$PAY_6) <- c("minustwo", "minusone", "zero",  "one",  "two", ␣
 ↪"three",  "four",  "five",  "six",  "seven",  "eight" )
levels(trainsets$DEFAULT_PAYMENT) <- c("zero",  "one")
```

[ ]:
```
# SVM
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
svm <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='svmLinear',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
```

```
        metric='ROC')
# LOGIT
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
logit <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='glmnet',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        family = 'binomial',
        # Accuracy to measure the performance of the model
        metric='ROC')
# NN
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
nn <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='nnet',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')

# Naive Bayes
set.seed(1234)
```

```
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
nb <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='nb',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')

# RF
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
rf <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='rf',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')

# DT
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
dt <- train(
```

```
        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='rpart',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')

# XGBoost
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
xg <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='xgbTree',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')

# ADA
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
ada <- train(

        # Formula. We are using all variables to predict Species
```

```
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='ada',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')

# lightgbm
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
gbm <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='gbm',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')
library(MLeval)
res <- evalm(list(svm, logit, nn, nb, rf, dt, xg, ada, gbm),gnames=c('svm',␣
 ↪'logistic regression', 'neural network', 'naive bayes', 'random forest',␣
 ↪'decision tree', 'xgboost', 'adaboost', 'lightgbm'))
```

[ ]:
```
jpeg("~/Documents/roc-vanilla.jpg")
res$roc
dev.off()
```

[ ]:
```
trainsets <- data_balanced_under
levels(trainsets$SEX) <- c("one",  "two")
```

```
levels(trainsets$EDUCATION) <- c("one",  "two",  "three",  "four")
levels(trainsets$MARRIAGE) <- c("one",  "two",  "three")
levels(trainsets$PAY_1) <- c("minustwo", "minusone", "zero",  "one",  "two", ␣
 ↪"three",  "four",  "five",  "six",  "seven",  "eight" )
levels(trainsets$PAY_2) <- c("minustwo", "minusone", "zero",  "one",  "two", ␣
 ↪"three",  "four",  "five",  "six",  "seven",  "eight" )
levels(trainsets$PAY_3) <- c("minustwo", "minusone", "zero",  "one",  "two", ␣
 ↪"three",  "four",  "five",  "six",  "seven",  "eight" )
levels(trainsets$PAY_4) <- c("minustwo", "minusone", "zero",  "one",  "two", ␣
 ↪"three",  "four",  "five",  "six",  "seven",  "eight" )
levels(trainsets$PAY_5) <- c("minustwo", "minusone", "zero",  "one",  "two", ␣
 ↪"three",  "four",  "five",  "six",  "seven",  "eight" )
levels(trainsets$PAY_6) <- c("minustwo", "minusone", "zero",  "one",  "two", ␣
 ↪"three",  "four",  "five",  "six",  "seven",  "eight" )
levels(trainsets$DEFAULT_PAYMENT) <- c("zero",  "one")
```

[ ]:
```
# SVM
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
svm <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='svmRadial',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')
# LOGIT
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
logit <- train(

        # Formula. We are using all variables to predict Species
```

```
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='glmnet',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        family = 'binomial',
        # Accuracy to measure the performance of the model
        metric='ROC')
# NN
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
nn <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='nnet',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')

# Naive Bayes
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
nb <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,
```

```
        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='nb',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')

# RF
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
rf <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='rf',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')

# DT
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
dt <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,
```

```
        # `rf` method for random forest
        method='rpart',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')

# XGBoost
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
xg <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='xgbTree',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')

# ADA
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
ada <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='ada',
```

```
        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')

# lightgbm
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
gbm <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='gbm',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')
library(MLeval)
res <- evalm(list(svm, logit, nn, nb, rf, dt, xg, ada, gbm),gnames=c('svm',␣
 ↪'logistic regression', 'neural network', 'naive bayes', 'random forest',␣
 ↪'decision tree', 'xgboost', 'adaboost', 'lightgbm'))
```

[ ]:
```
jpeg("~/Documents/roc-undersampling.jpg")
res$roc
dev.off()
```

[ ]:
```
trainsets <- data.rose
levels(trainsets$SEX) <- c("one",  "two")
levels(trainsets$EDUCATION) <- c("one",  "two",  "three",  "four")
levels(trainsets$MARRIAGE) <- c("one",  "two",  "three")
levels(trainsets$PAY_1) <- c("minustwo", "minusone", "zero",  "one",  "two", ␣
 ↪"three",  "four",  "five",  "six",  "seven",  "eight" )
levels(trainsets$PAY_2) <- c("minustwo", "minusone", "zero",  "one",  "two", ␣
 ↪"three",  "four",  "five",  "six",  "seven",  "eight" )
levels(trainsets$PAY_3) <- c("minustwo", "minusone", "zero",  "one",  "two", ␣
 ↪"three",  "four",  "five",  "six",  "seven",  "eight" )
```

```
levels(trainsets$PAY_4) <- c("minustwo", "minusone", "zero",  "one",  "two", ␣
 ↪"three",  "four",  "five",  "six",  "seven",  "eight" )
levels(trainsets$PAY_5) <- c("minustwo", "minusone", "zero",  "one",  "two", ␣
 ↪"three",  "four",  "five",  "six",  "seven",  "eight" )
levels(trainsets$PAY_6) <- c("minustwo", "minusone", "zero",  "one",  "two", ␣
 ↪"three",  "four",  "five",  "six",  "seven",  "eight" )
levels(trainsets$DEFAULT_PAYMENT) <- c("zero",  "one")
```

[ ]:
```
# SVM
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
svm <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='svmRadial',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')
# LOGIT
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
logit <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='glmnet',
```

```
        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        family = 'binomial',
        # Accuracy to measure the performance of the model
        metric='ROC')
# NN
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
nn <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='nnet',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')

# Naive Bayes
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
nb <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='nb',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
```

```
        metric='ROC')

# RF
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
rf <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='rf',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')

# DT
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
dt <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='rpart',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')

# XGBoost
```

```r
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
xg <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='xgbTree',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')

# ADA
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
ada <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='ada',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')

# lightgbm
set.seed(1234)
repeat_cv <- trainControl(method='cv', number=5,␣
 ↪summaryFunction=twoClassSummary, classProbs=T, savePredictions = T)
```

```
gbm <- train(

        # Formula. We are using all variables to predict Species
        as.factor(DEFAULT_PAYMENT) ~ PAY_1 + PAY_2 + BILL_AMT4 + BILL_AMT3 +␣
 ↪BILL_AMT2 + BILL_AMT5 + PAY_6 + PAY_3 + PAY_4 + BILL_AMT6 + PAY_AMT1 +␣
 ↪PAY_AMT3 + PAY_5 + PAY_AMT5 + PAY_AMT2 + PAY_AMT4 + LIMIT_BAL,

        # Source of data; remove the Species variable
        data=trainsets,

        # `rf` method for random forest
        method='gbm',

        # Add repeated cross validation as trControl
        trControl=repeat_cv,
        # Accuracy to measure the performance of the model
        metric='ROC')
library(MLeval)
res <- evalm(list(svm, logit, nn, nb, rf, dt, xg, ada, gbm),gnames=c('svm',␣
 ↪'logistic regression', 'neural network', 'naive bayes', 'random forest',␣
 ↪'decision tree', 'xgboost', 'adaboost', 'lightgbm'))
```

[ ]:
```
jpeg("~/Documents/roc-rose.jpg")
res$roc
dev.off()
```