

Classification for Credit Card Fraud Detection

BY

RAJSHRI TOTLA

Table of Contents

INTRODUCTION	1
DESCRIPTION OF THE DATASET.....	1
Description	1
Data Source	1
Goal.....	2
Additional Details About Dataset	2
Related Research.....	3
PRE-PROCESSING.....	6
Handling missing and noisy data	6
Balancing the class distribution.....	6
R-Code.....	7
Modifying the nominal attribute values	8
ATTRIBUTE SELECTION	9
One-R	9
Correlation	10
Information Gain	11
Gain Ratio	12
Manual Selection	13
BUILDING THE MODELS	14
J48.....	14
Random Forest	14
Simple Logistic.....	14
Naïve Bayes	14
Neural Networks (Multilayer Perceptron)	15
J48 Classifier	16
Random Forest Classifier	21
Simple Logistic Classifier	25
Naive Bayes Classifier	30
Multilayer Perceptron Classifier.....	35
MODEL EVALUATION.....	39
RESULTS AND DISCUSSION	40

CONCLUSION..... 41

STEPS TO CREATE MODELS 42

 1. Pre-process..... 42

 2. Load the data in weka 42

 3. Attribute Selection 44

 4. Running Classifier..... 45

CITED REFERENCES 47

REFERENCES 47

INTRODUCTION

Our chosen topic is a classification problem. We train our model using a training dataset and then apply the model to an unknown testing set to measure the accuracy of the results. The dataset was taken from Kaggle, and we used R for exploratory data analysis to find the most relevant attributes and Weka for building and applying the model.

DESCRIPTION OF THE DATASET

Description

The datasets contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

It contains only numerical input variables, which are the result of a PCA transformation. Due to confidentiality issues, the original features and more background information about the data cannot be provided.

Features V1, V2, ..., V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 (false transaction) in case of fraud and 0 otherwise.

- No. of attributes: 31
- No. of instances: 284,807
- Multivariate dataset
- No missing values
- Attribute values are real/integer

Data Source

The dataset was taken from Kaggle. It was collected and analyzed during a research collaboration of WordLine and the Machine Learning Group of ULB (Université Libre de Bruxelles) on big data mining and fraud detection.

Link: <https://www.kaggle.com/mlg-ulb/creditcardfraud>

Goal

Credit card frauds involve using the credit/debit card to purchase commodities or withdraw money at POS locations or over the internet. There are two ways in which the fraud can be committed, either using a stolen card or using the card information such as PAN (Primary Account Number).

Employment of stringent card-use policies is counter-productive for business as the customers may choose not to use the credit cards if it becomes too much hassle. For this reason, commonly transaction authentication is required only for purchases greater than \$50, although this can vary per company and type of card. Hence, majority of the frauds are limited to low value transactions at POS locations and are done with stolen credit cards. However, in the case of using card information, the transactions are performed online where a PAN number, expiry date and CVV number which is sufficient for authentication and may involve higher value purchases. This is referred to as Card Not Present(CNP) transaction. The information can be obtained from a variety of sources such as skimmers, phishing, access to email accounts, information available to a merchant from legitimate transaction, etc.

The second approach is preferred by fraudsters for the ease of scalability and maximizing return on investment. In one kind of fraud, fake accounts are created with the information obtained using identity theft, and these accounts are used to obtain credit cards that are used for illegal ends before the companies detect the validity of the account holder/information.

The goal of our project is to build a model that can accurately predict when a fraudulent transaction takes place. Due to the heavy class imbalance, we will be measuring accuracy of the model using the Area under the Precision Recall Curve (AUPRC), or the precision score.

Additional Details About Dataset

In binary classification problems such as Fraud Detection Systems (FDS), the two classes are not equally represented in the dataset. That is, the distribution of examples is skewed since representatives of some of classes appear much more frequently. In FDS, fraudulent transactions are normally outnumbered by genuine ones. This poses a difficulty for learning algorithms, as they will be biased towards the majority group and will be unable to generalize the behavior of the minority class well. And in this case typically minority classes are the class of interest. This leads to algorithm performing poorly in terms of predictive accuracy.

Three approaches to learning from imbalanced data are discussed here:

Data level methods:

It concentrates on modifying the training set to make it suitable for a standard learning algorithm. With respect to balancing distributions we may distinguish approaches that generate new objects for minority groups (over-sampling) and that remove examples from majority groups (under-sampling). Standard approaches use random approach for selection of target samples for preprocessing. However, this often leads to removal of important samples or introduction of meaningless new objects. Therefore, more

advanced methods were proposed that try to maintain structures of groups and/or generate new data according to underlying distributions. This family of algorithms also consists of solutions for cleaning overlapping objects and removing noisy examples that may negatively affect learners.

Algorithm level methods:

It concentrates on modifying existing learners to alleviate their bias towards majority groups. This requires a good insight into the modified learning algorithm and a precise identification of reasons for its failure in mining skewed distributions. The most popular branch is cost-sensitive approaches. Here, given learner is modified to incorporate varying penalty for each of considered groups of examples. This way by assigning a higher cost to less represented set of objects we boost its importance during the learning process (which should aim at minimizing the global cost associated with mistakes). It must be noted that for many real-life problems it is difficult to set the actual values in the cost matrix and often they are not given by expert beforehand. Another algorithm-level solution is to apply one-class learning that focuses on target group, creating a data description. This way it eliminates bias towards any group, as we concentrate only on a single set of objects. One, however, needs some specialized methods to use one-class learners for more complex problems.

Hybrid methods

It concentrates on combining previously mentioned approaches to extract their strong points and reduce their weaknesses. Merging data-level solutions with classifier ensembles, resulting in robust and efficient learners is highly popular. There are some works that propose hybridization of sampling and cost-sensitive learning.

Related Research

- In Credit Card Fraud Detection: A Realistic Modeling and a Novel Learning Strategy [1], the authors talk about challenges in the field of credit card fraud detection - class imbalance, concept drift, alert feedback interaction and verification latency and how they overcame these challenges and developed their classifier model. Class imbalance refers to class distribution being highly imbalanced, as in the case of credit card frauds, where less than 1% of the transactions are fraudulent in nature. Concept drift refers to the changing shopping habits of the cardholders and the evolving strategies overtime of the fraudsters. Thus, a classifier model built in such a scenario can quickly turn obsolete. There are two approaches that were discussed in the paper to resolve this issue: updating the classifier model as soon as a change is detected (Active) or continuously updating it as and when new supervised samples are available (passive). Majority of the classifiers that are built assume that the real transaction labels (either a fraudulent or a true transaction) are obtained the very next day, which is not the case in the real-world Fraud Detection System (FDS). Fraudulent alert feedbacks are received after some days. Verification latency refers to the delay in availability of the supervised samples. The authors overcome these problems through achieving an

adaptation by combining the ensemble model and resampling techniques and using a separate classifier for the feedback and verification delay.

- In Credit card fraud detection using Machine Learning Techniques: A comparative analysis [2], the authors seek to carry out a comparative analysis of credit card fraud detection using the less exploited algorithms like the Naïve Bayes, K-nearest neighbors, and Logistic regression. They claim that basis of the fraud detection lies in the analysis of the cardholder's spending behavior and therefore in their analysis they have considered only those variables which can exhibit a unique profile of a credit card. To handle the imbalance in the data set, they used a hybrid approach of sampling where positive classes were over-sampled, and the negative classes were under-sampled. They tested their results against different performance metrics like Matthews Correlation Coefficient, sensitivity, specificity, precision, and balance collection rate over different data distributions. They concluded the paper stating that KNN gave superior performance across all the metrics.
- In Credit card fraud detection using Machine Learning Techniques: A comparative analysis[3], the authors followed a hybrid approach where they used AdaBoost and majority voting methods. In all, they used twelve algorithms in conjunction with AdaBoost and majority voting method. They used the Matthews Correlation Coefficient (MCC) to measure the quality of the classifier. They performed 10 cross validation over a real-world data set from a financial institution based in Malaysia. They claimed that AdaBoost helped in improving the performance of the FDS by achieving a perfect fraud detection rate produced by the Naïve Bayes, Decision Trees, and Random Tree. Using AdaBoost in conjunction with Linear Regression, the maximum improvement was achieved from 7.4% to 94.1%. The fraud detection rates achieved for Decision Stump and Gradient Boosting, Decision Trees and Decision Stumps, Decision Trees and Gradient Boosting, and Random Forest and Gradient Boosting perfect. The results for majority voting are better for these than the individual models. The best MCC score achieved is 0.823 which is achieved using majority voting.
- In Credit Card Fraud Detection Using Fuzzy ID3[4], the authors have used fraud detection algorithm based on Fuzzy-ID3. They have split the intermediate nodes using the attribute which gave the highest information gain and thereby classified the leaf nodes transactions as fraud, doubtful or normal. They conducted test on some transactions with different values of the attributes in different situations to determine the detection rate and by considering all the test results stated that the detection rate is 89%.
- In Credit Card Fraud Detection Using Fuzzy ID3[5], the authors have used the RUSMRN ensemble for classifying default of the datasets for the client credit cards. This method adopts three basic classifiers, namely MLP, Naïve Bayes and NB. To improve the problem of class imbalance, they have used the RUS technique for data sampling by adjusting the class distribution of the training data set. The MRN algorithm is a hybrid ensemble model based on linear and non-linear mapping and probability theory for classification problems. The MRN algorithm creates the ensemble by three base learners- multi-layer perceptron (MLP), Radial Basis function (RBF) and Naïve Bayes by applying AdaBoost. The tested the performance of the model over a Taiwanese credit card company using

measures like sensitivity, specificity and accuracy. The accuracy, specificity and sensitivity obtained as stated by the authors was 79.73%, 83.1% and 53.36% respectively.

PRE-PROCESSING

Handling missing and noisy data

The dataset had no missing values or noisy data.

No standardization methods were applied as the attribute values were all obtained through Principal Component Analysis.

Balancing the class distribution

As the dataset was highly imbalanced (positive class only accounted for 0.172% of all the transactions), we needed to balance the dataset so that the classifier models would be more accurate in their predictions.

We proceeded with a data level method of fixing the class imbalance, which involved performing both over and under-sampling of the data to create an approximate 50-50 split of the nominal attribute, and was performed using R.

We performed different sampling methods:

- **Random over sampling**

This proved ineffective and did not generate a good split of the nominal attribute.

- **Random under sampling**

Got almost the same results as over sampling. Did not generate a good split.

- **Hybrid Sampling**

Performing both over and under sampling gave us the best results and generated an approximate 50-50 split of the class attribute.

The package 'ROSE' was used to perform the sampling procedure.

ROSE (Random Over Sampling Examples) package helps us to generate artificial data based on sampling methods and smoothed bootstrap approach. This package has well defined accuracy functions to do the tasks quickly.

R-Code

```
13 library('ROSE')
14 print("Undersampling Results")
15 balanced_data <- ovun.sample(Class ~ ., data = data, method = "under", N = nrow(data), seed = 5)$data
16 table(balanced_data$Class)
17
18 print("Oversampling Results")
19 balanced_data <- ovun.sample(Class ~ ., data = data, method = "over", N = nrow(data), seed = 5)$data
20 table(balanced_data$Class)
21
22 print("Applying both sampling Results")
23 balanced_data <- ovun.sample(Class ~ ., data = data, method = "both", N = nrow(data), seed = 5)$data
24 table(balanced_data$Class)
25
26 <
```

12:1 (Top Level) ↕

Console R Markdown x

~/

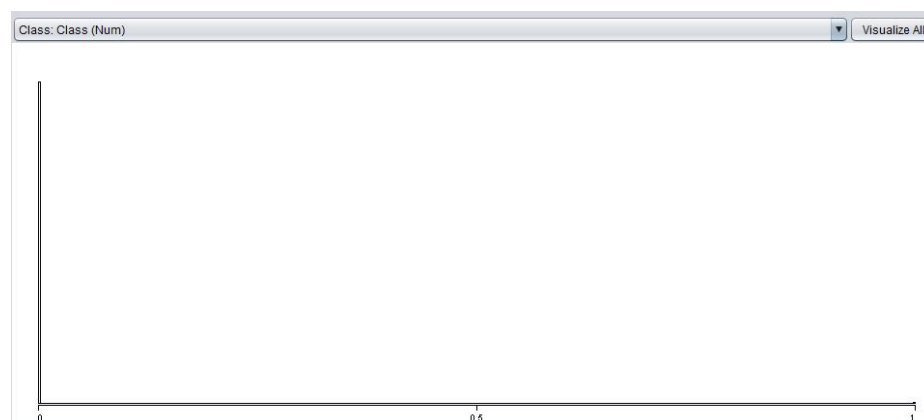
```
> library('ROSE')
> print("Undersampling Results")
[1] "Undersampling Results"
> balanced_data <- ovun.sample(Class ~ ., data = data, method = "under", N = nrow(data), seed = 5)$data
> table(balanced_data$Class)

  0      1
284315  492
>
> print("Oversampling Results")
[1] "Oversampling Results"
> balanced_data <- ovun.sample(Class ~ ., data = data, method = "over", N = nrow(data), seed = 5)$data
> table(balanced_data$Class)

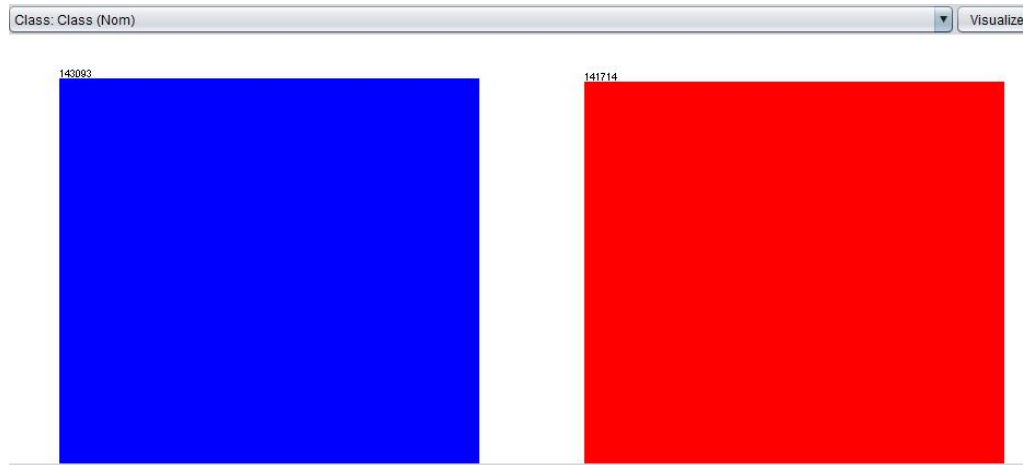
  0      1
284315  492
>
> print("Applying both sampling Results")
[1] "Applying both sampling Results"
> balanced_data <- ovun.sample(Class ~ ., data = data, method = "both", N = nrow(data), seed = 5)$data
> table(balanced_data$Class)

  0      1
143093 141714
```

Before Balancing



After Balancing



Modifying the nominal attribute values

The attribute selection algorithms could not be used for a numerical class. Therefore, using R we modified the class values to categorical values (yes or no) where 1 = yes and 0 = no.

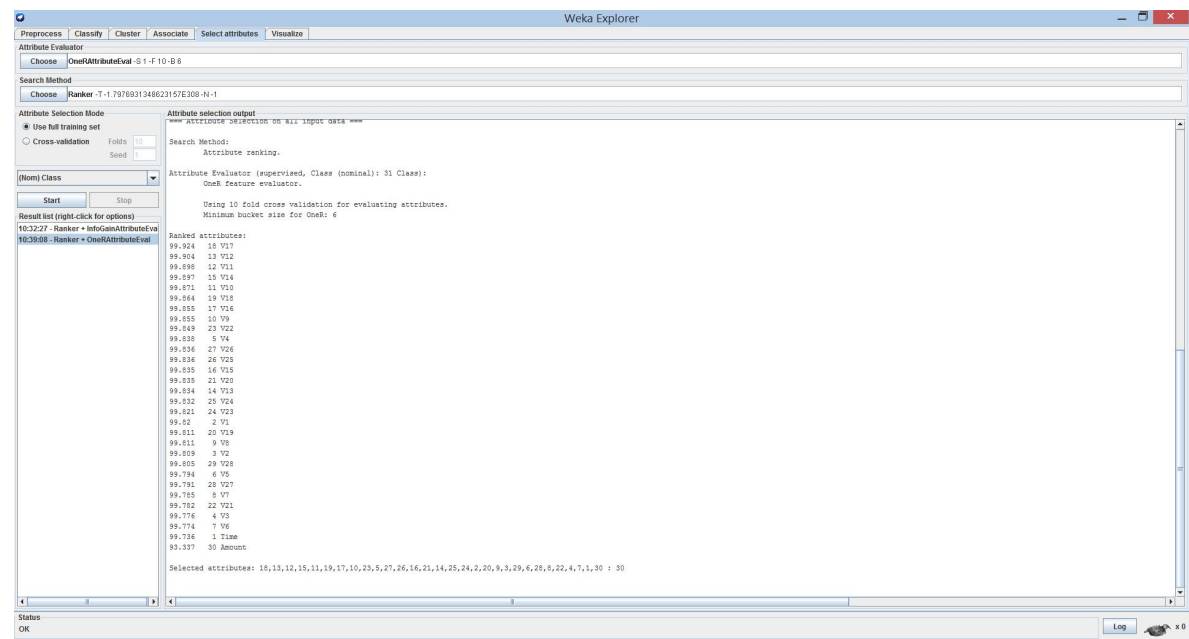
```
data <- read.csv("creditcard_balanced.csv", sep="," , header=TRUE)

data$Class[which(data$Class==1)] <- "yes"
data$Class[which(data$Class==0)] <- "no"
```

ATTRIBUTE SELECTION

The 4 attribute selection measures we used are:

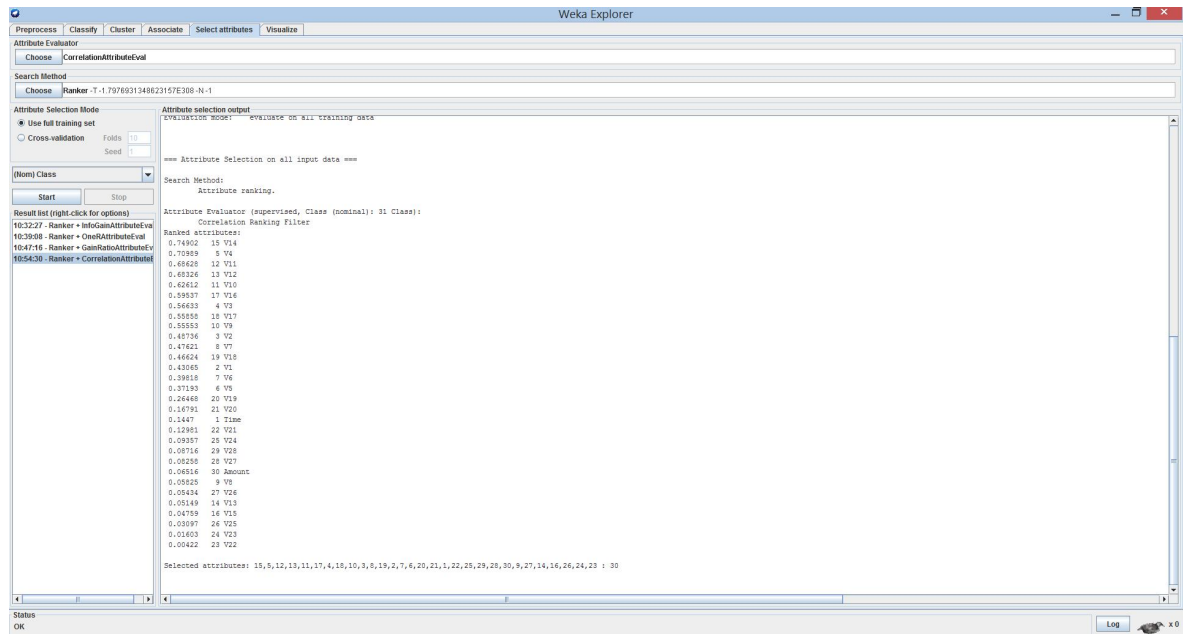
One-R



The top 6 attributes evaluated by the OneR attribute selection measure are:

Attributes		
All		
No.	Name	
1	<input checked="" type="checkbox"/>	V10
2	<input checked="" type="checkbox"/>	V11
3	<input checked="" type="checkbox"/>	V12
4	<input checked="" type="checkbox"/>	V14
5	<input checked="" type="checkbox"/>	V17
6	<input checked="" type="checkbox"/>	V18
7	<input checked="" type="checkbox"/>	Class

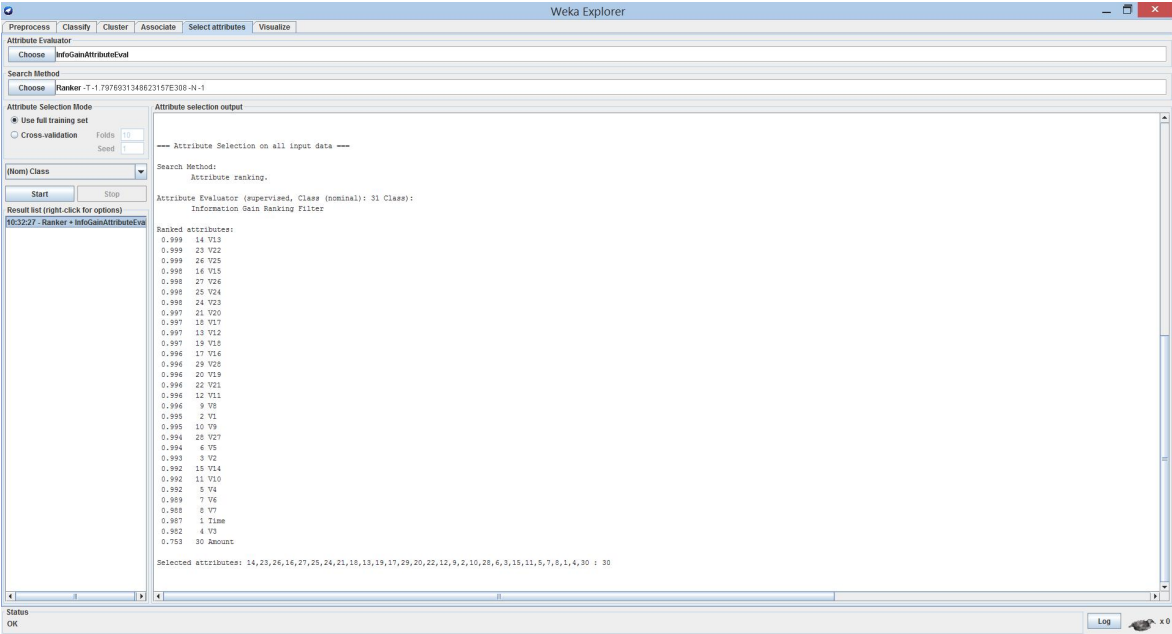
Correlation



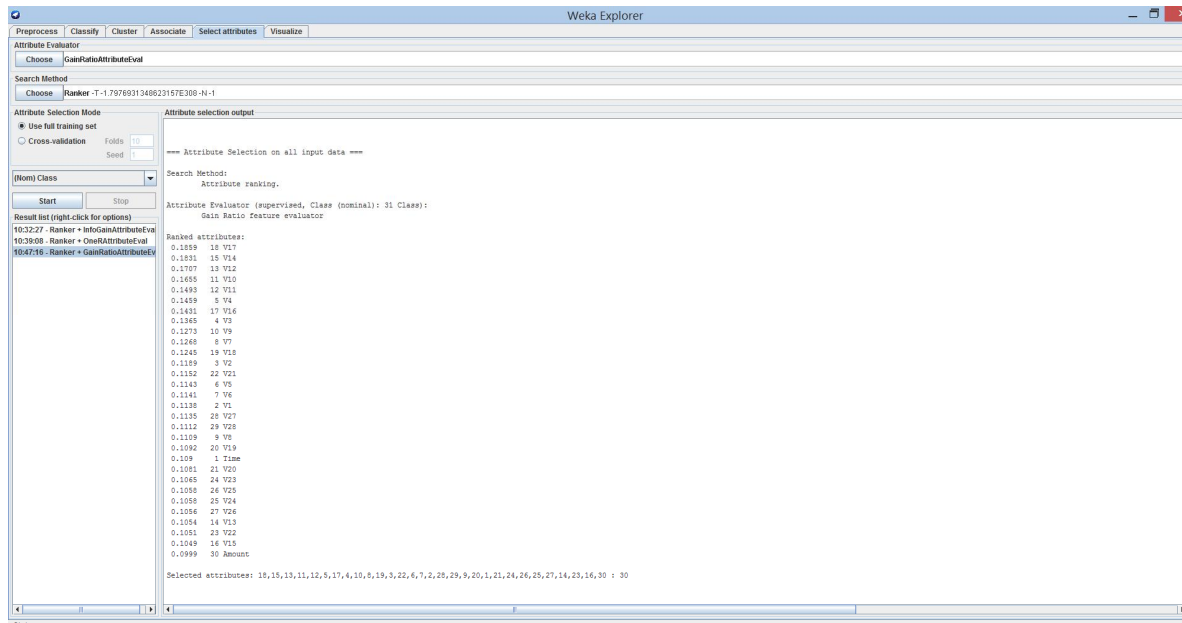
Top 6 attributes are:

Attributes	
All	
No.	Name
1	<input checked="" type="checkbox"/> V4
2	<input checked="" type="checkbox"/> V10
3	<input checked="" type="checkbox"/> V11
4	<input checked="" type="checkbox"/> V12
5	<input checked="" type="checkbox"/> V14
6	<input checked="" type="checkbox"/> V16
7	<input checked="" type="checkbox"/> Class

Information Gain



Gain Ratio

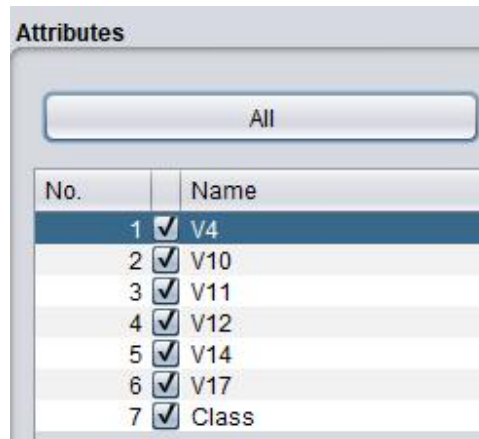


Top 6 attributes are:

No.		Name
1	<input checked="" type="checkbox"/>	V4
2	<input checked="" type="checkbox"/>	V10
3	<input checked="" type="checkbox"/>	V11
4	<input checked="" type="checkbox"/>	V12
5	<input checked="" type="checkbox"/>	V14
6	<input checked="" type="checkbox"/>	V17
7	<input checked="" type="checkbox"/>	Class

Manual Selection

Using python's scikit-learn function for feature importance, we generated a list of the importance of features and from this we picked the top 6 attributes.

A screenshot of a software window titled "Attributes". At the top, there is a button labeled "All". Below it is a table with two columns: "No." and "Name". The table contains seven rows, each with a number, a checked checkbox, and a name. The first row is highlighted in blue.

No.		Name
1	<input checked="" type="checkbox"/>	V4
2	<input checked="" type="checkbox"/>	V10
3	<input checked="" type="checkbox"/>	V11
4	<input checked="" type="checkbox"/>	V12
5	<input checked="" type="checkbox"/>	V14
6	<input checked="" type="checkbox"/>	V17
7	<input checked="" type="checkbox"/>	Class

BUILDING THE MODELS

The classification algorithms we used are:

J48

Decision Tree Algorithm is to find out the way the attributes-vector behaves for a number of instances. Also, on the bases of the training instances the classes for the newly generated instances are being found. This algorithm generates the rules for the prediction of the target variable. With the help of tree classification algorithm, the critical distribution of the data is easily understandable. J48 is an extension of ID3. The additional features of J48 are accounting for missing values, decision trees pruning, continuous attribute value ranges, derivation of rules, etc. In the WEKA data-mining tool, J48 is an open source Java implementation of the C4.5 algorithm. The WEKA tool provides a number of options associated with tree pruning. In case of potential over fitting pruning can be used as a tool for préising. In other algorithms, the classification is performed recursively till every single leaf is pure, that is the classification of the data should be as perfect as possible. This algorithm it generates the rules from which particular identity of that data is generated. The objective is progressively generalization of a decision tree until it gains equilibrium of flexibility and accuracy.

Random Forest

Random forests or randomdecisionforests are an ensemble learning method or classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of over fitting to their training set.

Random Forest is established on the fact that forests of trees splitting with oblique hyperplanes can gain accuracy as they grow without suffering from overtraining, as long as the forests are randomly restricted to be sensitive to only selected feature dimensions.

Simple Logistic

Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

Naïve Bayes

Naive Bayes is a simple, yet effective and commonly-used, machine learning classifier. It is a probabilistic classifier that makes classifications using the Maximum A Posteriori decision rule in a Bayesian setting. It

can also be represented using a very simple Bayesian network. Naive Bayes classifiers have been especially popular for text classification and are a traditional solution for classification problems.

The goal of any probabilistic classifier is, with features x_0 through x_n and classes c_0 through c_k , to determine the probability of the features occurring in each class, and to return the most likely class. Therefore, for each class, we want to be able to calculate $P(c_i | x_0, \dots, x_n)$.

Neural Networks (Multilayer Perceptron)

A multilayer perceptron (MLP) is an artificial neural network. It is composed of more than one perceptron. They are composed of an input layer to receive the signal, an output layer that makes a decision or prediction about the input, and in between those two, an arbitrary number of hidden layers that are the true computational engine of the MLP. MLPs with one hidden layer are capable of approximating any continuous function.

Multilayer perceptron's are often applied to supervised learning problems: they train on a set of input-output pairs and learn to model the correlation (or dependencies) between those inputs and outputs. Training involves adjusting the parameters, or the weights and biases, of the model in order to minimize error. Backpropagation is used to make those weight and bias adjustments relative to the error, and the error itself can be measured in a variety of ways, including by root mean squared error (RMSE).

J48 Classifier

Attribute Selection Feature: Information Gain

=== Summary ===

Correctly Classified Instances	85076	99.5716 %
Incorrectly Classified Instances	366	0.4284 %
Kappa statistic	0.9914	
Mean absolute error	0.0065	
Root mean squared error	0.0651	
Relative absolute error	1.2953 %	
Root relative squared error	13.0126 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.991	0.000	1.000	0.991	0.996	0.991	0.997	0.998	no
	1.000	0.009	0.991	1.000	0.996	0.991	0.997	0.996	yes
Weighted Avg.	0.996	0.004	0.996	0.996	0.996	0.991	0.997	0.997	

=== Confusion Matrix ===

a	b	<-- classified as
42503	366	a = no
0	42573	b = yes

Number of Leaves : 643

Size of the tree : 1285

Attribute Selection Feature: Gain Ratio

=== Summary ===

Correctly Classified Instances	85357	99.9005 %
Incorrectly Classified Instances	85	0.0995 %
Kappa statistic	0.998	
Mean absolute error	0.0017	
Root mean squared error	0.0314	
Relative absolute error	0.3349 %	
Root relative squared error	6.2815 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.998	0.000	1.000	0.998	0.999	0.998	0.999	1.000	no
	1.000	0.002	0.998	1.000	0.999	0.998	0.999	0.999	yes
Weighted Avg.	0.999	0.001	0.999	0.999	0.999	0.998	0.999	0.999	

=== Confusion Matrix ===

a	b	<-- classified as
42784	85	a = no
0	42573	b = yes

Number of Leaves : 127

Size of the tree : 253

Attribute Selection Feature: Correlation

=== Summary ===

Correctly Classified Instances	85367	99.9122 %
Incorrectly Classified Instances	75	0.0878 %
Kappa statistic	0.9982	
Mean absolute error	0.0015	
Root mean squared error	0.0295	
Relative absolute error	0.294 %	
Root relative squared error	5.8906 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.998	0.000	1.000	0.998	0.999	0.998	1.000	1.000	no
	1.000	0.002	0.998	1.000	0.999	0.998	1.000	0.999	yes
Weighted Avg.	0.999	0.001	0.999	0.999	0.999	0.998	1.000	1.000	

=== Confusion Matrix ===

a	b	<-- classified as
42794	75	a = no
0	42573	b = yes

Number of Leaves : 136

Size of the tree : 271

Attribute Selection Feature: One-R

```
Number of Leaves :    136

Size of the tree :    271

Time taken to build model: 7.42 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.06 seconds

=== Summary ===

Correctly Classified Instances      85367           99.9122 %
Incorrectly Classified Instances      75           0.0878 %
Kappa statistic                     0.9982
Mean absolute error                  0.0015
Root mean squared error              0.0295
Relative absolute error              0.294 %
Root relative squared error          5.8906 %
Total Number of Instances           85442

=== Detailed Accuracy By Class ===
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.998	0.000	1.000	0.998	0.999	0.998	1.000	1.000	no
	1.000	0.002	0.998	1.000	0.999	0.998	1.000	0.999	yes
Weighted Avg.	0.999	0.001	0.999	0.999	0.999	0.998	1.000	1.000	

```
=== Confusion Matrix ===

  a    b  <-- classified as
42794   75 |    a = no
  0 42573 |    b = yes
```

Attribute Selection Feature: Manual Selection

Time taken to build model: 8.63 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.23 seconds

=== Summary ===

Correctly Classified Instances	85357	99.9005 %
Incorrectly Classified Instances	85	0.0995 %
Kappa statistic	0.998	
Mean absolute error	0.0017	
Root mean squared error	0.0314	
Relative absolute error	0.3349 %	
Root relative squared error	6.2815 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.998	0.000	1.000	0.998	0.999	0.998	0.999	1.000	no
	1.000	0.002	0.998	1.000	0.999	0.998	0.999	0.999	yes
Weighted Avg.	0.999	0.001	0.999	0.999	0.999	0.998	0.999	0.999	

=== Confusion Matrix ===

a	b	<-- classified as
42784	85	a = no
0	42573	b = yes

Random Forest Classifier

Attribute Selection Feature: Information Gain

```
=== Classifier model (full training set) ===

RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 336.18 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 6.61 seconds

=== Summary ===

Correctly Classified Instances      85441           99.9988 %
Incorrectly Classified Instances      1           0.0012 %
Kappa statistic                      1
Mean absolute error                   0.0026
Root mean squared error               0.013
Relative absolute error               0.5127 %
Root relative squared error          2.6065 %
Total Number of Instances           85442

=== Detailed Accuracy By Class ===
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	no
	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	yes
Weighted Avg.	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	

```
=== Confusion Matrix ===

  a    b  <-- classified as
42868  1 |    a = no
  0 42573 |    b = yes
```


Attribute Selection Feature: Gain Ratio

=== Summary ===

Correctly Classified Instances	85438	99.9953 %
Incorrectly Classified Instances	4	0.0047 %
Kappa statistic	0.9999	
Mean absolute error	0.0005	
Root mean squared error	0.0084	
Relative absolute error	0.1088 %	
Root relative squared error	1.6882 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	no
	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	yes
Weighted Avg.	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	

=== Confusion Matrix ===

a	b	<-- classified as
42865	4	a = no
0	42573	b = yes

Attribute Selection Feature: Correlation

=== Summary ===

Correctly Classified Instances	85439	99.9965 %
Incorrectly Classified Instances	3	0.0035 %
Kappa statistic	0.9999	
Mean absolute error	0.0005	
Root mean squared error	0.0087	
Relative absolute error	0.1089 %	
Root relative squared error	1.7486 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	no
	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	yes
Weighted Avg.	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	

=== Confusion Matrix ===

a	b	<-- classified as
42866	3	a = no
0	42573	b = yes

Attribute Selection Feature: One-R

```
=== Classifier model (full training set) ===

RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 138.24 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 1.75 seconds

=== Summary ===

Correctly Classified Instances      85439           99.9965 %
Incorrectly Classified Instances      3           0.0035 %
Kappa statistic                    0.9999
Mean absolute error                  0.0005
Root mean squared error              0.0084
Relative absolute error              0.1095 %
Root relative squared error          1.6709 %
Total Number of Instances          85442

=== Detailed Accuracy By Class ===
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	no
	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	yes
Weighted Avg.	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	

```
=== Confusion Matrix ===

  a    b  <-- classified as
42866  3 |    a = no
  0 42573 |    b = yes
```

Attribute Selection Feature: Manual Selection

Time taken to test model on test split: 2.98 seconds

=== Summary ===

Correctly Classified Instances	85438	99.9953 %
Incorrectly Classified Instances	4	0.0047 %
Kappa statistic	0.9999	
Mean absolute error	0.0005	
Root mean squared error	0.0084	
Relative absolute error	0.1088 %	
Root relative squared error	1.6882 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	no
	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	yes
Weighted Avg.	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	

=== Confusion Matrix ===

a	b	<-- classified as	
42865	4		a = no
0	42573		b = yes

Simple Logistic Classifier

Attribute Selection Feature: Information Gain

=== Summary ===

Correctly Classified Instances	47417	55.4961 %
Incorrectly Classified Instances	38025	44.5039 %
Kappa statistic	0.1095	
Mean absolute error	0.4956	
Root mean squared error	0.4978	
Relative absolute error	99.1274 %	
Root relative squared error	99.5509 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.628	0.519	0.549	0.628	0.586	0.111	0.563	0.591	no
	0.481	0.372	0.562	0.481	0.519	0.111	0.563	0.527	yes
Weighted Avg.	0.555	0.446	0.556	0.555	0.553	0.111	0.563	0.559	

=== Confusion Matrix ===

a	b	<-- classified as
26934	15935	a = no
22090	20483	b = yes

SimpleLogistic:

Class no :

0.01 +
[V24] * 0.17

Class yes :

-0.01 +
[V24] * -0.17

Attribute Selection Feature: Gain Ratio

=== Summary ===

Correctly Classified Instances	80254	93.928 %
Incorrectly Classified Instances	5188	6.072 %
Kappa statistic	0.8785	
Mean absolute error	0.0927	
Root mean squared error	0.2134	
Relative absolute error	18.5443 %	
Root relative squared error	42.6814 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.977	0.099	0.909	0.977	0.942	0.881	0.979	0.972	no
	0.901	0.023	0.975	0.901	0.937	0.881	0.979	0.984	yes
Weighted Avg.	0.939	0.061	0.942	0.939	0.939	0.881	0.979	0.978	

=== Confusion Matrix ===

a	b	<-- classified as
41888	981	a = no
4207	38366	b = yes

SimpleLogistic:

Class no :

1.41 +
[V4] * -0.38 +
[V10] * 0.2 +
[V11] * -0.13 +
[V12] * 0.29 +
[V14] * 0.38 +
[V17] * -0

Class yes :

-1.41 +
[V4] * 0.38 +
[V10] * -0.2 +
[V11] * 0.13 +
[V12] * -0.29 +
[V14] * -0.38 +
[V17] * 0

Attribute Selection Feature: Correlation

=== Summary ===

Correctly Classified Instances	80289	93.969 %
Incorrectly Classified Instances	5153	6.031 %
Kappa statistic	0.8793	
Mean absolute error	0.1025	
Root mean squared error	0.216	
Relative absolute error	20.4999 %	
Root relative squared error	43.1969 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.982	0.103	0.906	0.982	0.942	0.882	0.979	0.971	no
	0.897	0.018	0.980	0.897	0.937	0.882	0.979	0.984	yes
Weighted Avg.	0.940	0.061	0.943	0.940	0.940	0.882	0.979	0.977	

=== Confusion Matrix ===

a	b	<-- classified as
42092	777	a = no
4376	38197	b = yes

Attribute Selection Measure: One-R

SimpleLogistic:

Class no :
1.15 +
[V10] * 0.06 +
[V11] * -0.13 +
[V12] * 0.35 +
[V14] * 0.4 +
[V18] * -0.08

Class yes :
-1.15 +
[V10] * -0.06 +
[V11] * 0.13 +
[V12] * -0.35 +
[V14] * -0.4 +
[V18] * 0.08

Time taken to build model: 36.24 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.07 seconds

=== Summary ===

Correctly Classified Instances	79648	93.2188 %
Incorrectly Classified Instances	5794	6.7812 %
Kappa statistic	0.8643	
Mean absolute error	0.1148	
Root mean squared error	0.2341	
Relative absolute error	22.9573 %	
Root relative squared error	46.8207 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.981	0.117	0.894	0.981	0.936	0.868	0.964	0.936	no
	0.883	0.019	0.979	0.883	0.928	0.868	0.964	0.975	yes
Weighted Avg.	0.932	0.068	0.936	0.932	0.932	0.868	0.964	0.956	

=== Confusion Matrix ===

```
      a      b  <-- classified as
42060  809 |      a = no
 4985 37588 |      b = yes
```

Attribute Selection Measure: Manual selection

Time taken to build model: 51.41 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.21 seconds

=== Summary ===

Correctly Classified Instances	80254	93.928 %
Incorrectly Classified Instances	5188	6.072 %
Kappa statistic	0.8785	
Mean absolute error	0.0927	
Root mean squared error	0.2134	
Relative absolute error	18.5443 %	
Root relative squared error	42.6814 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.977	0.099	0.909	0.977	0.942	0.881	0.979	0.972	no
	0.901	0.023	0.975	0.901	0.937	0.881	0.979	0.984	yes
Weighted Avg.	0.939	0.061	0.942	0.939	0.939	0.881	0.979	0.978	

=== Confusion Matrix ===

a	b	<-- classified as
41888	981	a = no
4207	38366	b = yes

Naive Bayes Classifier

Attribute Selection Feature: Information Gain

=== Summary ===

Correctly Classified Instances	54811	64.1499 %
Incorrectly Classified Instances	30631	35.8501 %
Kappa statistic	0.284	
Mean absolute error	0.4365	
Root mean squared error	0.4698	
Relative absolute error	87.3036 %	
Root relative squared error	93.9513 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.440	0.156	0.740	0.440	0.552	0.311	0.647	0.703	no
	0.844	0.560	0.600	0.844	0.701	0.311	0.648	0.589	yes
Weighted Avg.	0.641	0.357	0.670	0.641	0.626	0.311	0.647	0.647	

=== Confusion Matrix ===

a	b	<-- classified as
18867	24002	a = no
6629	35944	b = yes

Attribute Selection Feature: Gain Ratio

=== Summary ===

Correctly Classified Instances	79158	92.6453 %
Incorrectly Classified Instances	6284	7.3547 %
Kappa statistic	0.8528	
Mean absolute error	0.0733	
Root mean squared error	0.2649	
Relative absolute error	14.6663 %	
Root relative squared error	52.9765 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.982	0.129	0.884	0.982	0.931	0.858	0.971	0.963	no
	0.871	0.018	0.980	0.871	0.922	0.858	0.971	0.976	yes
Weighted Avg.	0.926	0.074	0.932	0.926	0.926	0.858	0.971	0.970	

=== Confusion Matrix ===

a	b	<-- classified as
42098	771	a = no
5513	37060	b = yes

Attribute Selection Feature: Correlation

=== Summary ===

Correctly Classified Instances	79213	92.7097 %
Incorrectly Classified Instances	6229	7.2903 %
Kappa statistic	0.8541	
Mean absolute error	0.0737	
Root mean squared error	0.2655	
Relative absolute error	14.7396 %	
Root relative squared error	53.0911 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.986	0.133	0.882	0.986	0.931	0.860	0.970	0.965	no
	0.867	0.014	0.984	0.867	0.922	0.860	0.970	0.974	yes
Weighted Avg.	0.927	0.073	0.933	0.927	0.927	0.860	0.970	0.970	

=== Confusion Matrix ===

a	b	<-- classified as
42282	587	a = no
5642	36931	b = yes

Attribute Selection Measure: One-R

Time taken to build model: 0.46 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.35 seconds

=== Summary ===

Correctly Classified Instances	79158	92.6453 %
Incorrectly Classified Instances	6284	7.3547 %
Kappa statistic	0.8528	
Mean absolute error	0.0733	
Root mean squared error	0.2649	
Relative absolute error	14.6663 %	
Root relative squared error	52.9765 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.982	0.129	0.884	0.982	0.931	0.858	0.971	0.963	no
	0.871	0.018	0.980	0.871	0.922	0.858	0.971	0.976	yes
Weighted Avg.	0.926	0.074	0.932	0.926	0.926	0.858	0.971	0.970	

=== Confusion Matrix ===

a	b	<-- classified as
42098	771	a = no
5513	37060	b = yes

Attribute Selection Measure: Manual Selection

Time taken to build model: 113.16 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.11 seconds

=== Summary ===

Correctly Classified Instances	80571	94.2991 %
Incorrectly Classified Instances	4871	5.7009 %
Kappa statistic	0.886	
Mean absolute error	0.0932	
Root mean squared error	0.2094	
Relative absolute error	18.637 %	
Root relative squared error	41.8895 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.942	0.056	0.944	0.942	0.943	0.886	0.976	0.966	no
	0.944	0.058	0.942	0.944	0.943	0.886	0.976	0.983	yes
Weighted Avg.	0.943	0.057	0.943	0.943	0.943	0.886	0.976	0.975	

=== Confusion Matrix ===

a	b	<-- classified as
40379	2490	a = no
2381	40192	b = yes

Multilayer Perceptron Classifier

Attribute Selection Feature: Information Gain

=== Summary ===

Correctly Classified Instances	50672	59.3057 %
Incorrectly Classified Instances	34770	40.6943 %
Kappa statistic	0.1847	
Mean absolute error	0.4391	
Root mean squared error	0.4925	
Relative absolute error	87.812 %	
Root relative squared error	98.5031 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.849	0.664	0.563	0.849	0.677	0.215	0.648	0.628	no
	0.336	0.151	0.688	0.336	0.451	0.215	0.648	0.670	yes
Weighted Avg.	0.593	0.409	0.625	0.593	0.564	0.215	0.648	0.649	

=== Confusion Matrix ===

a	b	<-- classified as
36388	6481	a = no
28289	14284	b = yes

Attribute Selection Feature: Gain Ratio

=== Summary ===

Correctly Classified Instances	81425	95.2986 %
Incorrectly Classified Instances	4017	4.7014 %
Kappa statistic	0.906	
Mean absolute error	0.0705	
Root mean squared error	0.1997	
Relative absolute error	14.1052 %	
Root relative squared error	39.9335 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.969	0.063	0.939	0.969	0.954	0.906	0.979	0.967	no
	0.937	0.031	0.967	0.937	0.952	0.906	0.979	0.984	yes
Weighted Avg.	0.953	0.047	0.953	0.953	0.953	0.906	0.979	0.975	

=== Confusion Matrix ===

a	b	<-- classified as
41528	1341	a = no
2676	39897	b = yes

Attribute Selection Feature: Correlation

=== Evaluation on test split ===

Time taken to test model on test split: 0.32 seconds

=== Summary ===

Correctly Classified Instances	81336	95.1944 %
Incorrectly Classified Instances	4106	4.8056 %
Kappa statistic	0.9039	
Mean absolute error	0.0769	
Root mean squared error	0.2001	
Relative absolute error	15.382 %	
Root relative squared error	40.0152 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.976	0.072	0.931	0.976	0.953	0.905	0.981	0.976	no
	0.928	0.024	0.975	0.928	0.951	0.905	0.981	0.986	yes
Weighted Avg.	0.952	0.048	0.953	0.952	0.952	0.905	0.981	0.981	

=== Confusion Matrix ===

a	b	<-- classified as
41849	1020	a = no
3086	39487	b = yes

Attribute Selection Measure: One-R

Time taken to build model: 113.16 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.11 seconds

=== Summary ===

Correctly Classified Instances	80571	94.2991 %
Incorrectly Classified Instances	4871	5.7009 %
Kappa statistic	0.886	
Mean absolute error	0.0932	
Root mean squared error	0.2094	
Relative absolute error	18.637 %	
Root relative squared error	41.8895 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.942	0.056	0.944	0.942	0.943	0.886	0.976	0.966	no
	0.944	0.058	0.942	0.944	0.943	0.886	0.976	0.983	yes
Weighted Avg.	0.943	0.057	0.943	0.943	0.943	0.886	0.976	0.975	

=== Confusion Matrix ===

a	b	<-- classified as
40379	2490	a = no
2381	40192	b = yes

Attribute Selection Measure: Manual Selection

Time taken to build model: 193.86 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.12 seconds

=== Summary ===

Correctly Classified Instances	81425	95.2986 %
Incorrectly Classified Instances	4017	4.7014 %
Kappa statistic	0.906	
Mean absolute error	0.0705	
Root mean squared error	0.1997	
Relative absolute error	14.1052 %	
Root relative squared error	39.9335 %	
Total Number of Instances	85442	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.969	0.063	0.939	0.969	0.954	0.906	0.979	0.967	no
	0.937	0.031	0.967	0.937	0.952	0.906	0.979	0.984	yes
Weighted Avg.	0.953	0.047	0.953	0.953	0.953	0.906	0.979	0.975	

=== Confusion Matrix ===

a	b	<-- classified as
41528	1341	a = no
2676	39897	b = yes

MODEL EVALUATION

Due to the high class imbalance, we measure the performance of the models based on the FP rate and ROC curve area along with the percentage of correctly classified instances (accuracy).

Classifier Algorithm	Attribute Selection Algorithm	Accuracy	FP Rate	ROC Curve Area
Random Forest	Information Gain	99.9988 %	0.000	1.000
Random Forest	Gain Ratio	99.9953 %	0.000	1.000
Random Forest	Correlation	99.9965 %	0.000	1.000
Random Forest	One R	99.965 %	0.000	1.000
Random Forest	Manual Selection	99.9943 %	0.000	1.000
J48	Information Gain	99.5716 %	0.996	0.997
J48	Gain Ratio	99.9005 %	0.001	0.999
J48	Correlation	99.9122 %	0.001	1.000
J48	One R	99.9181 %	0.001	0.999
J48	Manual Selection	99.9005 %	0.001	0.999
MultilayerPerceptron	Information Gain	64.1499 %	0.357	0.647
MultilayerPerceptron	Gain Ratio	95.2986 %	0.047	0.979
MultilayerPerceptron	Correlation	95.1944 %	0.048	0.981
MultilayerPerceptron	One R	94.2991 %	0.057	0.976
MultilayerPerceptron	Manual Selection	95.2986 %	0.047	0.979
Simple Logistic	Information Gain	55.4961 %	0.446	0.563
Simple Logistic	Gain Ratio	93.928 %	0.061	0.979
Simple Logistic	Correlation	93.969 %	0.061	0.979
Simple Logistic	One R	93.2188 %	0.068	0.964
Simple Logistic	Manual Selection	93.9280 %	0.061	0.979
Naive Bayes Classifier	Information Gain	59.3057 %	0.409	0.648
Naive Bayes Classifier	Gain Ratio	92.6453 %	0.074	0.971
Naive Bayes Classifier	Correlation	92.7097 %	0.073	0.970
Naive Bayes Classifier	One R	92.0133 %	0.080	0.958
Naive Bayes Classifier	Manual Selection	92.6453 %	0.074	0.971

RESULTS AND DISCUSSION

From the evaluation table, we observe that Random Forest Classifier is outperforming among all other classifiers. It has almost 100% accuracy, least average FP Rate and highest area under ROC curve. J48 is performing similar to Random Forest but we prefer random forest because Random Forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

Therefore, in Random Forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. You can even make trees more random, by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

The accuracy of the Neural Network could be improved using Bagging or Boosting algorithms, and also through manual modification of the perceptron parameters.

CONCLUSION

Observing the results across all the models that were built, we can conclude that best model to use to detect fraudulent credit card transactions is the Random Forest classifier, which gives ROC curve areas of 1 for all the different attribute selection measures.

Interestingly, even in real world banking or finance applications use the Random Forest classifier for learning and predicting consumer behaviour.

STEPS TO CREATE MODELS

1. Pre-process

Execute the following R code for pre-processing data. Set working directory as where the data is saved.

Also install package "ROSE" to perform sampling.

RCODE

```
setwd("")

data <- read.csv("creditcard_balanced.csv", sep=",", header=TRUE)

library('ROSE')

balanced_data <- ovun.sample(Class ~ ., data = data, method = "both", N = nrow(data), seed = 5)$data

balanced_data$Class[which(balanced_data$Class==1)] <- "yes"

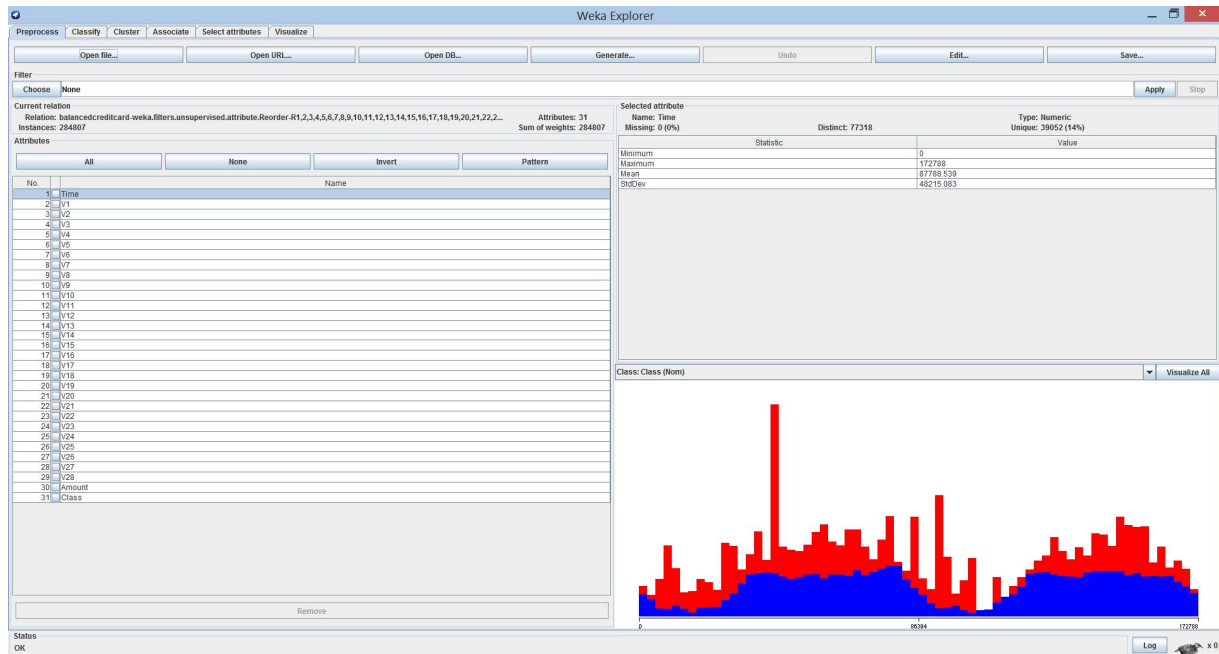
balanced_data$Class[which(balanced_data$Class==0)] <- "no"

table(balanced_data$Class)

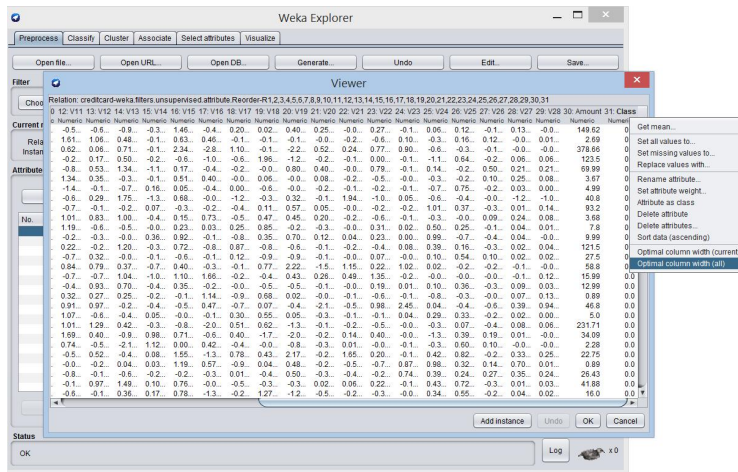
write.csv(balanced_data,"balancedcreditcard.csv")
```

2. Load the data in weka

1. Open Weka -> Explorer
2. Open file -> balanced_data.csv (*the preprocessed file*)
3. Click Save file ("Save as ARFF")
4. Open file -> balanced_data.arff

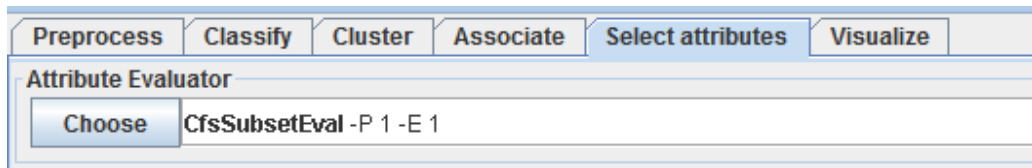


5. Make sure there are only attributes as shown in above screenshot. If there are some extra attributes due to pre-processing, select those extra attribute and click "remove".
6. From "edit" set class column as "class" attribute



3. Attribute Selection

1. Click on "Select Attributes"



2. In the window choose the "Attribute Evaluator" as following for different attribute selection algorithms.

Attribute/ Feature Selection	Name in Weka
Correlation	CorrelationAttributeEval
Gain Ratio	GainRatioAttributeEval
Information Gain	InfoGainAttributeEval
One R	OneRAttributeEval

3. We select top 6 features generated from each feature selection algorithm.
4. For the manual selection of attribute we calculated feature importance.

Therefore we will have following attribute selection as result:

Information Gain Ranking Filter

V13 V22 V25 V15 V26 V24 + Class

OneR feature evaluator.

V17 V12 V11 V14 V10 V18+ Class

Gain Ratio feature evaluator

V17 V14 V12 V10 V11 V4+ Class

Correlation Ranking Filter

V14 V4 V11 V12 V10 V16+ Class

Manual Selection

V4 V10 V11 V12 V14 V17+ Class

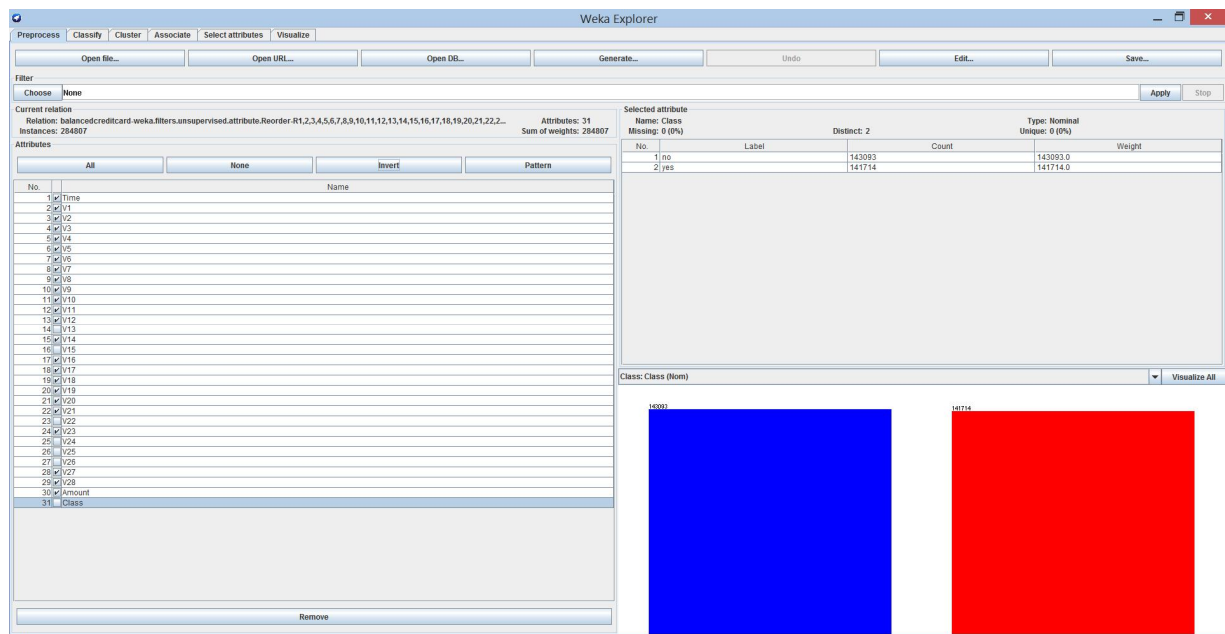
4. Running Classifier

1. For each classifier we will load the data and keep only 6 best attribute we got from previous steps. Therefore for each classifier we will have to
 - i. Load the data
 - ii. Select attributes
 - iii. And then run the classifier
 - iv. Repeat with attribute selection algorithm

TIP: The approach could be reversed i.e. first choose attribute according to algorithm and run all classifiers on that set and record results. Repeat for other attribute selection algorithm. This could save work of selecting attributes again and again.

2. Attribute Selected will be **6 best attribute + Class attribute**, to that load the data select the remaining attributes and **"Remove"** them.

Following is an example for Information Gain Ranking Filter:



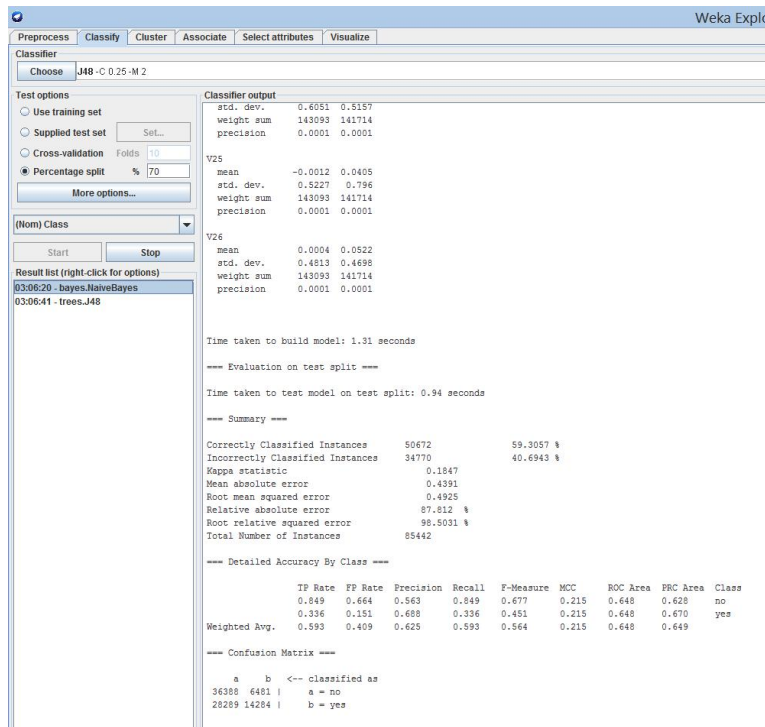
3. After attribute selection, choose **"Classify"** to create classifier models.
For different algorithms you need to follow the following:

- **Multilayer Perceptron (Neural Networks)**
Choose->functions->MultilayerPerceptron
- **Naïve Bayes**
Choose -> bayes -> naiveBayes

- **J48**
Choose -> trees ->J48
- **Random Forest**
Choose -> trees ->.RandomForest
- **Simple Logistic**
Choose -> functions->SimpleLogistic

4. Set **Percentage Split** as **70%** to divide the training and test data into 7:3 ratio.
5. Click **Start** to run classifier.
6. Results could be viewed in Classifier Output window once classifier finished running.

Here is a sample screenshot:



7. Repeat procedure for each classifier.

CITED REFERENCES

- [1]`Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, Gianluca Bontempi. "Credit Card Fraud Detection: A Realistic Modeling and a Novel Learning Strategy", IEEE transactions on neural networks and learning systems (Volume: PP, Issue: 99).`
- [2] `John O. Awoyemi, Adebayo O. Adetunmbi, Samuel A. Oluwadare. "Credit card fraud detection using Machine Learning Techniques: A comparative analysis", Computing Networking and Informatics (ICCNi), 2017 International Conference.`
- [3] `Kuldeep Randhawa, Chu Kiong Loo, Manjeevan Seera, Chee Peng Lim4, Asoke K. Nandi. "Credit card fraud detection using AdaBoost and majority voting", IEEE Access (Volume: PP, Issue: 99)`
- [4] `S Md. S Askari, Md. Anwar Hussain. "Credit Card Fraud Detection Using Fuzzy ID3" Computing, Communication and Automation (ICCCA), 2017 International Conference.`
- [5] `Anusorn Charleonnann. "Credit Card Fraud Detection Using RUS and MRN Algorithms", Management and Innovation Technology International Conference (MITicon), 2016`

REFERENCES

- Random Forest Classifier <https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd>
- Naïve Bayes <https://towardsdatascience.com/introduction-to-naive-bayes-classification-4cffabb1ae54>
- WekaManual http://statweb.stanford.edu/~lpekelis/13_datafest_cart/WekaManual-3-7-8.pdf
- ROSE R Package <https://cran.r-project.org/web/packages/ROSE/ROSE.pdf>