**Drexel University**

3141 Chestnut Street

Philadelphia, PA 19104

Loan Credit Risk Analysis

**March, 2021**

Team Members:

Olawin, Ridwan

Wu, William

Ta, Quan

Lee, Hajun

# 

# 

# 

# 

# 

# 

# 

# Introduction

Credit cards are widely used in our time. Nowadays, people look at your credit to lend you money. Banks, companies, and lenders look at your credit information in order to assess the amount of money that they can lend. However, even if one’s credit looks fine at the beginning, one can become a credit card defaulter in a minute, making the lenders lose their profit. Our application seeks to find out what interesting rules lie in people who borrow money from others and become credit card defaulters. This application seeks to allow lenders to have an almost accurate evaluation of the person who is borrowing money and assume whether lending to such individuals would be good or not. This application would be extremely useful especially if you are a prospective lender.

# Data Descriptions

**Data Collection**

One important note is that the current data set has 122 variables. The variables that we have below are part of an abridged version of the data set that was found on Kaggle(<https://www.kaggle.com/mishra5001/credit-card?select=application_data.csv>).

**Descriptive Statistics**

It should be noted that all of these statistics are before pre-processing. Furthermore, our dataset is currently a mix of nominal data as well as qualitative data. This results in the nominal variables having an empty row.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Variable** | **Observations** | **Mean** | **Std. Dev** | **Min** | **Max** |
| SK\_ID\_PREV | 1,670,214.00 | 1,923,089.00 | 532,598.00 | 1,000,001.00 | 2,845,382.00 |
| SK\_ID\_CURR | 1,670,214.00 | 278,357.20 | 102,814.80 | 100,001.00 | 456,255.00 |
| NAME\_CONTRACT\_TYPE | 0.00 | - | - | - | - |
| AMT\_ANNUITY | 1,297,979.00 | 15,955.12 | 12,782.14 | 0.00 | 418,058.20 |
| AMT\_APPLICATION | 1,670,214.00 | 175,233.90 | 292,779.80 | 0.00 | 6,905,160.00 |
| AMT\_CREDIT | 1,670,213.00 | 196,114.00 | 318,574.60 | 0.00 | 6,905,160.00 |
| AMT\_DOWN\_PAYMENT | 774,370.00 | 6,697.40 | 20,921.50 | -0.90 | 3,060,045.00 |
| AMT\_GOODS\_PRICE | 1,284,699.00 | 227,847.30 | 315,396.60 | 0.00 | 6,905,160.00 |
| WEEKDAY\_APPR\_PROCESS\_START | 0.00 | - | - | - | - |
| HOUR\_APPR\_PROCESS\_START | 1,670,214.00 | 12.48 | 3.33 | 0.00 | ` |
| FLAG\_LAST\_APPL\_PER\_CONTRACT | 0.00 | - | - | - | - |
| NFLAG\_LAST\_APPL\_IN\_DAY | 1,670,214.00 | 1.00 | 0.06 | 0.00 | 1.00 |
| RATE\_DOWN\_PAYMENT | 774,370.00 | 0.08 | 0.11 | 0.00 | 1.00 |
| RATE\_INTEREST\_PRIMARY | 5,951.00 | 0.19 | 0.09 | 0.03 | 1.00 |
| RATE\_INTEREST\_PRIVILEGED | 5,951.00 | 0.77 | 0.10 | 0.37 | 1.00 |
| NAME\_CASH\_LOAN\_PURPOSE | 0.00 | - | - | - | - |
| NAME\_CONTRACT\_STATUS | 0.00 | - | - | - | - |
| DAYS\_DECISION | 1,670,214.00 | -880.68 | 779.10 | -2,922.00 | -1.00 |
| NAME\_PAYMENT\_TYPE | 0.00 | - | - | - | - |
| CODE\_REJECT\_REASON | 0.00 | - | - | - | - |
| NAME\_TYPE\_SUITE | 0.00 | - | - | - | - |
| NAME\_CLIENT\_TYPE | 0.00 | - | - | - | - |
| NAME\_GOODS\_CATEGORY | 0.00 | - | - | - | - |
| NAME\_PORTFOLIO | 0.00 | - | - | - | - |
| NAME\_PRODUCT\_TYPE | 0.00 | - | - | - | - |

**Variable characteristics**

This can be found in Reference number one link.

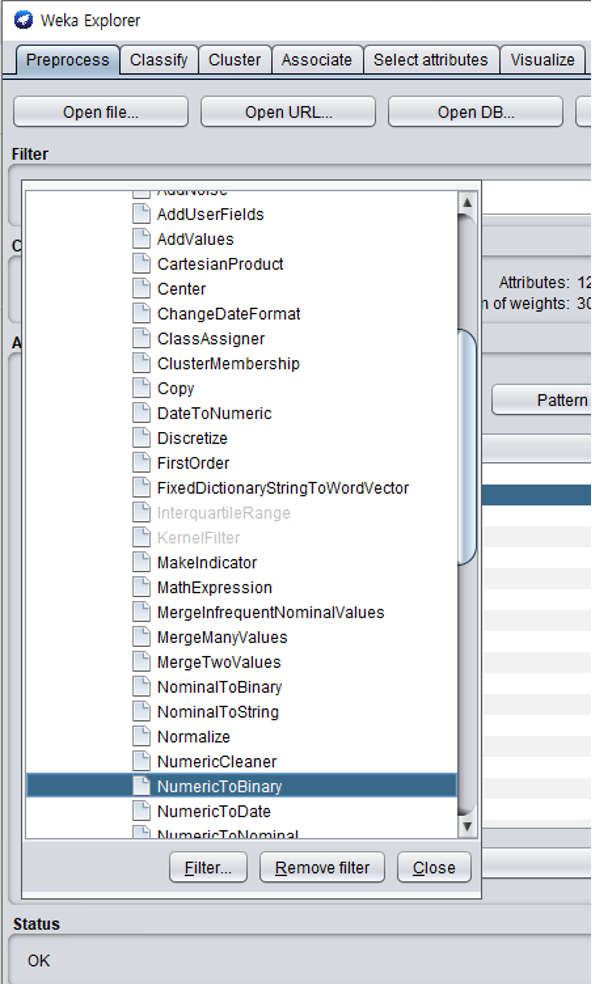
**Pre-processing**

Our data is in .csv format. Weka has a file converter feature that allows users to change .csv files into .arff files that could be used in Weka.

**Step 1 - Discretize Data:**

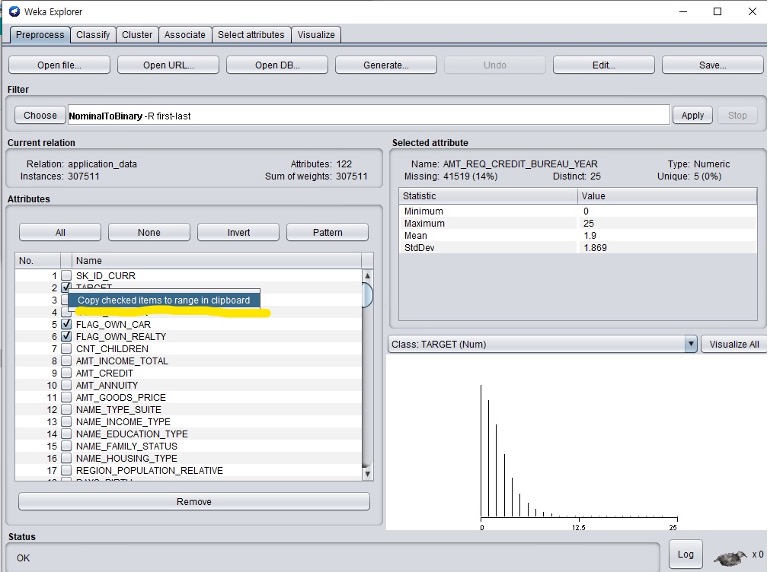
We realize that Binary data(Yes/No or 0/1) Data types are encoded as Numeric in Weka. We will have to transform these data into Binary data. Weka has filter features that allow users to transform data.

To start with, Select “Choose” from “Filter” (figure 1) below and then select **Weka>Filters>Unsupervised>Attribute>NumericToBinary**:

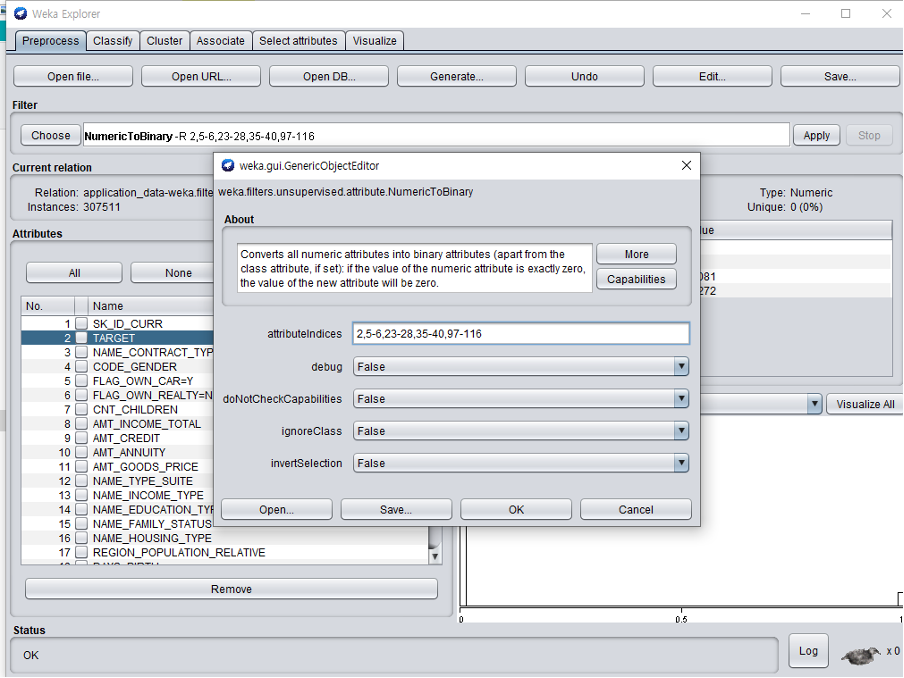
****

**FIGURE 1**

Weka allows the user to manually select attributes and save the indices to the clipboard (Figure 2) so that we could easily insert the indices to apply NumericToBinary Filter (Figure 3).

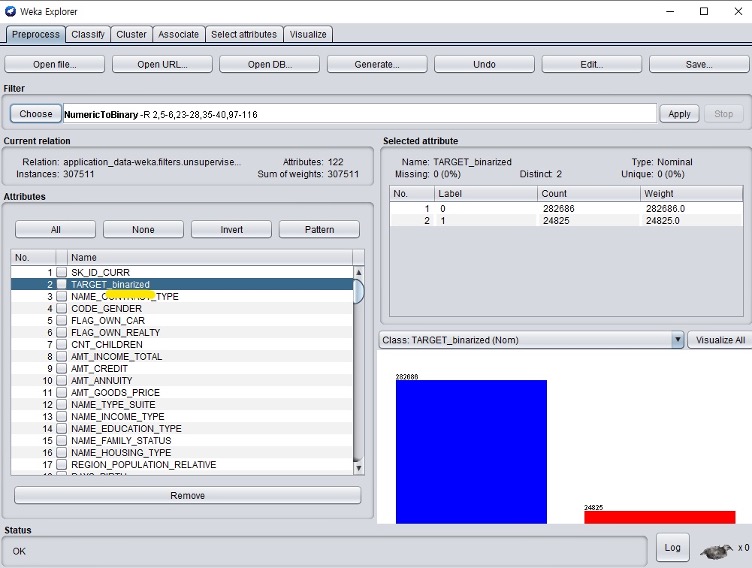


**FIGURE 2**



**FIGURE 3**

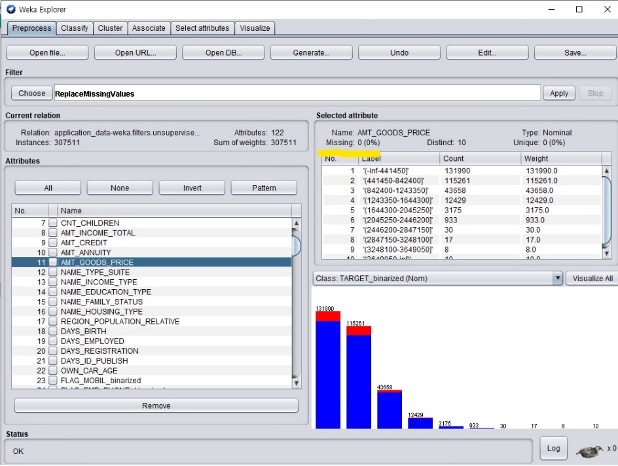
After running “Apply”, we see that the selected attributes have been “binarized” (Figure 4):



**FIGURE 4**

**Step 2 - Dealing with missing values:**

Weka has a feature to replace the missing values found in attributes using Filter>Unsupervised>Attribute>ReplaceMissingValues. We will use this filter to replace the missing values found in Numeric data (take the example of AMT\_GOODS\_PRICE which has 278 missing values - Figure 5):

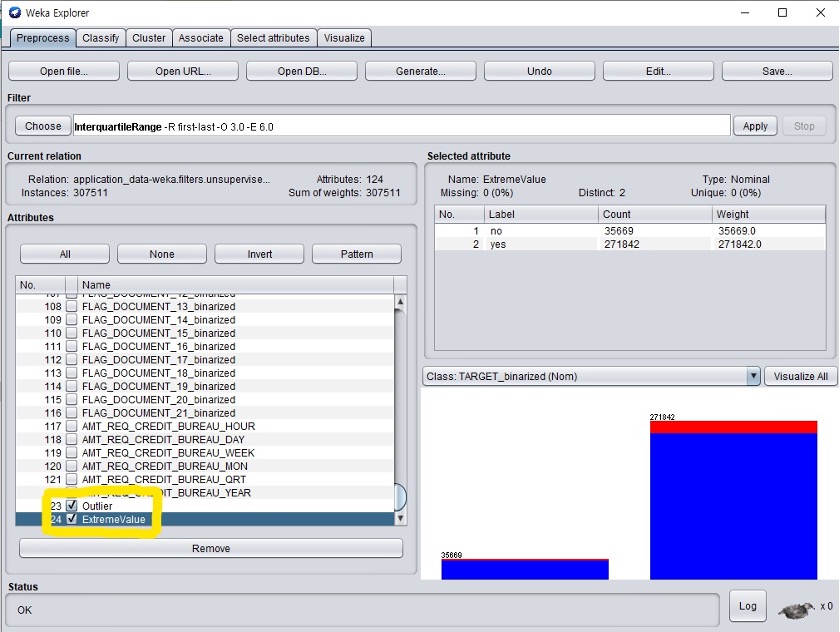


**FIGURE 5**

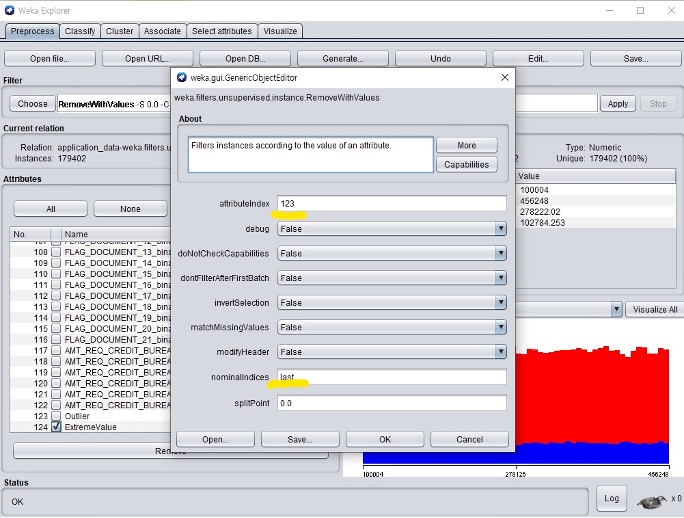
It should also be noted that nominal data such as OCCUPATION\_TYPE attributes have also been replaced with its mode accordingly.

**Step 3 - Removing outliers and extreme values:**

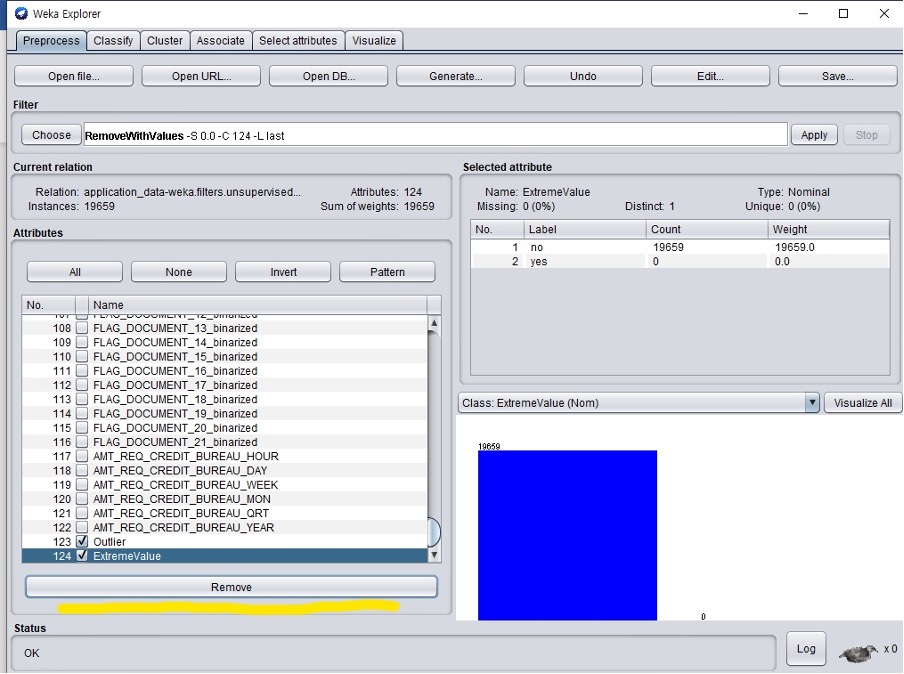
First, we must filter our attributes so that the outlier detection feature only deals with numeric values. This could be done by selecting “Filter” below the “Choose” button. Then only check on “Numeric attributes” and “Numeric Class”. Next, choose “Interquartile” Filter which allows us to determine the outliers and extreme values. This allows us to see newly created attributes: “Outlier” and “ExtremeValue” (Figure 6). We will remove the outliers and extreme values using “RemoveWithValues” Filter found in an instance of unsupervised. We insert the indices of the “Outlier” and “ExtremeValue” attributes and set the nominalIndices as “last” since the two attributes have “yes” index (meaning instances of outlier or extreme value) in the last index. We remove those instances: (An example of index 123 i.e Figure 7). We can remove those attributes by selecting the attributes and press “Remove” (Figure 8).



**FIGURE 6**

****

**FIGURE 7**

****

**FIGURE 8**

**Step 4 - Discretize:**

Next, we deal with Numeric data that needs to be transformed to nominal data by using Discretize. A typical attribute that needs to be discretized is the one that involves the count number of family members or children. Below is an example of how to discretize “CNT\_CHILDREN” attribute:

* Choose “Discretize” from Weka>Filters>Unsupervised>Attribute>Discretize
* Then insert the index of “AMT\_INCOME\_TOTAL” (index 7), set “binRangePrecision” to 0 so that we don’t get decimal points for the bin range, and select 7 bins. Click “Apply”, our data is now discretized

We apply this to each attribute(AMT\_CREDIT, etc…) that requires discretization with corresponding “binRangePrecision” that suits each attribute.

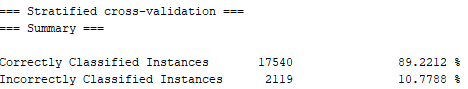
# Methods / Analysis and Results

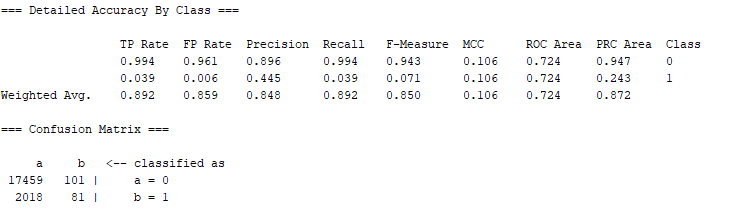
Given the nature of the data set, the technique intended to be used in solving this problem is classification-based learning. The reason is that the given data set contains frequent patterns that have been established with a lender. Hence, it would be possible to establish the correlation between classifying them whether they are low or high risk. It should be noted that the target label chosen is the TARGET attribute as this contains the necessary binary classification to reference.

**Algorithm 1 - Logistic Regression - Weka (functions.logistic)**

This algorithm models the probability of some event occurring as a linear function of a set of predictor variables. Given the nature of our data set containing almost 126 predictor variables, we decided that this algorithm was efficient enough to model our data. It should be noted that a cross-validation fold was used to measure our accuracy.

# Analysis and Results

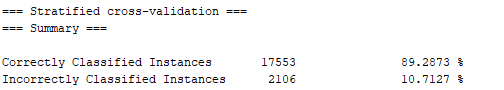
****

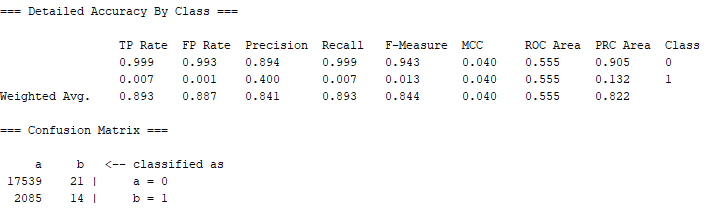
****

Logistic regression resulted in similar results to the majority of the other tests conducted. All metrics including correctly classified instances (CCI) were accurate to the same levels as the decision tree and KNN tests done below. However, something rather different is that the ROC Area for the logistic regression is the highest of all the tests.

**Algorithm 2 -Decision tree - Weka (trees. J48)**

The general approach to a decision tree is that it is constructed in a top-down recursive divide-and-conquer manner. We decided to use a decision tree since decision trees work best with nominal data which the majority of our dataset consists of.





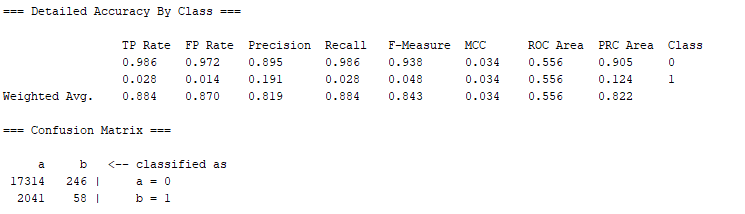
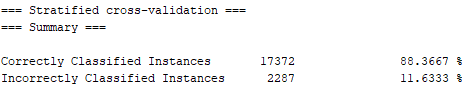
It is worth noting that the accuracy of this dataset is rather high with a nearly 100% True Positive Rate for evaluating low-risk transactions. Furthermore, the accuracy for evaluating precision, recall, and F-Measure for Class 0 is also very high. However, the accuracy for Class 1 for high-risk transactions is much lower and this in part can be attributed to the low sample size.

**Algorithm 3 - KNN - Weka (lazy.IBK)**

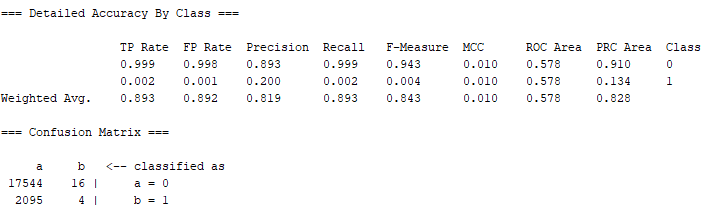
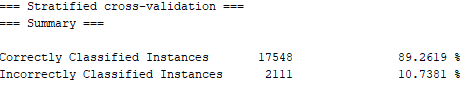
KNN stores all available instances and then classifies any new instances based on a similarity measure. In the nearest-neighbor classification method, each new instance is compared with existing ones by using a distance metric and the closest existing instance is used to assign the class to the new one.

In high-risk fraudulent credit transactions, KNN allows us to classify an incoming transaction by calculating a nearest point to the new incoming transaction. Then if the nearest neighbor is fraudulent, we classify the transaction as fraudulent. Since we are working with a large dataset, we decided to use a large K value to help reduce the effect of a noisy dataset.

**K=5**



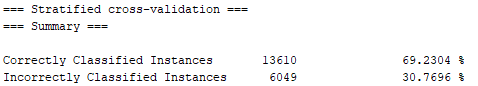
**K=10**

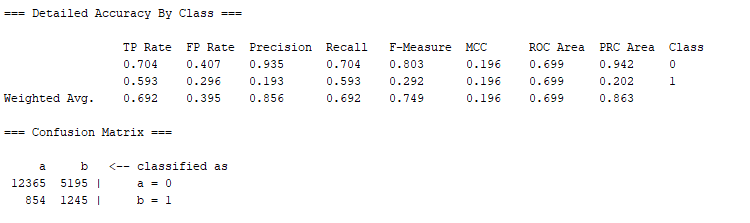


Testing for when K=10 seems to be most accurate, for tests that go beyond 10 such as when K= 20, 100, and 1000 the results start to make much less sense. Almost all measures besides for ROC Area share similarities with the metrics in the decision tree results.

**Algorithm 4 - Naive Bayes - Weka (bayes.NaiveBayes)**

Naïve Bayes is a supervised machine learning method that uses a training dataset with known target classes to make predictions of future instances. This algorithm works best with an almost large training dataset and lets us solve binary classification problems.



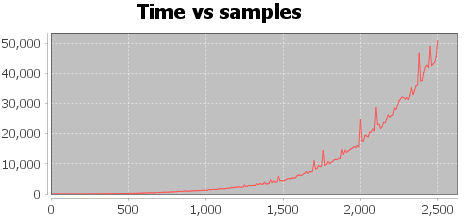


Naive Bayes resulted in the most insignificant and inaccurate results of all the tests conducted. For instance, the CCI drops to 69.23% compared to the 89% found in most other tests. The accuracy for evaluating precision, recall, and F-Measure for Class 0 and Class 1 are not consistent with other tests. The recall and F-Measure for Class 1 in Naive Bayes are significantly higher than those in other tests

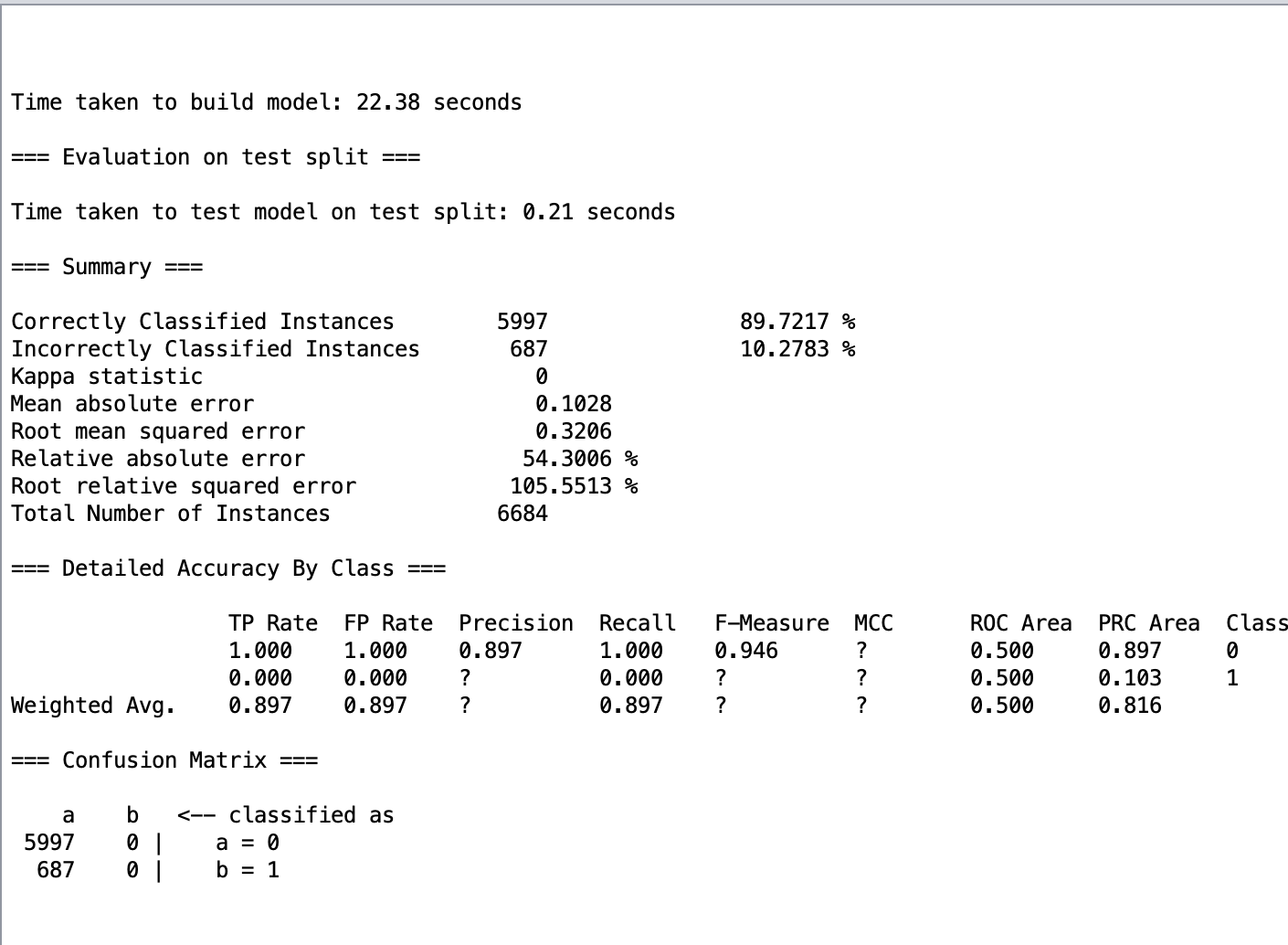
**Algorithm 5 - Support Vector Machines (SVM) - Weka (functions.SMO)**

Support Vector Machine is a technique that is especially suitable for binary classification techniques. In our case, we have legitimate (low-risk) and fraudulent (high-risk) classes. SVM methods require large training dataset sizes to achieve maximum prediction accuracy.

Training time complexity of non-linear SVMs is between O(n2) and O(n3). Weka SMO seems to be closer to the latter. For instance, the execution time in ms for a random set with 205 features with Weka SMO looks like this:



Given the nature of the graph being hard to predict, but it could take anywhere from 1 day (n2) and 124 days (n3). Hence, we chose an alternative algorithm using linear kernel implementation: LIBLINEAR.



The measure of accuracy used here is a percentage split of 66%. We can see that the number of correctly classified instances is almost 89% showing that it is on par with the other classifying algorithms. The use of this algorithm was also considerably faster than the SVM. We noted that running the SVM made our machines very slow hence the libLinear alternative.

# 

# Conclusions

One of the common issues found in datasets that are used for classification is imbalanced classes issue. Imbalanced data typically refers to a classification problem where the number of observations per class is not equally distributed. It usually reflects an unequal distribution of classes within a dataset.

# 

If there are two classes, then balanced data would mean 50% points for each of the classes. In our case, there was a 50:1 ratio between the high-risk and low-risk classes which made it highly unbalanced. It should be noted that we didn’t work on correcting this given the scope of this project.

Hence, the best classifying algorithm recommended to be used to model our data would be the Decision Tree given the extensive advantages it had on our pre-processed data. We also noted that exploratory analysis on the data allowed us to understand our data better and ensure that un-necessary and missing information was pre-processed accordingly.

# References

1. <https://www.kaggle.com/mishra5001/credit-card?select=previous_application.csv>.
2. Pre-processed file: [Clean-file](https://drive.google.com/file/d/1AWqLcNx1-PdPTGksORQiBsAIqp4ztbGq/view?usp=sharing)

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# Appendix

|  |  |
| --- | --- |
| **Team member** | **Contributions** |
| Olawin, Ridwan | * Final Project Formatting * Set up and co-ordinated weekly meeting with group members * Attended weekly meetings * Worked on Pre-processing the data * Worked on finding an alternative for the SVM algorithm * Worked on the Conclusion |
| Wu, William | * Attended weekly meetings * Worked on choosing algorithms to use on the pre-processed data * Analysed the results of the chosen algorithm |
| Ta, Quan | * Attended weekly meetings * Worked on choosing algorithms to use on the pre-processed data * Analysed the results of the chosen algorithm |
| Lee, Hajun | * Worked on Pre-processing the data * Attended weekly meetings * Documented and took major responsibility for pre-processing the data in Weka |