NATIONAL UNIVERSITY OF SINGAPORE
Faculty of Science

Predicting Credit Fraud Using Various Machine Learning Algorithms

Indrik Wijaya (A0107380M)
Christopher Hendra (A0130979U)
April 13, 2017

TABLE OF CONTENTS

LIST OF FIGURES

1.0 Introduction

In class, we have learnt a few classification techniques. Starting off with the simple perceptron algorithm, we progressed to more complicated and useful techniques such as Support Vector Machines (SVMs), SVM variant with slack, regularisation, non-linear classification via kernel methods, AdaBoost and EM algorithm.

In this project, we will explore a few methods to perform binary classification. To motivate this journey, we consider a problem that is quite challenging yet very important: *Given some features about a credit card transaction, can we create a model that can accurately predict fraudulent transactions?* This will allow us to minimise undetected frauds which create huge costs for the bank and/or the customer and minimise false fraud alert which may cause the customer to switch to another bank.

We ran our experiments on a Python3 Jupyter notebook. The choice of Python3 was due to existence of useful packages such as NUMPY, MATPLOTLIB, SCIPY and SCIKIT-LEARN.

2.0 The dataset

To tackle this problem, we collect dataset that 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 imbalanced, 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. Unfortunately, due to confidentiality issues, we cannot get access to the the original features and more background information about the data. Features **V1, V2, . . . , V28** are the principal components obtained with PCA and 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. Feature **Amount** is the transaction Amount. Feature **Class** is the response variable and it takes value **1** in case of fraud and **0** otherwise.

Due to the imbalanced nature of the dataset (very few cases of fraud transaction), we need to use another metric to measure the performance of our model. If we would naively count the fraction of classes we got right (i.e. correctly predicted 0 or 1), already the simple model that always predicts 0 (non-fraud) would achieve an "accuracy" of 99.827 %, since only frauds are not recognized correctly, and they are obviously rare. However, such a model is nonsense, since it cannot tell us whether a given transaction is a fraud or not. And the extremely high accuracy is misleading, and does not tell us anything about the actual quality of the prediction. Hence, we need a more sophisticated approach to evaluate a prediction.

2.1 Handling imbalanced dataset

Before we further explain how we handle this problem, we first introduce some terms that will be helpful for our analysis.

- **Precision**,p: the probability that a transaction classified as fraud is actually fraudulent. The precision of any model dealing with highly imbalanced dataset should be as large as possible (close to 1) as it is indicative of the model prediction power over minority labels. On the other hand, a precision close to 0 means that a fraud alert will turn out as a mistake in the majority of the cases.
- **Recall**(True Positive Rate), r: the probability that a fraudulent transaction is recognized by the classifier. The recall of our model should be close to 1 in order to detect fraudulent transaction with high probability. A recall of 60% means that the classifier can only recognise fraudulent transactions around half of the times.
- **Fallout**(False Positive Rate): the probability that a non-fraudulent transaction is wrongly classified. As most of the transactions made are non-fraudulent, a high fallout rates will dramatically increase the number of false alarms sent to non-fraud customers. As such, we aim to minimise the fallout rate of our prediction
- F_1 **score**: a measure of a test's accuracy that considers both the precision p and recall r in which we take the harmonic mean of precision and recall: $F_1 = 2\frac{pr}{p+r}$

The classification of imbalanced dataset in our context involves the trade off between maximising our sensitivity towards suspicious transactions (i.e the recall rate) and minimising the fallout. To further illustrate this point, please be noted that undetected frauds creates costs for the bank and/or the customers and we would like to ensure that our model can capture all fraudulent activities. However, if we increase the sensitivity of our model (i.e. the recall), we will inevitably increase the rate of 'false alarms', and this can be very damaging. Imagine, if a customer gets a false fraud alert every week, he or she will consider changing to another bank. Keeping false alarms small means keeping the Fallout small, or (almost) equivalently, keeping the precision of our model high.

In this experiment, we perform parameter tuning by considering the F_1 score of our model in place of the normally used accuracy. We aim to find the parameter that maxismises the F_1 score.

2.2 Training and testing sets

To ensure fair evaluation, we partition the dataset into 2 groups - the training set (70%) and testing set (30%). We may choose to further split out a evaluation set from the training set to tweak hyper-parameters, or perform k-fold cross validation on the training set. In any case, it is imperative to hide the testing set from the model training process. After the model is trained, we then evaluate its performance by asking it to predict the labels of the testing set.

3.0 Methods

We explored 4 different binary classification methods:

1. Logistic Regression
2. Decision Trees
3. Random Forest
4. AdaBoost, ala SAMME

3.1 Logistic Regression

Logistic Regression aims to find the best fitting, and most parsimonious model, to describe the relationship between a response or outcome variable, and a set of explanatory or predictor variables. This model predicts the probability of occurences in which if the odds of occurences are higher than a certain fixed number, then the prediction will be assigned to class denoted by binary variable "1" (fraud), if less it is of class "0" (normal).

We write our model as $h_\theta(x) = \theta_0 + \sum_{k=1}^{n} \theta_k x_k$, where θ's are our parameter estimates and $x_i's$ are our independent variables. We want to estimate parameters that will maximise our probabilistic model. This is done by maximising the log-likelihood regression function $l(\theta)$. We use logit probabilistic model $g(z) = \frac{1}{1+e^{-z}}$ here. Then, our model can be written as $h_\theta(x) = \frac{1}{1+e^{-\theta^T \mathbf{x}}}$. We are then interested in finding $P(y = 1|X;\theta) = h_\theta(x)$ and $P(y = 0|X;\theta) = 1 - h_\theta(x)$.

More compactly, $P(y|X;\theta) = (h_\theta(x))^y(1 - h_\theta(x))^{1-y}$.

Assuming that, m training samples are generated independently, the likelihood function is

$$L(\theta) = \prod_{i=1}^{m} P(Y^{(i)}|x^{(i)}l\theta) = \prod_{i=1}^{m} (h_\theta(x^{(i)})^{y^{(i)}}(1 - h_\theta(x^{(i)}))^{1-y^{(i)}}$$

It is equivalent with maximising the log-likelihood which is computationally easier

$$l(\theta) = \sum_{i=1}^{m} [y^{(i)} log h(x^{(i)}) + (1 - y^{(i)}) log(1 - h(x^{(i)}))]$$

Other than that, we also optimize parameter C in the logistic regression. This is done through performing a grid search.

3.2 Decision Trees

Decision tree is among the simplest and easy to visualise classification algorithm. It works by partitioning the sample space into many partitions and assign a specific probability of a datapoint $x_i \in R_m$ being labeled y_i. In this paper, we implement the CART variation of the decision tree algorithm using Gini index as our impurity measure.

Define

$N_m = \# \{x_i \in R_m\}$, $\qquad c_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i$,

$\hat{p_{mk}} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$, $\qquad Q_m(T) = \sum_{k \neq k'} \hat{p_{mk}} \hat{p_{mk'}}$

Also define the cost complexity criterion $C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|$. The idea of the algorithm is to first grow the tree by going through each node one at a time, then split them based on specific criteria. At every node we select a specific feature j and s that will minimise the impurity measure Q_m for x_{ij} in each partition $x_{ij} \leq s$ $x_{ij} > s$ until each partition comprises a single data point or a pre-defined minimum number of data points.

Furthermore, in order to reduce overfitting problem that arises from over-partitioning the dataset, Breiman et al. (1984)[1] suggested pruning the tree by successively collapsing the internal node that produces the smallest per node increase in $\sum\limits_m N_m Q_m(T)$ and continue to do so until we find subtree T_α that minimises $C_\alpha(T)$

3.3 Random Forest

Random forest is an ensemble classification method that creates a whole committee of decision trees to classify datasets. This algorithm makes use the idea of bagging and random selection of features subset in order to minimise the correlation between each decision tree and reduces the variance of the prediction. Typically, and in this paper, we choose the square root of the total number of features to train each decision tree classifier in the random forest model. We produce the Random Forest algorithm below [2].

[1]For further references, see [1]
[2]For further references, see [2]

Algorithm Random Forest

```
1: for $b = 1$ to $B$ do:
2:     Draw a bootstrap sample $Z^*$ of size $N$ from the training data.
3:     Grow a random-forest tree $T_b$ to the bootstrapped data, by recursively repeating the
       following steps for each terminal node of the tree until the minimum node size $n_{min}$ is reached
         a) Select $m$ variables at random from the $p$ variables
         b) Pick the best variable/split-point among the $m$
         c) Split the node into two daughters node
4: end for
5: Output the ensemble of trees $\{T_b\}_1^B$
```

To make a prediction at a next point x:

Let $\hat{C}_b(x)$ be the class prediction of the b^{th} random-forest tree. Then $\hat{C_{rf}^{B}}(x)$ is the majority vote of $\left\{\hat{C}_b(x)\right\}_1^B$

3.4 AdaBoost, ala SAMME

We produce the AdaBoost algorithm below.

- In class, $K = 2$, which will be the same as the case in this project
- Here, we use Decision Trees the base learners.
- In class, $\alpha_m = 0.5 log\frac{1-\epsilon_m}{\epsilon_m}$
- In class, predict new labels via $\sum_{m=1}^{M} \alpha_m$.sign$(h_m(x) = k)$ and rounding off to '+1' or '-1'. It is easy to check that taking this argmax formulation gives the same prediction.

We also perform a grid search to obtain the optimal number of base learners.

```
Algorithm AdaBoost ($n$ training data points, 2 output classes, $M$ base learners)
Initialise weights $W_0(t) = \frac{1}{n}$ for $t = 1, 2, \dots, n$
1: for $m = 1$ to $M$ do
2:     $\hat{W_{m-1}}$ = Normalised $W_{m-1}$
3:     Fit a new base learner $h_m$ that minimises weighted classification error
4:     Compute error (zero-one loss) incurred by $h_m : \epsilon_m = \sum_{t=1}^{n} \hat{W_{m-1}}(t).\text{sign}(y_i \neq h_m(x_i))$
5:     Compute $\alpha_m$
6:     Update weights: $W_m(t) = \hat{W_{m-1}}.exp(\alpha_m.\text{sign}(y_i \neq h_m(x_i)))$
7: end for
8: Prediction of new data point $x = argmax_K \sum_{m=1}^{M} \alpha_m.\text{sign}(h_m(x) = k)$
```

4.0 Experiments

Due to the size of our dataset, we first run logistic regression with $l1$ penalty in order to remove several variables that are not so statistically significant in predicting fraudulent transactions. Below are the coefficients of our logistic regression with $l1$ penalty

Amount	Time	V1	V10	V11	V12	V13
7.67×10^{-4}	-3.80×10^{-6}	8.37×10^{-2}	-7.81×10^{-1}	-5.14×10^{-2}	6.68×10^{-2}	-3.17×10^{-1}
V14	V15	V16	V17	V18	V19	V2
-5.56×10^{-1}	-1.11×10^{-1}	-1.98×10^{-1}	-2.10×10^{-2}	-3.05×10^{-4}	8.44×10^{-2}	1.06×10^{-3}
V20	V21	V22	V23	V24	V25	V26
-4.19×10^{-1}	3.79×10^{-1}	6.08×10^{-1}	-9.93×10^{-2}	1.08×10^{-1}	-7.00×10^{-2}	0
V27	V28	V3	V4	V5	V6	V7
-7.55×10^{-1}	-2.69×10^{-1}	-1.07×10^{-2}	6.80×10^{-1}	1.19×10^{-1}	-1.10×10^{-1}	-8.07×10^{-2}
V8	V9					
-1.78×10^{-1}	-2.73×10^{-1}					

We then proceed to drop the variable "Amount" , "Time" , "V18".

We now show the training times and performances in Table 1 and confusion matrix in Figure 1.

- For Logistic Regression, we find the optimal value of C to be 10 after performing a grid search and use $l2$ norm.
- For decision trees, the splitting criterion of choice is gini index. We do not limit the maximum height of the tree. SCIKIT-LEARN uses a optimised variant of CART.
- For Random Forest, we find the optimal number of estimators to be 28 and minimum sample leaves to be 70 after performing grid search method
- The base learners used in SAMME were decision stumps and we used $M = 270A$ after performing a grid search.

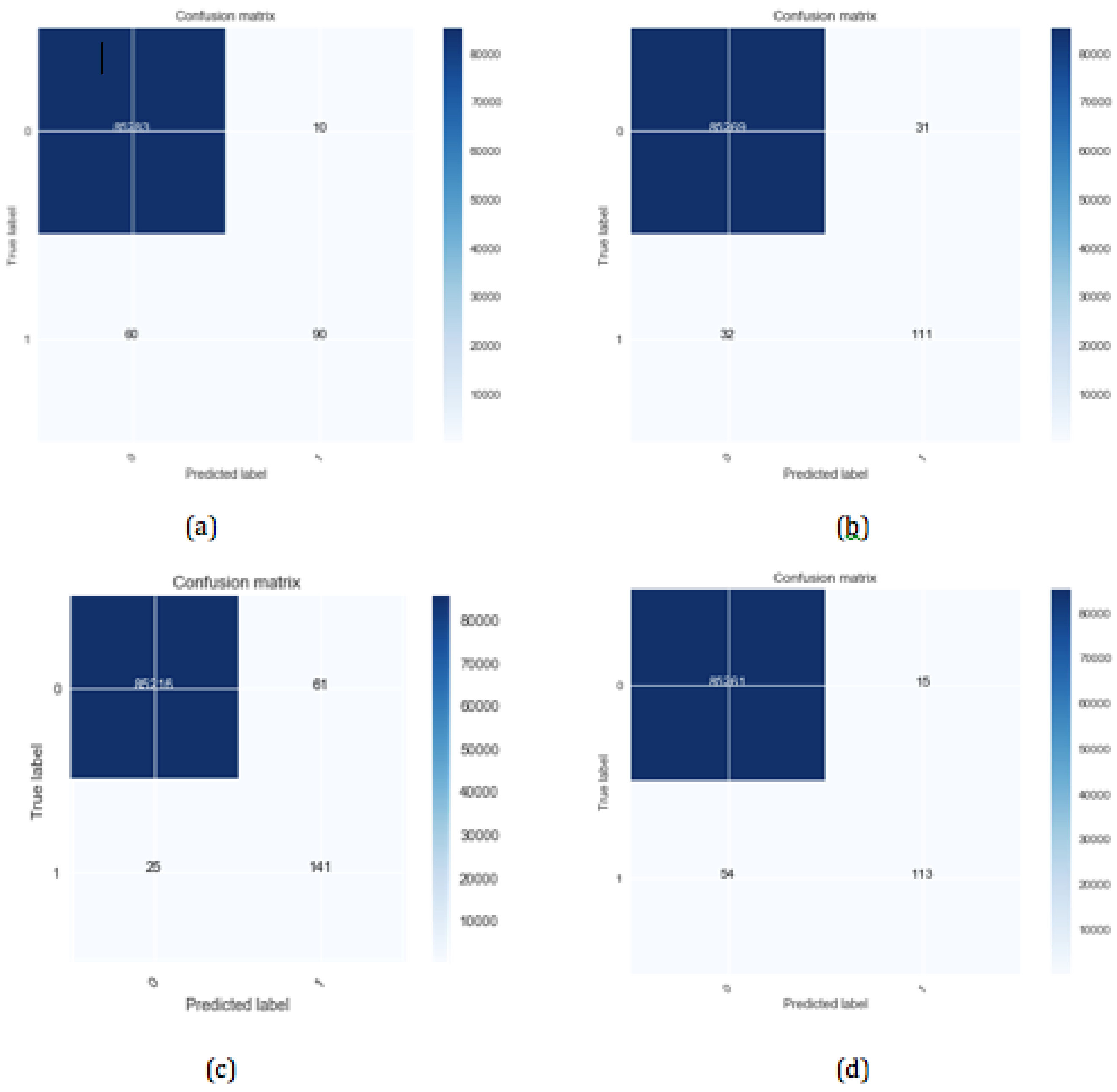

Figure 1: Confusion matrices of the 4 classifiers on the dataset. (a): Logistic Regression; (b): Decision Trees; (c): Random Forest; (d): AdaBoost

Algorithm	**Training Time**	**Precision**	**Recall**	**Fallout**	**F_1 score**
Logistic Regression	34.58510	0.900	0.600	1.172×10^{-4}	0.720
Decision Trees	13.48287	0.782	0.776	3.634×10^{-4}	0.779
Random Forest	21.75055	0.698	0.849	7.153×10^{-4}	0.761
AdaBoost	272.67311	0.883	0.677	1.759×10^{-4}	0.766

Table 1: Performance summary of the 4 explored binary classification algorithms.

5.0 Conclusion

Our result seems to suggest that there is no significant difference in terms of F_1 score between our models. Surprisingly, decision tree classifier that has higher tendency to overfit the data seems to perform better against ensemble methods like Random Forest or AdaBoost. We suspect the reason behind this peculiar result lies in the way we sample our data for the purpose of training these classifiers. As our cross-validation samples each data point with equal probability, then it is only natural that we have very few fraudulent transactions in our data and this might have significantly affected the quality of our base learners in the ensemble methods. As such, one possible improvement to this result is to take into account the sampling strategy to over-represent the minority class in this data

REFERENCES

[1] Breiman, L. (1984). Classification and regression trees Regression trees. Belmont: Wadsworth/Thomson Learning.

[2] Hastie, T., Tibshirani, R., & Friedman, J. H. (2016). The elements of statistical learning: data mining, inference, and prediction. New York: Springer.