Financial Fraud Detection Using Machine Learning Techniques

Yordan Ivanov

Variable Description

N - number of observations

$\{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$ - binary classification dataset, N pairs of features and dependent variables

X - vector of features, with dimensions $N \times p$

$y_i \in \{-1, 1\}$ - scope of values which the dependent variable can take

Y - vector of dependent variables (or classes), with dimensions $N \times 1$

β - coefficient produced by fitting logistic regression

w - weight vector normal to the hyperplane in SVM algorithm.

Φ - nonlinear mapping function

b - constant for the SVM algorithm

ξ - slack variable in SVM algorithm

C - misclassification cost in SVM algorithm

$K(x_i, x_j)$ - kernel functions

γ - paramater in radial kernel function

B - number of bootstrapped datasets in Random Forest algorithm

Z^* - bootstrapped data sample

T_b - tree grown on bootstrapped data sample

s - randomly chosen subset of features

$\Psi(y_i, \rho)$ - loss function

ρ - terminal nodes

λ - regularization parameter

F - number of nodes in neural network hidden layer

g - number of classes

1.Introduction

What we provide here is an extensive study of machine learning methods on different both real-world and simulated datasets in an attempt to provide better guidelines for fraud detection.

- specify the methods that are going to be used in the study

2. Financial Fraud

3. Methodology

Classification is one of the most widely used model framework used for the application of machine learning techniques in terms of FFD (Ngai et al. 2011). Some of the most common classification techniques include logistic regression, neural networks, support vector machine and decision trees.

3.1. Preliminaries

In the current section, we will give a description of the machine learning techniques that we apply to predict fraudulent transactions.

Let us define our binary classification dataset as $\{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, where $x_i \in \mathbb{R}^n$ represents an n-dimensional data point and $y_i \in \{-1, 1\}$ represents the label of the class of that data point, $i = 1, ..., p$. Let X represent the vector of features and Y the vector of dependent variables.

3.2. Machine Learning Algorithms

3.2.1. Logistic Regression

The logistic regression framework falls under the category of generalized linear models and allows the prediction of discrete outcomes. Then, by defining the probability of a transaction being fraudulent by $p(X) = Pr(Y = 1|X)$, we can portray the relationship between the dependent and independent variables as follows:

$$p(X) = \frac{e^{\beta_0+\beta_1 X_1+...+\beta_p X_p}}{1 + e^{\beta_0+\beta_1 X_1+...+\beta_p X_p}} \quad (1)$$

The number of independent variables is indexed by p. After manipulating (1), we can also see that

$$log(\frac{p(X)}{1 - p(X)}) = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p \quad (2)$$

with the LHS being called the logit. Using equation (2), we will predict the probabilities of a transaction being fraudulent i.e. $p(Y = 1)$. The fitting of a logistic regression is done by the method of maximum likelihood (see Appendix). The logistic regression has been among the most widely used framework in fraud detection (Ngai et al. 2011) due to simplicity of ease of implementation, but it does have its shortcomings - it tends to underperform when there are multiple or non-linear decision boundaries (*SEARCH FOR SOME PAPER OR BOOK ON IT?*)

3.2.2. Neural Networks

Neural Networks is a term that currently describes a wide array of different algorithms. However, we will be using the "vanilla" version, which contains only a single hidden layer, but this can be extended to include more [friedman2001elements]. However, that is beyond the scope of this paper.

A feed-forward neural network with one hidden layer can be seen in Figure 1. It has g outputs nodes, which in our case would be $g = 2$, as we are dealing with two-class classification problem. Each output node would give us the probability of an observation belonging to a specific class.

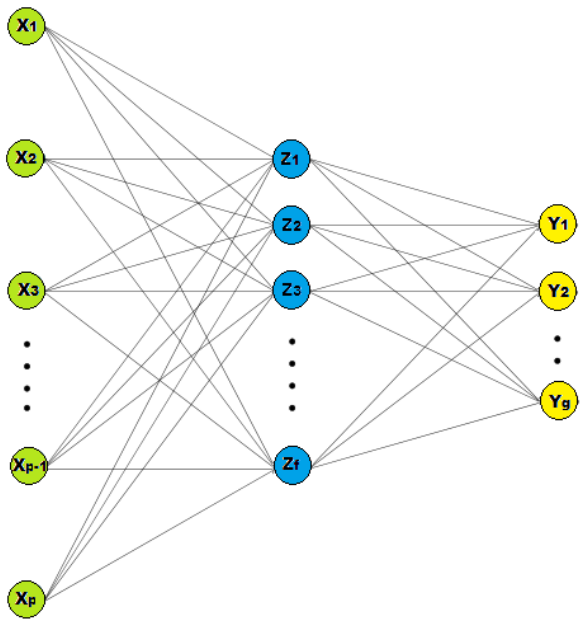

Figure 1: Neural Network with 1 hidden layer

The features Z_f are being derived through linear combinations of the inputs, while the Y_g outputs are then modeled as a function of linear combinations of the Z_m, or mathematically formulated as follows:

$$Z_f = \sigma(\alpha_{0f} + \alpha_m^T X),\ f = 1, ..., F$$
$$T_g = \beta_{0g} + \beta_k^T Z,\ g = 1, 2$$
$$f_g(X) = g_g(T),\ g = 1, 2$$

where $T = (T_1, T_2)$, $Z = (Z_1, Z_2, ..., Z_F)$ and $g_g(T) = \frac{e^{T_g}}{\sum_{f=1}^{F} e^{T_f}}$ represents the softmax function. We use the sigmoid activation function $\sigma(v) = \frac{1}{1+e^{-v}}$. In order to fit the neural network and estimate the set of weights $\{\alpha_{0f}, \alpha_f; f = 1, 2, 3..., F\}$ and $\{\beta_{0g}, \beta_g, g = 1, 2\}$, we use the backpropagation equations in order to minimize the error term.

3.2.3. Support Vector Machines

Support Vector Machines, developed by Vapnik et. al. (Cortes and Vapnik 1995), have become a popular machine learning method that has seen its implementation rise in various domains that require the use of classification models (Batuwita and Palade 2013). Among the factors for its success is the fact that the SVMs are linear classifiers, which work in a high-dimensional feature that represents a non-linear mapping of the input space of the problem being dealt with (Bhattacharyya et al. 2011). Working in a high-dimensional feature space has its benefits - often, the problem of non-linear classification in the original input space is transformed to a linear classification task in the high-dimensional feature space.

The goal of the SVM classifier consists of finding the optimal separating hyperplane, which manages to effectively separate the observations from the data into two classes. As mentioned above, the observations are initially transformed by a nonlinear mapping function Φ. Thus, we can write a possible separating hyperplane that resides in the transformed higher dimensional feature space by:

$$w \cdot \Phi(x) + b = 0 \quad (3)$$

with w the weight vector normal to the hyperplane.

We will further use two variations of the SVM soft margin optimization problem - one that assigns the same cost for missclassification of the different classes and one that penalizes more the missclassification of the minority class.

Non-cost sensitive learning

For the same missclassification cost case, we can write the soft optimization problem as follows:

$$\begin{gathered} \min(\frac{1}{2} w \cdot w + C \sum_{i=1}^{p} \xi_i) \\ s.t. \;\; y_i(w \cdot \Phi(x_i) + b) \geq 1 - \xi_i \quad (4) \\ \xi_i \geq 0, i = 1, ..., p \end{gathered}$$

The slack variables $\xi_i > 0$ hold for missclassified examples. Thus, the penalty term $\sum_{i=1}^{p}$ can be perceived as the total number of missclassified observations of the model. Thus from (4), we can see that there are two goals - maximizing the

margin the minimizing the number of missclassifications. The cost parameter C controls the trade-off between them, i.e. assigned misclassification cost. The quadratic optimization problem in (4) can be represented by a dual Lagrange problem and then solved:

$$\max_{\alpha_i}\{\sum_{i=1}^{p}\alpha_i - \frac{1}{2}\sum_{i=1}^{p}\sum_{j=1}^{p}\alpha_i\alpha_j\Phi(x_i)\cdot\Phi(x_j)\} \;\; s.t. \;\; \sum_{i=1}^{p} y_i\alpha_i = 0, \;\; 0 \leq \alpha_i \leq C, \;\; i = 1, ..., p \quad (5)$$

α_i are the Lagrange multipliers also satisfying the Karush-Kuhn-Tucker (KKT) conditions (see Appendix). Thanks to another one of the strenghts of SVM - kernel representation - we don't need to explicitly know the mapping function $\Phi(x)$, but by applying a kernel function (i.e. $K(x_i, x_j) = \Phi(x_i)\cdot\Phi(x_j)$), we can rewrite (5) as:

$$\max_{\alpha_i}\{\sum_{i=1}^{p}\alpha_i - \frac{1}{2}\sum_{i=1}^{p}\sum_{j=1}^{p}\alpha_i\alpha_j K(x_i, x_j)\} \;\; s.t. \;\; \sum_{i=1}^{p} y_i\alpha_i = 0, \;\; 0 \leq \alpha_i \leq C, \;\; i = 1, ..., p \quad (6)$$

The solution then gives us $w = \sum_{i=1}^{p}\alpha_i y_i \phi(x_i)$ for the optimal values of α_i and w, while b is determined from KKT. The data points that have α_i different than zero are called the support vectors. Thus, the decision function can be written was:

$$f(x) = sign(w\cdot\Phi(x) + b) = sign(\sum_{i=1}^{p}\alpha_i y_i K(x_i, x_j) + b) \quad (7)$$

Cost Sensitive learning

The regular SVM model has been effectively implemented when the dataset used has balanced classes, however it fails to produce good results when applied on imbalanced data (Batuwita and Palade 2013). When trained on a dataset with extremely imbalanced classes, the SVM framework could produce so skewed hyperplances that all observations are recognized as the majority class (Akbani, Kwek, and Japkowicz 2004, @veropoulos1999controlling). This is due to the fact that when we take the soft margin optimization problem, we try to maximize the margin and minimize the penalty for the misclassifications. As we consider a constant C for all training examples, the minimization of the penalty is achieved through the minimization of all misclassifications. However, when the used dataset suffers from imbalanced classes, the majority class density would be higher than the minority class density, even when considering the class boundary region (through which the ideal hyperplane would pass).

Thus, we consider here the application of the Different Error Costs (DEC) variation of the SVM algorithm proposed by Veropoulos et al. (1999). The DEC method introduces different misclassification costs - C^+ for the minority and C^- for the majority class. With the inclusion of the higher misclassification cost for the minority class observations, the imbalanced class effect could be brought down. The soft margin optimization problem then has the following form:

$$\min (\frac{1}{2} w \cdot w + C^+ \sum_{i|y_i=+1}^{p} \xi_i + C^- \sum_{i|y_i=-1}^{p} \xi_i)$$
$$s.t. \;\; y_i(w \cdot \Phi(x_i) + b) \geq 1 - \xi_i \quad (8)$$
$$\xi_i \geq 0, i = 1, ..., p$$

The Dual Lagrange optimization form is the same as before, with the exception of replacing $0 \leq \alpha_i \leq C$ with $0 \leq \alpha_i^+ \leq C^+, \; 0 \leq \alpha_i^- \leq C^-$ for $i = 1, ..., p$. The α_i^+ and α_i^- are the Lagrange multipliers. The solution of the DEC dual Lagrangian problem follows the same outline as in the normal form.

Kernels

As mentioned before, we used a kernel function ($K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$) in order to transform the dual Lagrange problem. The advantages of using kernel functions are computational or in some cases it allows for computations that otherwise would be impossible (James et al. (2013)). For the purpose of this study, we are using the linear and radial kernels. The radial kernel shows good performance on non-linear class separation. They have the following representations:

$$Linear: \quad K(x_i, x_j) = \sum_{k=1}^{l} x_{ik} x_{jk} \quad (9)$$

$$Radial: \quad K(x_i, x_j) = exp(-\gamma \sum_{k=1}^{l} (x_{ik} - x_{jk})^2), \;\; \gamma = \frac{1}{2\sigma^2} \quad (10)$$

3.2.4. Tree-Based Methods

Tree-based methods involve segmentation of the feature space into a set of regions and then fitting a simple model to each one. Even though are not too complex conceptually, they are still a very powerful method (J. Friedman, Hastie, and Tibshirani 2001). Firstly, we will give a short overview of a standard

classification decision tree (DT) before moving on the methods used in this study.

Let us call the set of non-overlapping regions $R_1, R_2, ..., R_M$ that are used to divide the feature space. The forms of those regions are high-dimensional rectangles - for simplicity and interpretability. The aim for DT would be to find the boxes that minimize the error term, which in the case of classification can be represented in several ways - misclassification error, Gini index or cross-entropy. However, it is very computationally taxing to consider every feasible partition of the feature space into M boxes. Thus, the recursive binary splitting method is used, which is a top-down, greedy approach - it starts at the top of the tree and then it splits the feature space, making the best split possible, without looking forward. The algorithm is further described in the Appendix.

However, the beforementioned algorithm can sometimes lead to over-fitting and producing very ineffecient prediction results. Thus, the tree pruning technique is applied - first a large tree is grown, then it is "pruned" and a smaller version is obtain. The procedure leads to reduction in variance at the cost of some bias. Usually the cost complexity pruning algorithm is used in practice - a more thorough description is included in the Appendix.

The regular classification DT has high interpretability, but it sometimes lacks sufficient prediction power - they are often unstable and can be too sensitive to training data (Bhattacharyya et al. 2011). This leads us to variations of the classic classification DT.

Random Forests

A random forest algorithm is an ensemble of classification trees (Breiman 2001). The model starts with growing each tree on separate bootstrapped dataset. Moreover, only a randomly selected feature subset, typically $s \simeq \sqrt{p}$ (Khoshgoftaar, Golawala, and Van Hulse 2007), is used at each individual node. As a result, the entire algorithm is based on two basic, yet powerful, concepts - bagging and random subspace method. To better illustrate the random forest method, the pseudo-algorithm is given below:

Algorithm Random Forest

1. For $b = 1$ to B (B is representing the number of bootstrapped datasets):
    - Draw bootstrapped sample Z^* of size N from the dataset used for training

- Grow a RF tree T_b on the previously bootstrapped dataset by recursively looping the below given steps for each tree terminal node, up until the n_{min} node (minimum size node) is reached:
  - Select s features randomly from the entire feature subset, p.
  - Choose the best available variable/split-point among the s.
  - Split node into two children nodes.

2. Output the tree ensemble $\{T_b\}_1^B$.
3. Making the prediction for a new point x: If $\widehat{C}_b(x)$ is the prediction of the class for the bth RF tree, then $\widehat{C}_{rf}^B(x) = majority\ vote\{\widehat{C}_b(x)\}_1^B$.

The Random Forest algorith has been quite popular lately, due to its simplicity and performance. It has only two parameters that can be changed - the number of trees grown and the size of the feature subset used (Breiman 2001). Furthermore, it has often shown superior performance to one if its rival statistic learning method - the SVM (Meyer, Leisch, and Hornik 2003). In the area of financial fraud detection, RF has also shown to be a promising framework (Whitrow et al. 2009, @ngai2011application).

Stochastic Gradient Boosting Machines and eXtreme Gradient Boosting Machines

The ideas introduced by the boosting methodology have been among the most influential in the last twenty years (J. Friedman, Hastie, and Tibshirani 2001). Among the most used and influential boosting algorithms is "AdaBoost.M1" (Freund and Schapire 1997). It was the first developed "adaptive" boosting algorithm, due to its incorporated function to directly adjust its parameters to the used training dataset. This is due to the fact, that the performance of the model was re-evaluated at each iteration - the parameters weights and the final aggregated weights, were re-calculated and improved at each iteration, thus leading to an overall better performance. Moreover, it was found that not only AdaBoost, but boosting algorithms in general, managed to not only reduce variance, but also bias - an improvement over bagging, where only the variance could be decreased.

The followwing success led to the formulation and development of the gradient-descent based methods, which received the name gradient boosting machines or simply GBM (Freund and Schapire 1997; Friedman et al. 2000; J. H. Friedman 2001). The general idea of GBMs is to fit new models iteratively, constructing the base-learners to be maximally correlated with the loss function's negative gradient, in order to get a better estimate of the response variable (Natekin

and Knoll 2013). Moreover, the algorithm offers flexibility in terms of choosing the form of the loss function. In this study, we use the Bernoulli distribution, as we are dealing with a two-class classification problem.

Due to its flexibility and ease of implementation, the GBM algorithm has proven to be a successful model, that offers high predictive power when dealing with machine learning problems (Whiting et al. 2012, @johnson2014learning). Furthermore, when including a random element in the algorithm, as in the Stochastic GBM (SGBM), results tend to improve (Friedman 2002). We describe the pseudo-code of the SGBM for a two-class problem below.

Stochastic Gradient Tree Boosting Machines (Stochastic GBM)

Two-Class Classification as in the **gbm** R package.

1. Initialize $f_0(x) = \arg\min_\rho \sum_{i=1}^{N} \Psi(y_i, \rho)$.
2. For $m = 1$ to M do:
    - Compute negative gradient:

$$z_{im} = -\frac{\partial \Psi(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}|_{f(\mathbf{x}_i)=\widehat{f}(\mathbf{x}_i)},\ i = 1, ..., N$$

    - Randomly select $s \times N$ subset.
    - Fit regression tree with K number of terminal nodes on the previously selected subset, $g(\mathbf{x}) = E(z|\mathbf{x})$.
    - Compute the optimal terminal node predictions, $\rho_{1m}, ..., \rho_{Km}$, as:

$$\rho_{km} = \arg\min_{\rho} \sum_{\mathbf{x}_i \in S_k} \Psi(y_i, \widehat{f}(\mathbf{x}_i) + \rho_k)$$

      , where S_k is the set of $\mathbf{x}$'s that define terminal node k.
    - Compute $\widehat{f}_m(x) = \sum_{k=1}^{K} \rho_{km} I(x_i \in \widehat{R}_{km})$, as $\{\widehat{R}_{km}\}_{k=1}^{K}$ represents the tree structure.
    - Update $\widehat{f}^m(\mathbf{x})$ as $\widehat{f}^m(\mathbf{x}) \leftarrow \widehat{f}_{m-1}(\mathbf{x}) + \lambda \widehat{f}_m(x)$.
3. Output $\widehat{f}(x) = f_M(x)$

The eXtreme Gradient Boosting algorithm, or shortly XGBoost, was developed as an attempt to improve on the performance of GBM, both in terms of speed and prediction power (Chen and Guestrin 2016). It has has not only succeeded in doing so, but it has also become one of the most used models (Nielsen 2016). Among the differences that XGBoost introduces in comparison to the GBM model is that the former uses clever penalization of the individual trees - the

leaf weights are not all decreased at the same rate, instead the weights which are estimated without much evidence from the training set will be shrunk more heavily relative to others. Furthermore, the XGBoost employs another type of boosting structure when compared to GBM - Newton boosting. Due to this, the algorithm manages to better learn tree structures, leading to learning better neighbourhoods. XGBoost also incorporates a randomization paramater, which contributes to the individual tree decorellation and reducing variance. Some technical detail references regarding the differences between XGBoost and GBM are included in Appendix 9.1 - such as the gradient and Newton boosting and leaf weight learning. The general pseudo-codes are similar.

3.3. Cross-Validation

The framework for model training that we use in this study is a k-fold Cross-Validation (CV) with $k = 10$ (James et al. 2013). The method incorporates a random division of the training set into k folds with approximately similar size. The initial fold is used a validation set and the chosen statistical learning model is fit on the remaining $k - 1$ folds. The error is then calculated on the observations in the held-out fold. The entire process is then repeated k times, with a different validation set at each iteration, thus computing k estimates of the test error. The k-fold CV estimate can then be shown as:

$$CV_{(k)} = \frac{1}{k} \sum_{i=1}^{k} Err_i$$

where $Err_i = I(y_i \neq \widehat{y}_i)$.

There are other approaches to CV, such as the Leave-One-Out CV (LOOCV), but the k-fold CV has shown to be superior in both computational time and estimate accuracy (J. Friedman, Hastie, and Tibshirani 2001). The latter is connected to the bias-variance trade-off problem, but a discussion is out of the scope of this paper. A graphical representation can be seen in Figure 1 in Appendix Section 9.4.

4. Data

4.1. Datasets

The number of datasets that we will be using in this study is four - two real-world and two simulated. Each dataset has been randomly splitted to two - a train and a test subset. The train consists of 60% of the original dataset, while the test the remaining 40%.

4.1.1. Real-World Datasets

UCSD-FICO Data Mining Contest 2009

The dataset was released by FICO, one of the leading analytics providers, and the University of California, San Diego (UCSD). It consists of real-world e-commerce transactions and it was released in two versions - an "easy" and a "hard" one. We will be using the "hard" version for the assessments of the models used. Due to the fact, that is a real-world data set, anonymiziation is introduced, which means that methods depending on feature aggregation or feature engineering will not lead to improvement in efficiency (Seeja and Zareapoor 2014). Nonetheless, some data preprocessing is done.

The dataset consists of 100 000 transactions made by 73 729 different customers throughout 98 days. Each transaction is characterized by 20 features - amount, hour1, state1, zip1, custAttr1, field1, custAttr2, field2, hour2, flag1, total, field3, field4, indicator1, indicator2, flag2, flag3, flag4, flag5. It can be observed that custAttr1 is the customer card number, while the custAttr2 is the e-mail address. As both fields are unique, we discard custAttr2. The other unique feature pairs per customer are the amount/total, hour1/hour2 and state1/zip1, thus discarding total, hour2 and state1 leaves us with 16 fields.

Furthermore, customers with just one transaction have been removed, leaving us with 40 918 transactions. Only 2.922% of the transactions are fraudulent, which indicates severe imbalacedness.

Université Libre de Bruxelles (ULB) Machine Learning Group

The dataset was released by the Université Libre de Bruxelles (ULB) Machine Learning Group and it consists of real-world credit card transactions made throughout two days from the year 2013 (Dal Pozzolo et al. 2015). Strong

anonymization is introduced, as the entire dataset has gone through a Principal Component Analysis (PCA) transformation.

The number of transactions included are 284 807 with 30 features characterizing each one. The only two labeled features are Time and Amount, while all 28 others are numericals resulting from the PCA. We do not do any feature engineering, as there is no information on what each column represents.

The number of frauds is just 492, which means only 0.172% of all transactions are positively labeled, indicatin very severe class imbalance.

4.1.2. Simulated Datasets

Due to lack of publicly available datasets on real-world fraud, the simulation of such has become important (E. A. Lopez-Rojas and Axelsson 2014). Thus, we will use two generated sets in order to evaluate the chosen statistical learning methods and gather more data on their performance in the area.

PaySim

The first synthetic dataset that we will use originates from a simulator called PaySim, which was developed by E. A. Lopez-Rojas and Axelsson (2014). It represents a simulation based on an e-commerce payment platform for making payments through a mobile device.

The generated dataset is constructed thanks to an agent-based-simulation application, which uses real-world information. It consists of nine features - step, type, amount, nameOrig, oldbalanceOrg, newbalanceOrig, nameDest, oldbalanceDest, newbalanceDest. The step represents a time-stamp, the type shows what kind of transaction occurs, nameOrig and nameDest give us unique ID's of the sender and receiver of the transaction, while the rest oldbalanceOrg, newbalanceOrig, oldbalanceDest and newbalanceDest represent the amount present in the accounts of the sender and receiver before and after the financial transaction has been made. Furthermore, we introduce some feature engineering by creating two more variables - errorBalanceOrig and errorBalanceDest. The errorBalanceOrig is generated by summing up newbalanceOrig and amount and then subtracting oldbalanceOrg. The motivation behind this is that a positive errorBalanceOrig could be a fraud pattern, due to an amount of being lost along the line of the transaction.

Overall, the dataset consists of 6 362 620 observations, but only 0.129% of them are fraudulent, indicating very severe class imbalance.

BankSim

The second synthetic dataset that we will be working on was generated by a simulator called BankSim, developed by E. Lopez-Rojas and Axelsson (2014). It is a simulation application based on aggregated transactional data provided by a bank entity in Spain.

4.2. Problems of Imbalanced Data and Data Sampling Techniques

4.2.1. Problem of Imbalanced Data

One of the biggest challenges faced in detecting fraudulent transactions is the one of unbalanced class sizes, with legitimate class outnumbering vastly the fraudulent one (Bhattacharyya et al. 2011). The application of data-sampling techniques has been widely used in the literature with various results when combined with different algorithms, as when such a problem occurs, it could hinder the model performances (Van Hulse, Khoshgoftaar, and Napolitano 2007). Moreover, in our particular case, the cost of missclassifying the minority class could prove to be a lot more costly than predicting wrongly the majority one.

4.2.2. Data Sampling Techniques

Random Oversampling (ROS)and Random UnderSampling (RUS)

The two techniques are the simplest and most common (Van Hulse, Khoshgoftaar, and Napolitano 2007). In minority oversampling (ROS), the observations from the minority group are randomly duplicated in order to balance the dataset. In majority undersampling (RUS), the aim is the same, but it is achieved by randomly removing observations of the majority class.

SMOTE

The Synthetic Minority Oversampling Technique (SMOTE), proposed by Chawla et al. (Chawla et al. 2002), artificial minority instances are created not simply through duplication, but rather with the extrapolation between preexisting observations. The technique starts by taking into account the k nearest neighbourhoods to a minority observation for every instance from that class. Then, the artificial observation are created, taking into account just a part of the nearest neighbours or all of them (with respect to the desired oversampling specification).

5. Results

6. Further Improvements

7. Conclusion

8. References

Akbani, Rehan, Stephen Kwek, and Nathalie Japkowicz. 2004. "Applying Support Vector Machines to Imbalanced Datasets." In *European Conference on Machine Learning*, 39–50. Springer.

Batuwita, Rukshan, and Vasile Palade. 2013. "Class Imbalance Learning Methods for Support Vector Machines."

Bhattacharyya, Siddhartha, Sanjeev Jha, Kurian Tharakunnel, and J Christopher Westland. 2011. "Data Mining for Credit Card Fraud: A Comparative Study." *Decision Support Systems* 50 (3). Elsevier:602–13.

Breiman, Leo. 2001. "Random Forests." *Machine Learning* 45 (1). Springer:5–32.

Chawla, Nitesh V, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. "SMOTE: Synthetic Minority over-Sampling Technique." *Journal of Artificial Intelligence Research* 16:321–57.

Chen, Tianqi, and Carlos Guestrin. 2016. "Xgboost: A Scalable Tree Boosting System." In *Proceedings of the 22nd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining*, 785–94. ACM.

Cortes, Corinna, and Vladimir Vapnik. 1995. "Support-Vector Networks." *Machine Learning* 20 (3). Springer:273–97.

Dal Pozzolo, Andrea, Olivier Caelen, Reid A Johnson, and Gianluca Bontempi. 2015. "Calibrating Probability with Undersampling for Unbalanced Classification." In *Computational Intelligence, 2015 Ieee Symposium Series on*, 159–66. IEEE.

Freund, Yoav, and Robert E Schapire. 1997. "A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting." *Journal of Computer and System Sciences* 55 (1). Elsevier:119–39.

Friedman, Jerome H. 2001. "Greedy Function Approximation: A Gradient Boosting Machine." *Annals of Statistics*. JSTOR, 1189–1232.

———. 2002. "Stochastic Gradient Boosting." *Computational Statistics & Data Analysis* 38 (4). Elsevier:367–78.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2001. *The Elements of Statistical Learning*. Vol. 1. Springer series in statistics New York.

Friedman, Jerome, Trevor Hastie, Robert Tibshirani, and others. 2000. "Additive Logistic Regression: A Statistical View of Boosting (with Discussion and a Rejoinder by the Authors)." *The Annals of Statistics* 28 (2). Institute of Mathematical Statistics:337–407.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An Introduction to Statistical Learning*. Vol. 112. Springer.

Johnson, Rie, and Tong Zhang. 2014. "Learning Nonlinear Functions Using Regularized Greedy Forest." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36 (5). IEEE:942–54.

Khoshgoftaar, Taghi M, Moiz Golawala, and Jason Van Hulse. 2007. "An Empirical Study of Learning from Imbalanced Data Using Random Forest." In *Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th Ieee International Conference on*. Vol. 2. IEEE.

Lopez-Rojas, EA, and S Axelsson. 2014. "BankSim: A Bank Payment Simulation for Fraud Detection Research." In *26th European Modeling and Simulation Symposium, Emss 2014*, 144–52.

Lopez-Rojas, Edgar Alonso, and Stefan Axelsson. 2014. "Social Simulation of Commercial and Financial Behaviour for Fraud Detection Research." In *Social Simulation Conference. Bellaterra, Cerdanyola Del Valles, 1a: 2014*.

Meyer, David, Friedrich Leisch, and Kurt Hornik. 2003. "The Support Vector Machine Under Test." *Neurocomputing* 55 (1-2). Elsevier:169–86.

Natekin, Alexey, and Alois Knoll. 2013. "Gradient Boosting Machines, a Tutorial." *Frontiers in Neurorobotics* 7. Frontiers:21.

Ngai, EWT, Yong Hu, YH Wong, Yijun Chen, and Xin Sun. 2011. "The Application of Data Mining Techniques in Financial Fraud Detection: A Classification Framework and an Academic Review of Literature." *Decision Support Systems* 50 (3). Elsevier:559–69.

Nielsen, Didrik. 2016. "Tree Boosting with Xgboost-Why Does Xgboost Win" Every" Machine Learning Competition?" Master's thesis, NTNU.

Seeja, KR, and Masoumeh Zareapoor. 2014. "FraudMiner: A Novel Credit Card Fraud Detection Model Based on Frequent Itemset Mining." *The Scientific World Journal* 2014. Hindawi.

Van Hulse, Jason, Taghi M Khoshgoftaar, and Amri Napolitano. 2007. "Experimental Perspectives on Learning from Imbalanced Data." In *Proceedings of the 24th International Conference on Machine Learning*, 935–42. ACM.

Veropoulos, Konstantinos, Colin Campbell, Nello Cristianini, and others. 1999. "Controlling the Sensitivity of Support Vector Machines." In *Proceedings of the International Joint Conference on Ai*, 55:60.

Whiting, David G, James V Hansen, James B McDonald, Conan Albrecht, and W Steve Albrecht. 2012. "Machine Learning Methods for Detecting Patterns of Management Fraud." *Computational Intelligence* 28 (4). Wiley Online Library:505–27.

Whitrow, Christopher, David J Hand, Piotr Juszczak, D Weston, and Niall M Adams. 2009. "Transaction Aggregation as a Strategy for Credit Card Fraud Detection." *Data Mining and Knowledge Discovery* 18 (1). Springer:30–55.