Binary Classification on a Highly Imbalanced Dataset

Detecting credit card fraud based on transaction data by using a novel Bump Hunting method

Author:
Tom Peters

Supervisors:
Pieter-Jan van Kessel, MSc (PwC)
Prof. Dr. Arno Siebes (UU)
Dr. Karma Dajani (UU)

A thesis presented for the degree of
MSc Mathematical Sciences

Universiteit Utrecht
Netherlands
July 13, 2018

Abstract

Credit card fraud is a significant problem that has proven to be both costly and difficult to restrain. As current systems fail to efficiently detect these fraudulent transactions, an increasing interest goes out to data-driven methods. This research approaches this challenge as a binary classification problem, for which most relevant solutions are introduced. However, due to the high level of imbalance between the amount of fraudulent and legitimate transactions, some additional processing has to be done.

Within this research, the task of resolving the issues induced by the imbalance between the two classes is approached by applying data-level solutions. For resampling the pool of training data, a widespread range of options is described and explored. Furthermore, a novel Bump Hunting method is introduced which, instead of targeting specific observations, attempts to restrict the volume of the feature space. In this way a more balanced situation is achieved which can be generalized and extended to different datasets and unobserved observations.

Through experimental testing on an actual credit card transaction dataset, the result of each method and their differences are investigated. After implementing and empirically testing the novel bump hunting method, it is concluded that indeed it is able to compete with the state-of-the-art resampling methods known throughout literature.

Acknowledgements

First and foremost I want to thank my daily supervisor Pieter-Jan van Kessel at PwC. Not only, especially in the beginning of this journey, for the hours of discussions and brainstorm sessions we had in defining the directions of this research project, but also for professionally mentoring me throughout the entire nine months here at PwC. I have learned a lot!

Furthermore, I would like to thank my supervisor Arno Siebes for the input in this project, the very interesting and constructive discussions we had and for sometimes slowing me down when I lost control of what I wanted this project to become. I also would like to thank Karma Dajani, not only for agreeing to be the second reeder, but also for all tutoring and guidance throughout my entire master trajectory.

A final thanks to all my colleagues at PwC, and especially PAIS. For everything they taught me about pensions, insurance, actuarial services, data analytics and many more, but mostly for all the fun lunches and nice conversations we had.

Contents

Chapter 1

Introduction

Credit cards have taken a major share of all non-cash transactions. They provide the cardholder with a small extra line of credit, they require very little effort in use, are widely accepted and on top of that they often involve a benefit and rewards program. Unfortunately, the use of credit cards did invoke a new and versatile circuit of fraud. For the detection of fraud, based on suspicious behavior amongst certain transaction, data driven methods are proposed.

Problem description

The detection of fraudulent transactions can be broken down to a binary classification problem: one wants to label each new observation either legitimate or fraudulent based on the characteristics of the transaction. On this particular problem, a large amount of research has already been done. This specific setting does however introduce a slight altercation to the general binary classification domain: the two outcome classes are highly imbalanced. This means that it is assumed that, in a normal situation, one would expect to observe a lot more legitimate transactions as opposed to the fraudulent ones. While this of course is fortunate for the financial industries involved, it does make the classification task more challenging.

Throughout this research, this problem of imbalanced classification will be addressed. After a very brief introduction on the actual meaning of credit card fraud, its impact and the measures that are already being taken to counter it, the content of this document can roughly be divided into a theoretical and an experimental part. The theoretical part aims to extensively study the problem and its potential solutions, while the experimental part serves as an empirical validation of these theoretic concepts.

Research goals

There are two main goals behind this research project. First, this work aims to serve as an extensive **survey** on the most relevant data-level solutions that can be found throughout literature. Not only will they be stated, and their algorithms will be written out, but also the reasoning behind a specific method and why it should work in a given setting is investigated. It is precisely this aspect that is slightly neglected in comparable reviews. This is enhanced by the experimental section where specific attention is being put into relating the (relative) performances to the theoretical motivations behind each method.

Furthermore, a **novelty method** on the data-level will be proposed which aims to overcome the issues that arise from high levels of imbalance in alternative manner. It is based on the domain of subgroup discovery which thus far has not been linked to imbalanced classification. Formalizing the specific procedure, together with the introduction of a specialized validation structure, and a theoretical motivation behind this method will be the second main research goal. After developing

an implementation in `R`, the performance of this novelty method will be thoroughly compared to the existing state-of-the-art approaches.

Structure

The remainder of this thesis is structured as follows:

- Chapter 2 features the theoretical background behind **binary classification.** After the general setting is defined, together with some notation, the section contains a description of five of the most relevant known classifiers that can be applied to this specific setting. The mechanics behind each classifier are briefly explained together with corresponding strengths and weaknesses and, if relevant, its effects are visualized by using an illustrative and openly available dataset.
- Chapter 3 focuses on learning with **imbalanced data.** The chapter starts with an extensive investigation into the origin and definition of imbalanced data and, more importantly, why it introduces extra difficulties in the classification task. After this, a wide array of resampling (data-level) methods which aim to overcome these difficulties are stated. Each method is accurately described such that if needed they could easily be implemented. The original motivation and (potential) impact behind each method is presented and accompanied by a visualization of its effects on a randomly generated dataset.
- Chapter 4 introduces the novelty **bump-hunting method.** The original subgroup discovery method on which this approach is based is explained in detail, after which altercations are presented to include it in the current setting if learning with imbalanced data. Throughout the chapter arguments are provided on why this method is being used and where it could possibly improve on current techniques. This is enhanced at the end of the chapter by defining a theoretical setting on which the bump hunting method is compared to the common method of resampling, encountered previously.
- Chapter 5 provides the necessary **experimental framework.** The first section treats all relevant measures in order to evaluate the performance of a classifier, with special focus on skew-invariant metrics needed in this setting. Furthermore, the credit card fraud dataset that is used for the experimental part is introduced and explored. Finally, some benchmark testing between multiple implementations of the featured classifiers is performed in order to make sure that they can be compared without introducing a certain bias.
- Chapter 6 displays the experimental **results** of both the methods featured in the survey and the introduced bump hunting method as well as the necessary comparisons between them.
- Chapter 7 contains some concluding remarks about this research, as well as some possibilities for future further exploration of this topic.

1.1 Credit card fraud

According to the 2017 World Payments Report [1] the global transaction volume for credit cards reached a total of 85 billion transactions. As a result, credit cards gained a market share of about 30% of all non-cash transactions. As a consequence, credit card fraud is a major and ongoing problem which can even become more severe if the proper repercussions will not be taken. This section will provide a brief overview about the mechanisms behind credit card fraud, its impact on society and some ways in which it currently is being prevented.

What is fraud? According to American dictionary *Merriam-Webster*, fraud is defined as: *intentional perversion of truth in order to induce another to part with something of value or to surrender a legal right.* Within this definition, there are two types which are generally distinguished when it comes to credit card fraud [2, 3]:

- **card-not-present fraud:** The most common [4] and inventive way of credit card fraud. A cards information is obtained by the fraudsters without even possessing or even seeing the physical card itself. Multiple ways exist to commit CNP fraud such as recreating false messages from the credit card company or sending contaminated links, often referred to as *phishing*, or through for instance hacking or a data leak. Until the fraudster makes a transaction, there usually is no way of knowing that a customer has been a victim to CNP fraud.

- **stolen card fraud:** The more straightforward way of committing fraud. In this case the actual card is involved, either through direct theft (pickpocketing e.g.) or through methods such as *skimming*.

It is important to realize that, in both of the above mentioned distinctive ways of credit card fraud, it is crucial for the fraudster to handle quick and in such a way that he or she can make the most out of the crime. This as, once a fraud is being suspected or discovered the card is being deactivated and repercussions against the fraudster will be attempted to take. It is exactly this suspected **abnormal behavior** that potentially can be used for preventing fraud before it can incur any real damage.

Impact

In the latest issue of *The Nilson Report* [6], which is an independent annual report analyzing the global card and mobile payment industry, the worldwide loss due to payment card fraud has reached an amount of 22.80 billion dollars. They even project this number to be much higher (32,96 billion) by the end of 2021. Apart from that, the latest report on card fraud issued by the *European Central Bank* [5] stated that just within the considered European countries (SEPA) the total share of fraud amongst credit cards is increased to over 0.010% of the total value of transaction. Apart from that, as mentioned above, the CNP fraud's share of total frauds steadily increased to about two-thirds, with a total value of close to a billion Euro. This is shown in figure 1.1, which is copied from the ECB report [5]

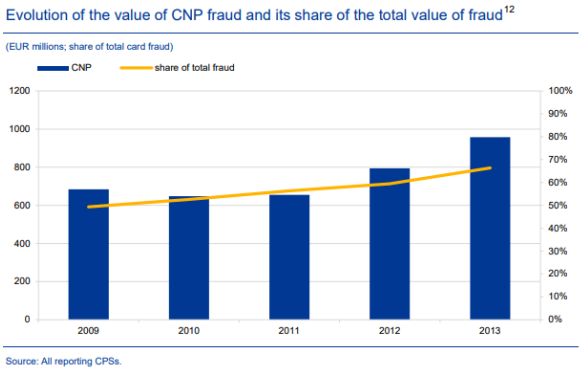

Figure 1.1: Source: *European Central Bank* [5]

These damages do not only affect the card issuers, who most of the time have to reimburse the clients for their losses after being defrauded, but also parties like merchants and retailers who offer the service of the possibility of paying through credit card, or the individual card owners itself. Think for instance about extra repercussions that merchants must take in order to be compliant with the required safety level. And on customer level, although client liability is usually limited up to a fixed and relatively low amount (or zero), this can still result in financial consequences. Then after that, upon which party fall the complete liability, will have to bear the costs of refund.

Countering fraud

Several actions are being taken against credit card fraud. They can be roughly divided into two categories: **prevention** and **detection**.

Prevention of fraud takes place before the transaction is being made. Most of these prevention measures involve a screening process to verify the validity of a future transfer. Examples are a card verification code (CVC) which is only printed on the back and not embedded in the account information of magnetic strip hence attempts to specifically prevent CNP fraud, or user authentication

like a password or an PIN code. Other less straightforward methods are setting up lists or databases with specific suspicious fields such as card numbers, countries or addresses which are suspected to have a higher fraud risk.

On the other hand, the detection of fraud happens after a transaction has been authorized. It involves, based on the **attributes** of a transaction, assessing whether the transaction belongs to the group of legitimate or fraudulent transactions. These attributes are for instance amount of the transaction, a timestamp, location and of course account of sender and recipient. Based on this information, suspicious transactions can be **flagged**, after which further human investigation is performed to put a definitive label on it. These investigations are time consuming and costly, hence it is important that these flags are placed as efficient and precise as possible.

Placing these flags, or more precisely, assigning risk scores to each transaction usually goes trough a rule based approach. These rules are made by fraud experts based on current knowledge and can be as simple and obvious as:

$$\begin{gathered}\text{AMOUNT} > 5000 \\ \text{COUNTRY\_RECIPIENT} = \textit{Afghanistan} \\ \text{TRANSACTION\_FREQUENCY} > 5 \text{ x } \textit{normal},\end{gathered}$$

or combinations of similar boolean operations. Risk scoring of this kind has some obvious advantages. They are easy to develop, interpret and execute and they directly involve the developed expert knowledge and experience on the domain. On the other hand, these methods also have some important drawbacks. Rule systems are static and do not evolve as methods of committing fraud evolve, unless they are altered through human intervention, which means it requires constant supervision. On top of that they depend fully on the experts ability to understand and explain all kinds of fraud. This makes them rather subjective as expert's opinions can vary largely.

The biggest issue however, is that this method generates a lot of **false positives.** A single high transaction can just as easily imply that the card owner has bought a new car, or a sudden increase in frequency can be due to holiday shopping or a wild night out. As all of these 'false flags' have to be humanly investigated this can result in an inefficient and expensive process. Some literature even suggests that these costs can succeed the loss due to actual fraud [7].

An alternative way of placing these flags is through a **data-driven** system. In this approach, fraud detection systems are set-up by using the knowledge from past transaction and automatically processing them with the goal of accurately predicting future instances. Machine learning techniques can be used to build and maintain these systems, without the need of constant human updating/supervision. On top of these reduced labor costs, the system can maintain its relevance through continuously learning from past events, and it can detect and reconstruct far more complex multidimensional patterns than a human could ever do.

Chapter 2

Classification

The task of credit card fraud detection, as formulated in the final part of the previous section, is in essence a classification problem: given a certain amount of information about a transaction, put a single label on it. In the following section, classification will be formalized in a general setting, and the most relevant classification algorithms, or classifiers, will be treated.

2.1 Statistical learning

Before diving into specific components and methods for the classification task, it is important to formalize the *statistical learning* setting, from which classification is a sub-area. Throughout this document, a few general definitions will be used. The most important component that is featured in every learning setting is the **data**, denoted by X. Usually, a data set X consists of N different data points which are individually addressed through subscript X_i. Each individual data point itself lies within the *data space* denoted by $\mathcal{X}$, which usually is a feature vector with p different features and corresponding values. These can be both numerical and categorical (in practice some components may even be empty). Apart from data points or the data set, throughout this document components of X may also be referred to as sample points or the more general term *observations*.

Within the scope of this project, the focus is limited to the task of *supervised learning*, meaning that input data X always maps to an output (or response) y. Note that this y can either range over a continuous interval (regression) or be a member of distinct pre-defined classes (classification). As this research is focused on (credit card) fraud detection, the model is restricted to have *binary output*, or formally $y \in \{0, 1\}^N$. Throughout this document the class with $y = 1$ will be referred to as the positive class (mostly), hit class or minority class (in the imbalanced framework), where the class with $y = 0$ will be referred to as the negative class, non-hit/miss class or majority class.

The goal of classification is as follows: given the available set of labeled observations, find a predictor that maps unlabeled data with similar structure to one of the outcome classes. Or formally:

- Given observations set $\mathrm{S} = \{\mathrm{X}_i, \mathrm{y}_i\}_{i=1}^N$ where $\mathrm{X}_i \in \mathcal{X}$ and $\mathrm{y}_i \in \{0,1\}$,
- Find predictor $\mathrm{f} : \mathcal{X} \to \{0, 1\}$.

This predictor, which may also be referred to as classifier or hypothesis, is generated through an algorithm that processes the available data S in a certain manner. Each algorithm uses another approach or assumption about the data in attempt to find a good predictor. Therefore understanding the functionalities and underlying ideas and assumptions of different algorithms is key in the task of classification. Some algorithms may directly map new sample points to either one of the two classes, while others produce class probabilities (formalized in equation (2.1)). As mapping these to a class is straightforward (obvious choice is pick the class which has the highest probability) the difference

between a direct class label or class probabilities as output will not be distinguished throughout this document.

2.1.1 Prim Indian diabetes

Furthermore, in this section, a brief theoretical background about the most important classification algorithms that are used within this research will be provided. Also, when relevant, the specific algorithms will be illustrated by implementing them on the `PimaIndiansDiabetes` dataset, which can be accessed directly through the `mlbench` library in `R`. The reason for choosing this dataset is that, as desired, it has binary output, and a selected number of easily-interpretable attributes which aides in explaining and understanding how specific models operate.

This dataset features the data of the medical records for occurrence of diabetes within a group of 768 female Pima Indians. Each sample contains 8 attributes such as age, blood pressure, body-mass index (BMI) and plasma glucose concentration (Glucose) during a medical examination (where the last two attributes are most prominently featured in this section, see figure 2.1) combined with a binary label column containing the positive/negative diagnosis of diabetes. It is important to note that the scope is, by no means, to find or tune the best model for the most accurate predictions, but merely to illustrate and visualize the functionality of different algorithms.

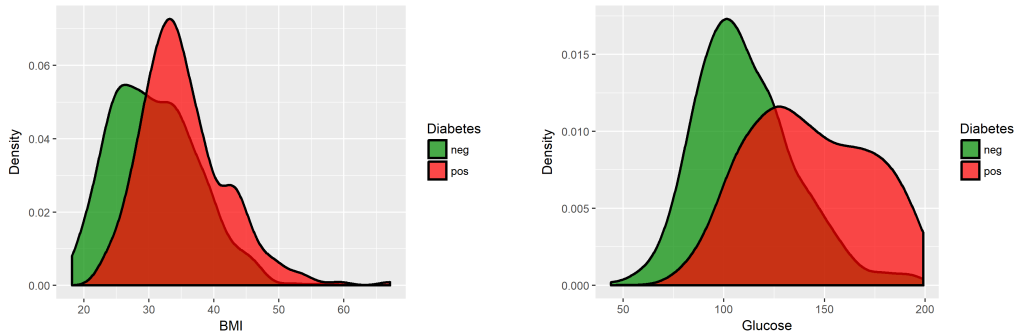

Figure 2.1: Density plots of the occurrence of diabetes in the Indian data set for two attributes: BMI (left) and Glucose (right).

2.2 Logistic regression

Throughout the field of machine learning generalized linear models always have been one of the most popular learning methods. They are intuitively easy to explain, the implementation is very straightforward, generally require little computational capacity and the results are interpretable. In the case of classification, the most common implementation used is the *Logistic Regression* model, which aims to reconstruct the underlying posterior distribution $\mathbb{P}(y|X)$. In the binary domain, this can be further abbreviated by estimating

$$\mathbb{P}(X) := \mathbb{P}(y = 1|X) \tag{2.1}$$

which in the logistic regression model is represented by:

$$\mathbb{P}(X; \beta_0, \hat{\beta}) = \frac{e^{\beta_0 + \hat{\beta}^T X}}{1 + e^{\beta_0 + \hat{\beta}^\top X}} \tag{2.2}$$

with parameters β_0 and $\hat{\beta} = (\beta_1, ..., \beta_p)$. It is straightforward to see that equation (2.2) can be rewritten as

$$\log\left(\frac{\mathbb{P}(X)}{1-\mathbb{P}(X)}\right) = \beta_0 + \hat{\beta}^T X, \tag{2.3}$$

from where we can see that the logistic regression model has *log-odds* (left hand side of equation (2.3)) which are linear in X.

Likelihood function

The usual procedure to fit the logistic regression model is through *maximum likelihood estimation* using the probability function defined in (2.2). The likelihood function over the data (X,y) is defined as:

$$\begin{aligned} L(\beta_0, \hat{\beta}) &= \prod_{i:y_i=1} \mathbb{P}(x_i; \beta_0, \hat{\beta}) \prod_{i:y_i=0} (1 - \mathbb{P}(x_i; \beta_0, \hat{\beta})) \\ &= \prod_{i=1}^{N} \mathbb{P}(x_i; \beta_0, \hat{\beta})^{y_i} (1 - \mathbb{P}(x_i; \beta_0, \hat{\beta}))^{1-y_i} \end{aligned} \tag{2.4}$$

Taking the log on both sides results in:

$$\begin{aligned} \ell(\beta_0, \hat{\beta}) &= \sum_{i=1}^{N} \Big(y_i \log \mathbb{P}(x_i; \beta_0, \hat{\beta}) + (1 - y_i) \log(1 - \mathbb{P}(x_i; \beta_0, \hat{\beta})) \Big) \\ &= \sum_{i=1}^{N} \Big(y_i(\beta_0 + \hat{\beta}^T x_i) + \log(1 - \mathbb{P}(x_i; \beta_0, \hat{\beta})) \Big) \\ &= \sum_{i=1}^{N} \Big(y_i(\beta_0 + \hat{\beta}^T x_i) - \log(1 + e^{\beta_0 + \hat{\beta}^T x_i}) \Big), \end{aligned} \tag{2.5}$$

where in both steps the definition of equation (2.2) is used. The log-likelihood (2.3) is maximized over the available training data through gradient descent or, more commonly, through the Newton–Raphson method [8] (as the second derivative over the parameters can easily be computed).

Characteristics of logistic regression

Evidently, by the way this model is defined (2.2), the output of the logistic regression model when being presented with a new observation is a probability of it belonging to the positive (and thereby negative) class. By defining a probability threshold (0.5 by default) the model returns a distinct class prediction (displayed by the dotted line in the left side of figure (2.2)). As it will turn out later, the fact that class label probabilities are directly computed does provide certain advantages in analyzing and validating the performance of the model. Another advantage for interpretation, which follows directly from equation (2.3), is that evaluating the model results in a **linear decision boundary** within the feature space. This means that the decision boundary can be visualized clearly, especially in the two-dimensional case (figure 2.2). This does however means that the method assumes this linearity of the boundary, which can also be a potential downside of the method, as this is obviously not always the case.

2.3 Decision trees

The foundation on which the, very frequently used, second class of classifiers is built is the decision tree. This approach of classifying disjunct areas of the feature space is very different from that

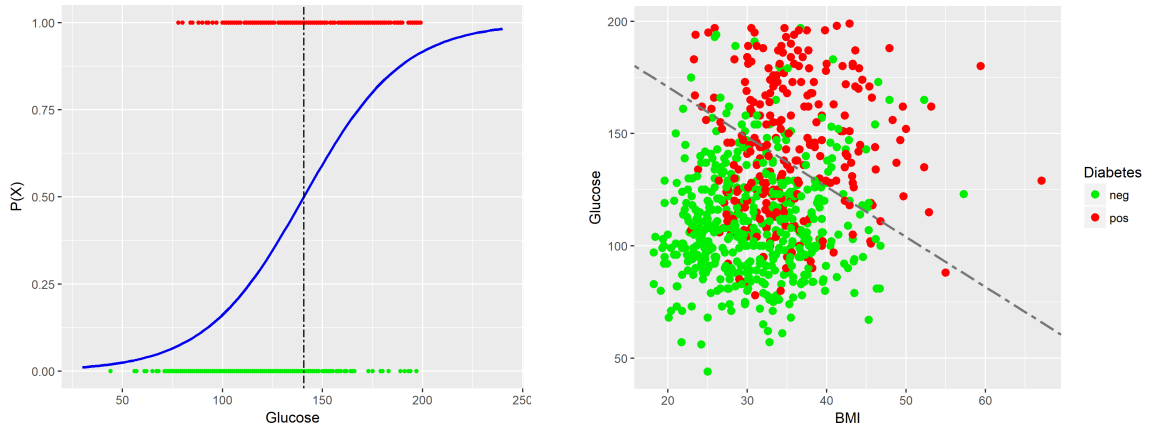

Figure 2.2: Result of applying the logistic regression algorithm (`glm` implementation in `R`) to the diabetes data. The figure on the left shows the predicted class probability (2.2) based on one attribute (Glucose). The right figure illustrates the (linear) decision boundary when performing logistic regression on two attributes

of generalized linear models. As opposed to attempting to fit a certain predictive function for the response in terms of the set of features, this method proceeds by directly partitioning the feature space into **rectangular areas**, each belonging to a specific outcome (class label). This is visualized in figure 2.3 by using the same two-dimensional setting as in the previous section.

The way these partitions are made is by a *top-down greedy* approach [9]. This means that the process starts with the entire feature space, and proceeds by sequentially making partitions within one of the regions that resulted from previous splits. Each partition is made by a binary split ($x_i < v$) in terms of one of the features. This process can be represented as a tree where the top represents the entire feature space and at each node the specific region is divided into a branch of points that either satisfy (left) or not satisfy (right) the split condition. For the two dimensional case (that was treated in the right side of figure 2.2) such a tree is displayed in figure 2.3. On the right, the result of each split (the blue digits) is drawn in the feature space.

After $n-1$ splits, the entire feature space is divided into regions $R_1, ..., R_n$. In the binary classification case, each region i gets assigned label l_i according to a *majority vote* over the class of each data point that falls within the region. From these regions and their corresponding labels, a predictor is defined as follows:

$$f(X) = \sum_{i=1}^{n} l_i \cdot \mathbb{1}\{X \in R_i\}. \tag{2.6}$$

This immediately implies that, within a region, every unlabeled point that is observed within it is assigned to the same class. This can result in one or multiple decision boundaries (for instance the line segment with parts of splits 1,3 and 5 on the right side of figure 2.3) and possibly multiple disjoint clusters that are assigned to the same class.

Tree construction

What remains is the task of determining which split to make at each step. To formalize this, denote

$$\hat{p}_{mk} = \frac{\sum\limits_{\{i|y_i=k\}} \mathbb{1}\{x_i \in R_m\}}{\sum\limits_{i=1}^{N} \mathbb{1}\{x_i \in R_m\}}, \tag{2.7}$$

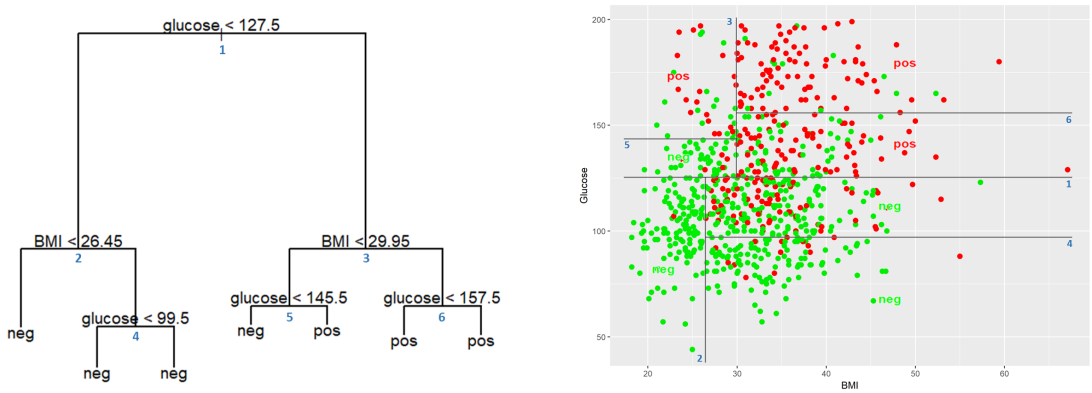

Figure 2.3: An example of the decision tree algorithm, trained by using BMI and glucose as predictors. On the left the outcome is displayed in tree form, where each split (blue number) corresponds to a partitioning within the two-dimensional feature space, visualized on the right.

so the proportion training observations having label k ($k \in \{0, 1\}$) that fall within region m. The idea is maximizing the *information gain* at each split, which is most commonly achieved through considering the Gini Index

$$G_{R_m} = \sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} \tag{2.8}$$

which is maximized when the proportions of both classes (as defined in (2.7)) are close to zero/one, which implies a region that either has a very high proportion of positive or negative training observations within its domain. Another, more straightforward *measure of impurity* that is often being used is the misclassification error of region m:

$$\frac{1}{N_m} \sum_{i \in R_m} \mathbb{1}\{y_i \neq \text{argmax}_k \hat{p}_{mk}\} \quad \text{where} \quad N_m = \sum_{i=k}^{N} \mathbb{1}\{x_i \in R_m\}. \tag{2.9}$$

By considering one of the above impurity measures and weighting them with the size of both sides of a possible split, a best feasible partitioning for each available region can be computed as a pair (j,v):

$$\min_{j,v} \left[\min N_{m_L} \cdot Q_{R_{m_L}} + \min N_{m_R} \cdot Q_{R_{m_R}} \right]. \tag{2.10}$$

Here Q_{R_m} is defined as the chosen *purity measure* on a resulting region R_m, N_m as introduced in equation (2.9) and

$$R_{m_L} = \{X | x_j \leq v\} \quad \text{and} \quad R_{m_R} = \{X | x_j > v\} \tag{2.11}$$

are the left and right region resulting from a binary split in feature j at value v.

Another point of attention is when to stop the splitting process. Of course, if the only requirement is minimizing the impurity at each final node, one could continue until each node perfectly represents a single class. Obviously this is overfitting and hence should be avoided. Options in overcoming this are setting a minimal information gain threshold to which each split must satisfy, limiting the number of splits, setting a minimal support to each final node or, preferably, apply *pruning*. This process involves growing a very large tree, and prune it back by removing splits that dot not contribute much to the eventual quality of the segmentation. This way one does not risks the possibility that the algorithm isn't allowed to make a split that, in terms of quality, did not seem very significant at that time, but resulted in a valuable split later on.

Relevance of the decision tree

The concept behind tree-based methods as showed above is, as mentioned, a tool that is used a lot throughout the field of machine learning and classification. It has a few clear advantages. First of all, both the decision making process, which is said to mimic the decision making process of the human brain quite closely, and the actual partitions that result from it (left and right part of figure 2.3) are both easily interpretable and explainable if a single tree is used. Apart from that, the method **does not require any assumptions** about the posterior distribution (for instance linearity) and furthermore it is non-parametric. A downside of using the decision tree for classification is that usually it lacks in predictable capability as opposed to other common classification methods. For that reason, the decision tree classifier on itself is not used particularly often. Methods that involve multiple trees however, are considered amongst the top of the current classifiers. This means that this high level of interpretability and the easy tree-like visualization is less straightforward, but that is being made up for in predictive capacity. One of the most relevant classifiers that directly uses multiple classification trees is the random forest.

2.4 Ensemble methods

The random forest algorithm mentioned above specifically uses classification trees in the training process. However, it is still considered an ensemble method, hence this is the first of this class of predictors that is discussed in detail in this review. The random forest classifier currently is one of the most relevant and frequently used algorithms, mainly due to the fact that it generally excels in predictive capabilities. It is essentially a modified and improved version of bagging. Although bagging by itself will not be used heavily as classification algorithm, both the ideas behind it and the technique it introduces will, hence it will first be introduced briefly.

2.4.1 Bagging

Bagging, or *Bootstrap Aggregating*, effectively trains multiple classifiers and combines them into one predictor (hence the aggregating part). Each individual classifier is trained on a bootstrap sample. The idea behind this concept is as follows: from the entire available pool of training data, multiple samples are drawn **with replacement** to form a bootstrap dataset $(X, y)^b$ of the same size as the original dataset. This method originally stems from the model assessment domain, where it is introduced to analyze the stability of a learner and estimate its true prediction error more precisely than just divide the available data into a train and a test partition.

In the bagging process, B of such bootstraps are created, and on each of them a predictor $f^b(X)$ is trained which is mapped to a predicted label by observing whether it falls above or below a certain probability threshold θ. These individual predictors are aggregated and mapped to a predicted class label either trough averaging the individual class probabilities

$$f^{bag}(X) = \frac{1}{B}\sum_{b=1}^{B} f^b(X) \text{ where } \hat{y} = 1 \iff f^{bag}(X) > \theta, \tag{2.12}$$

or through a majority vote over all individual class label predictions

$$f^{bag}(X) = \frac{1}{B}\sum_{b=1}^{B} \mathbb{1}\{f^b(X) > \theta\} \text{ where } \hat{y} = 1 \iff f^{bag}(X) > 1/2. \tag{2.13}$$

The main concept behind bagging is that it **reduces variance** by aggregating multiple predictions, hence it works best for individual predictors f^b with a high variance and low bias. Typical example of such a predictor is a large (deep) decision tree that isn't pruned. Such predictor generally has a low bias, but suffers from a lot of noise (variance), hence it is ideal for aggregating. This is illustrated

in figure 2.4, where individual trees are grown until no further split can be made where the total information increases (2.7) (a predictor that obviously does not generalize well to unseen data). As figure 2.4 indicates, the initial test error of a single tree is significantly worse than trowing a weighted dice, but this decreases as the number of bootstraps increases.

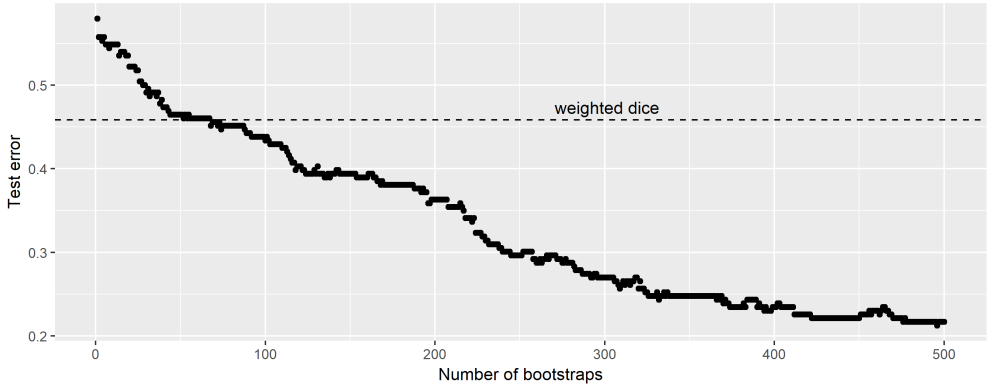

Figure 2.4: Test error of the bagging classifier, trained on bootstrap samples of the Pima Indians diabetes dataset, where a 70/30 train-test split is made for evaluation. The predicted labels are computed through majority voting (2.13).

The reason that indeed variance is reduced by bagging can be explained, as done in the original paper on bagging [10], as follows. By the underlying assumption of statistical learning, we can see each observation (X, y) as being drawn independently from distribution P, hence the aggregated predictor f^{agg} can be written as:

$$f^{agg}(X) = \mathbb{E}_P\left[\hat{f}(X)\right], \tag{2.14}$$

where $\hat{f}(X)$ is the predictor over the entire dataspace. Then for a fixed pair (X, y)

$$\begin{aligned}\mathbb{E}_P\left[y - \hat{f}(X)\right]^2 &= y^2 - 2y * \mathbb{E}_P\left[\hat{f}(X)\right] + \mathbb{E}_P\left[\hat{f}(X)^2\right] \\ &\geq y^2 - 2y * f^{agg}(X) + (f^{agg}(X))^2 = (y - f^{agg}(X))^2,\end{aligned} \tag{2.15}$$

where the equality in (2.14) is used, together with the inequality $\mathbb{E}X^2 \geq (\mathbb{E}X)^2$. Integrating both sides of (2.15) shows that, when taken over the entire dataspace, aggregating multiple predictors never increases the mean squared error. This gives an intuitive explanation on why aggregating on bootstrapped samples may reduce mean-squared error as well.

2.4.2 Random Forest

The random forest classifier builds on the concept of bagging decision trees, but with a important altercation. While combining multiple decision trees does a good job at variance reduction, there is still a potential downside. Consider the situation where few individual features act as a very strong predictor for the outcome. Then each individual tree, although optimized over different bootstrap samples, will somewhat have the same structure as it will always select either one of the dominant splits in an early stage. Hence the structures of each tree will be similar and all corresponding outcomes will be strongly correlated. This will decrease the amount of variance reduction substantially.

Decorrelation

To overcome this potential downside, the random forest algorithm [11] proceeds by, at each split, selecting a fixed number of predictors out of the pool of available features to consider for the following

split. Typically, this number is taken small in comparison to the total number of features (common practice for classification is $\sqrt{p}$). Just as with bagging, the final predictor is composed by combining the predictors of all individual (decorrelated) decision trees, either trough averaging (2.12) of through majority vote (2.13).

Apart from the before mentioned excellent predictive capacity of the random forest algorithm, there are a few more reasons why it is so widely adopted. It takes a lot **less iterations to stabilize** then other state-of-the-are ensemble methods such as *Boosting* (discussed later) and they are easy to work with as the performance is not influenced significantly by the user's input (only real required meta-parameters are number of trees to grow and number of features to consider at each split) and tuning. On the other hand, the high level on interpretability that came with the decision tree classifier is less present with random forests, especially for visualizing. However, feature importance can be computed by considering information gain at each split. This greatly helps in interpreting and understanding the predictions made by the classifier.

2.4.3 Boosting

The above mentioned classification algorithms excel in either simplicity, statistical/visual interpretability or performance, and hence all are widely used throughout literature. However, there is one more classifier that is considered to be very important: *Boosting*. Similar to random forest, boosting belongs to the class of ensemble methods which is are esteemed for their predictive capabilities.

There are multiple implementations of boosting available, but they all build on the same concept of **combining weak classifiers**. This seems similar to the bagging procedure, but it turns out the two are actually quite different. The main reason for this deviance lies in the notion of using weak classifiers; classifiers that perform only marginally better than random guessing. An example of such a weak classifier, and in fact one that is often used, is a decision *stump*, meaning a decision tree with only one binary split (2.10), yielding a predictor that performs a little better than random guessing (especially in the common case where p is quite substantial, so each element in $\mathcal{X}$ consists of a number of features).

The definition of a weak classifier can be formalized as follows: Given the training set $S = \{\mathrm{X}_i,\mathrm{y}_i\}_{i=1}^N$ compute a classifier $\mathrm{f}^w : \mathcal{X} \to \{0,1\}$ such that for $\gamma > 0$, small:

$$\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(y_i \neq f^w(X_i)) \leq \frac{1}{2} - \gamma. \tag{2.16}$$

The above equation implies that the *error rate* over all training observations must be (slightly) under 50%.

A collection of weak classifiers (2.16) can easily be computed by an algorithm (in this case a decision tree with a single split) which results in a sequence of M weak classifiers. For each classifier, a certain importance factor α_i is calculated, and the sequence is combined to a single classifier by:

$$f^{boost}(X) = \sum_{m=1}^{M} \alpha_m \cdot f_m^w(X), \tag{2.17}$$

which is equivalent to the combined predictor from equation (2.12), but with *non-uniform weight* on each individual classifier. This results in a single predictor, but yet there is no indication that this yields an improved performance. While training them on bootstrap samples (bagging) or restricting the number of features to choose the split candidate on (random forest) were previously used to reduce bias or correlation, boosting approaches the prediction challenge from a different angle. Key concept is that each weak classifier is sequentially trained on a **modified version** of the entire observation set S.

The modification of the dataspace is achieved through **weighting** each data point, and recalibrating this after each step. These weights $w = (w_1, ..., w_N)$ can easily be introduced into the optimization

phase of the algorithm. In the case of using tree-based models, this is achieved through altering equation (2.7):

$$\hat{p}_{mk} = \frac{\sum\limits_{\{i|y_i=k\}} w_i \mathbb{1}\{x_i \in R_m\}}{\sum\limits_{i=1}^{N} w_i \mathbb{1}\{x_i \in R_m\}}, \tag{2.18}$$

such that points with higher weight have a larger influence in calculating the proportions over which the algorithm optimizes. Initially, all weights are equal (1/N), and at each step the weight of each individual points is increased or decreased, depending on the way it is being processed by the previous predictor: samples that are misclassified (hence that can be considered as harder) receive a greater weight, while the weight of samples that are classified correctly (the 'simple' ones) is reduced accordingly. This way, after each iteration, the algorithm is being forced to focus more on the points that are harder to classify. This does imply that, at any time during the training process, the entire training set, as well as all available features are being used. The way this exact calibration is defined, and the way the optimization process goes, depends on which implementation is being used.

AdaBoost

A very straightforward, and probably the most significant implementation is the *AdaBoost* [12] algorithm (formally known as Adaboost.M1). It follows the steps described in the previous section. Adaboost is formalized as follows. Weights are initialized to be equal for each sample point (1/N) and a weak classifier f_1^w is trained (for instanced the above mentioned decision stump with the weights incorporated through (2.18)). Using this classifier, the weighted training error is

$$\text{error} = \sum_{i=1}^{N} p_i \cdot \mathbb{1}\{y_i \neq f_1^w(X_i)\} \text{ with } p_i = w_i / \sum_{j=1}^{N} w_j, \tag{2.19}$$

where the p_i can be interpreted as the distribution over all observations induced by the set of weights. Trough this error term the importance factor α_1 can be computed as

$$\alpha_1 = \log\left((1 - \text{error})/\text{error}\right), \tag{2.20}$$

and for all $i \in [1, N]$ the weights can be updated accordingly:

$$w_i = w_i \cdot \exp\left(\alpha_1 \mathbb{1}\{y_i \neq f_1^w(X_i)\}\right). \tag{2.21}$$

This process is repeated until all M classifiers are trained, after which they are combined according to (2.17). Interpreting equations (2.19)-(2.21) is rather straightforward. The lower the error rate, the higher the importance factor α_m which implies two things: the weak classifier f_m^w gets a larger weight in the final classifier, and the factor with which the weight of the (fewer) points that still are misclassified increases is higher.

Boosting (in general) is believed to be the algorithm with the overall best performance in terms of predictive capacity. However, a boosting classifier is very hard to interpret and it may take more computational needs and a longer time to reach a desirable solution than simpler classification algorithms or for instance random forest.

Chapter 3

Imbalanced Data

Now that the classification framework is defined, and several methods are provided with theoretical background, the next area of focus is imbalanced data. There are numerous literature articles and surveys which state and compare multiple different methods and solutions for dealing with imbalanced data [13, 14, 15, 16]. However, relatively little is written about the true underlying problem definition and the two main questions: *what defines imbalanced data and why is it considered to be a problem?* Before formalizing and comparing some of the state-of-the-art techniques for improved classification on imbalanced datasets, some further elaboration about these two important questions will be provided.

3.1 Learning with Imbalanced data

Although, as will be described in the following section, imbalance occurs in numerous types and variations, imbalanced data usually is characterized by the following definition:

> *Data with unequal number of examples in each of its classes.*

Within this definition, multiple possible types associated with data-imbalance can be distinguished (although they can occur at the same time).

Rarity

The first thing to realize, is that all forms of imbalance are the result of an underlying form of rarity. Two types are distinguished [17]. The first type is *absolute rarity*, which means that there is a lack of sufficient minority class sample points due to its overall rarity. Examples are for instance very rare diseases; maybe there are only a few patients even diagnosed with the disease. This form of rarity usually is considered to be the most challenging to learn with [17, 18] as it is very hard to find certain similarities to generalize the few known examples to unlabeled new observations. In most real life examples however, usually absolute rarity is not the case.

The second and most important form of rarity is *relative rarity*. In this case a sufficient number of minority class sample points is available, but this come with a much larger number of majority class observations. This means that the ratio of negative to positive class data is very high. Of course (as one would hope) this is the case with credit card fraud. Theoretically speaking, relative rarity should not be an issue, but to quote [19]:

> *"The key point is that relative rarity/class imbalance is a problem only because learning algorithms cannot efficiently handle such data".*

This has multiple origins which will be discussed below, but first it is important to state how these forms of rarity are expressed in the data.

Imbalance in the data

The above-mentioned types of rarity can be manifested within the data or the classification problem through multiple ways[17, 19, 13]. The most common and intuitive way is *between-class imbalance*, which means that there is an uneven ratio between two (or more) outcome- classes originating from difference in class prior probabilities. In the case of two outcome classes (such as legitimate/fraudulent) this means that one of the two classes is represented significantly more within the data. Advantage of this type of imbalance is that it is both easy to detect (simply comparing the size of both outcome classes) and easy to quantify its (relative) performance on both classes.

The second manner in which rarity can manifest itself in the data is by *within-class imbalance*. In this case, within a class, rare cases or sub-concepts can be distinguished. Examples are for instance specific uncommon types of cancer within a CT-scan dataset. The difficulties in learning with within-class imbalance are much harder to detect and quantify, as common performance metrics focus on detecting and reconstructing a certain class (cancer/non-cancer) and thus do not accurately reflect if a classifiers overlooks such rare sub-concepts. While literature is much more elaborate on learning with between-class imbalance, some research specifically focuses on this form [20].

The way that different classes are represented within the data by most classifiers is by disjunctive areas in the feature space. Each disjunct aims to maximize the correctly classified number of observations that fall within them. In the case of rarity within the data, this tends to result in *small disjuncts*. These small disjunct are proven to be more prone to error than the large disjuncts [21, 22] Also, small disjuncts within the data are hard to distinguish from noise, especially when data is imbalanced.

Finally, another result from rarity is that noise had a much larger influence on distorting the decision boundary. This *increased influence of noise* is very intuitive to explain: if one of the classes is much more represented within the dataset and hence the other class has much less observations to begin with, it will require much less 'noisy' observations (points belonging to the majority class, but located in regions that, according to the underlying distributions, should theoretically belong to the minority class) to impact the classifiers decision process. This can result in rarer (sub-)concepts vanishing due to the presence of noisy majority class points.

Learning

Now that the underlying reason of imbalance and the way it can be manifested in the data is discussed, the second question remains: why is imbalanced data considered to be a problem? It is important to realize that, by itself, the above described phenomena do not automatically imply that they hinder the task of classification in any way. But in fact, there are certain algorithmic difficulties associated with them [19].

The first issue stems from the way usual learning algorithms are designed. As pointed out in [19], most learners are structured and tested to optimize its performance over accuracy (5.1). As by definition accuracy ignores the size of individual classes, this performance metric is not suited in the case of imbalance. Consider for instance the scenario where 1 out of every 1000 transactions is fraudulent (which, as it turns out, is being conservative in terms of imbalance ratio [23]). An accuracy of 99.9% can be achieved by completely ignoring the fraud class, resulting in a worthless fraud detection system. An example of this design is for instance the way which decision trees make a split (2.8)(2.9). Both quantifications of information gain are directly related to accuracy and hence do not distinguish between classes and their size.

Another issue related to the design of algorithms are problems caused by the underlying inductive bias. To avoid overfitting, learners utilize biases or assumptions that encourage generalization over specialization, meaning that this restrains them from focusing too much on smaller classes and disjuncts. Also most learners tend to favor the majority class, for instance by averaging over the outcome space as was observed in decision trees, where labels were assigned through majority voting.

While the above mentioned algorithmic issues resulted in favoritism of one class over the other, it should not be ignored that it is also fundamentally more difficult to identify rare patterns than to

identify more common patterns. Illustrated with the needle in a haystack idiom, or to quote [19]: *"The problem is not so much that there are few needles, but rather that there is so much hay".* One explanation for this is that greedy search heuristics, which are used in numerous algorithms, fail to identify rare patterns as they are most likely a conjunction of multiple predictors, and furthermore the true signal may be obscured by common (majority) objects.

3.1.1 Relevance

So far the imbalanced data setting has been investigated a little further, but little has been said about the relevance of the topic. To illustrate this, consider the following classification-problem: predicting the tastiness of a certain fruit based on characteristics such as color and hardness. In a completely balanced situation, hence half of the pieces of fruit is considered tasty, it would be relevant to obtain such predictor, without having to try them all. But if the classes are highly unbalanced, so for instance one out of every 1000 pieces is tasty, it would be far more reasonable to just disregard the type of fruit entirely as being unpleasant.

However, all of the commonly studied imbalanced data cases such as fraud or disease detection have one thing in common: they have **non-uniform importance**. This means that misclassifying one of the classes (the minority class) is significantly more costly than the other. Missing a case of fraud completely can have far-reaching consequences, both legal and financial, but on the other hand falsely flagging a legitimate transaction as (potential) fraud may result in an awkward phone call, some extra investigative work or, at worst, losing a customer.

Precisely these **highly skewed misclassification costs**, in combination with the increased difficulties that appear, are the reason that the field of classification on imbalanced datasets is such a relevant and, throughout literature, widely studied topic.

3.2 Data pre-processing

In the previous section the key issue with learning on (highly) imbalanced datasets was explained due to the fact that available learners can not effectively process the presented data. Just by considering this statement, two obvious solutions arise: either change/alter the learner, or modify the presented dataset. The first solution can either be achieved through designing or choosing a skew-insensitive algorithm [24, 25], include individual class weights in the leaning process according to individual misclassification costs [26, 27] or alter the underlying bias of a learner.

These methods however each require certain information or knowledge which makes them less generalizable to each possible situation or classifier. Changing bias for instance limits the user in the choice of learning algorithm and including cost-proportional weights in the optimization phase requires that misclassification costs are (roughly) known for the specific domain that it is used on. This is, more often than not, difficult to pinpoint. By those arguments, the following section focuses on the second approach: **pre-processing the data**.

The advantage of this approach is that, instead of specializing the process for either a learner or a setting on which the learner is applied, developing approaches that work on data level can be implemented in every machine learning pipeline, independent of which type of problem it is, what level om imbalanced is encountered or if the order of magnitude of making an error can be estimated. This also makes method comparing more convenient, as a range of classifiers and datasets can be used.

The idea behind data-preprocessing is that, before that training data is presented to the learning algorithm, several techniques can be applied to it in order to aid the learning algorithm in properly learning from the imbalanced data. This as mentioned, opposed to other methods such as introducing different misclassification costs within the classifier, is completely independent on the actual learner that will be used, the type of dataset involved or the way that specific learners output is interpreted.

The common method for the pre-processing is *resampling*, which involves reducing the imbalance in the data by either removing majority class sample points, or by magnifying the size of the minority class. This creates a new relatively balanced data set based on the original (imbalanced) one. There are multiple ways in achieving this. Some are more general, others target a specific sub-area of the imbalanced "challenge". Which (combination of) method(s) should be used is dependent on type of data, level of imbalance and what outcome is "desired". This choice, combined with the level to which the chosen pre-processing method should be used, can be quite a challenge.

Resampling

As mentioned above, resampling can come in either of two forms. One possibility is to remove majority class observations within the training dataset to restore the balance up to a certain level, which is referred to as *undersampling*. Alternatively, with *oversampling*, the size of the minority class is increased. Over- and undersampling both primarily address the problem of relative rarity, with the exception of some oversampling techniques aim generate new minority class points.

3.2.1 Random over-/undersampling

The first and most straightforward way of performing resampling is by doing it randomly. *Random undersampling* eliminates majority class sample points at random. *Random oversampling* exactly replicates minority class sample points at random,with replacement. Both until a desired majority to minority ratio is achieved. Both methods have obvious drawbacks: undersampling results in (potentially valuable) information loss about the characteristic concepts defining the majority class, while oversampling significantly increases the risk of overfitting. Furthermore, in the case of relative rarity, but not absolute rarity (sufficient minority class points but a very unbalanced class distribution) this results in a lot more data points, hence especially for algorithms that consider every data point separately (for instance the Support Vector Machine [28]) the computational requirements significantly increase.

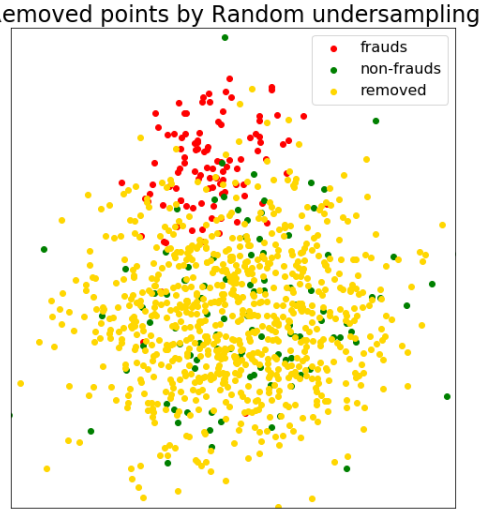

Figure 3.1: Result of performing random undersampling on a randomly generated dataset.

Key aspect of random undersampling, visualized in figure 3.1, is that it completely ignores any structure within the majority class. Therefore, apart from the overall loss of information, it runs the risk of randomly removing sample points that play a vital role in defining, and thereby retrieving the decision boundary between the classes. By that logic, it would be a more reasonable solution to, instead of randomly "eliminating" majority class points from your set, target sample points that lie within regions of your data set that can be seen as less important for reconstructing the underlying distribution of the data. On the other hand, sampling your points at random does bring some kind of generality, in the sense that it can be directly applied to whatever imbalanced data set is encountered.

With random oversampling, it is important to realize that absolutely no new information is added. It is simply a repetition of existing data. Not inly does this, as mentioned above, increases the risk of overfitting. This could also imply that in the case of a high level of imbalance, so a lot of added new points to restore this, each point of interest is replicated so many times that algorithms

focus on *specific configurations* of features in stead looking for ways to generalize. By that argument, an oversampling method that, in some structured manner, would be able to generate new (synthetic) data would be an improvement to random oversampling.

Another important component to consider is the similarity between the resampling methods described above, and other possible solutions to the data imbalance, such as introducing costs/weights in the data or adjusting the decision threshold. Intuitively, either random over- or undersampling (or a combination of both) up to a certain factor seems equivalent to increasing the cost or weight of each minority sample (i.e. the weight of each sample point) by the same factor. Literature is divided on this subject [29, 30, 31].

3.2.2 Data regions

The main drawback of performing undersampling at random is the risk of losing sample points that contain valuable information for the classification task. This, while intuitively clear, remains somewhat of a vague notion. To be able to formalize valuable information a little further, throughout literature, a dataset can be roughly categorized into four different regions [32, 33]:

1. *Safe points*: sample points that lie within the expected region of the predictor space, according to the underlying distribution.
2. *Redundant points*: sample points that lie far away from any decision bound and thus can be taken over by other (closer) examples in terms of reconstructing the location of the classes. In general the redundant points do not particularly harm classification, but they do result in increased computational costs.
3. *Borderline points*: sample points located' close to the decision boundary, i.e. points that are important, but unreliable as the slightest amount of noise within the predictor space can move them across the decision boundary, resulting in a deterioration of the classifier.
4. *Class-label noise*: points that lie on the 'wrong' side of the decision boundary, hence sample points belonging to a certain class that, due to noise in the underlying distribution, lie within the predictor subspace that should ideally be mapped to another class.

As the following section will show, multiple strategies exist which are designed to focus on one (or multiple) of these concepts.

3.3 Targeted undersampling

Instead of removing observations from the available training data at random, there are numerous of methods to do this in a more structured manner. This can directly be related to the four categories specified above; it would for instance be preferable to remove redundant points (very far away from any decision boundary) over safe points (which are important in reconstructing classification boundaries). In this case the difficulty classification would not increase (or even decrease) but computational costs are much less and, for instance, optimizing over accuracy will result in predictions that are less biased towards a certain class.

Removing specific sample points belonging to certain sub-areas can thus limit the information loss that occurs when removing points at random to overcome imbalance. On the other hand, it is not useful to do this case-specific as one does not want to lose the generality of the random resampling and, more important, it is not always clear where these areas lie within the feature space or if they even exists as clearly as one theoretically assumes. For this more generalized form of targeted resampling, there exist multiple techniques and ideas. Throughout this section, some relevant ones will be stated, together with which area they ideally should target and why they could be helpful in a specific case. The goal of targeted undersampling, and therefore of all of these methods, can be formulated as follows:

find a subset S from the majority class C_- that, combined with all the positive class sample points C_+, when used as training data, will help overcome the difficulties for traditional classifiers when learning on imbalanced data.

How to find the *optimal* of such subsets can be case-specific and may very well use multiple of the described methods.

3.3.1 Tomek Link removal

The method of *Tomek Link removal* is the first of the so-called *distance-based* resampling methods. It builds on the idea that, if two points belonging to different outcome classes lie very close together in the predictor space, it will probably result in a distortion in the decision boundary or in decrease in performance of the classifier. Formally, sample points x_1 and x_2 form a Tomek link [34] if:

- x_1 and x_2 belong to a different outcome class.
- The (Euclidean) distance in the predictor space between the two points is given by $d(x_1,x_2)$.
- For each other data point y in S: $d(x_1, y) > d(x_1, x_2)$ and $d(x_2, y) > d(x_1, x_2)$.

It is reasonable to assume either both points lie close, but on opposite sides of the decision border, or one of the points can be considered as class-label noise.

Algorithm 1 Tomek Link removal undersampling

Require: data $X = (x_1, ..., x_N)$ with labels $y = (y_1, ..., y_N) \in \{0, 1\}^N$

procedure TOMEK(X, y)
  for all $i : y_i = 1$ **do** ▷ Every point belonging to the minority class
    $x_j \leftarrow nn(x_i)$ ▷ select the nearest neighbor of point i
    if $x_i = nn(x_j)$ **and** $y_j = 0$ **then** ▷ Ensure i and j form a Tomek link (definition above)
      $X \leftarrow X \setminus \{x_j\}$
      $y \leftarrow y \setminus \{y_j\}$
  return X, y ▷ Return the undersampled set

After identifying all of such pairs, resampling can occur by either removing all pairs or, more commonly, only the majority class sample points that form a Tomek link with a minority class point (figure 3.2). The pseudo-code of this undersampling method is shown in algorithm 1. This can remove some unwanted negatives within the "positive" outcome space, or make the border a little bit clearer which should guide the learner to better recall scores. It is important to note that this technique will in no way restore the balance to a certain factor, as did the random undersampling. It therefore is often considered more of a *data cleaning* method instead of undersampling, but by definition it does classify.

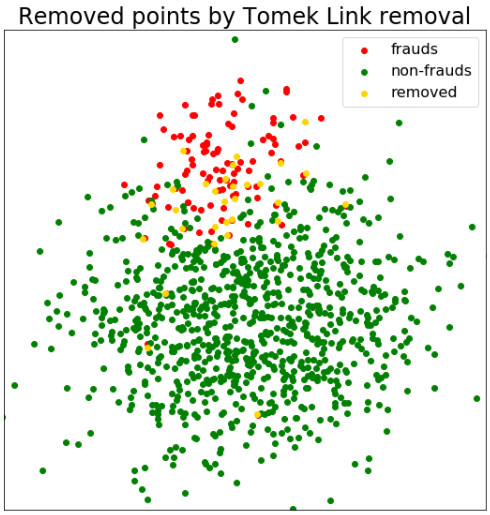

Figure 3.2: Result of performing Tomek Link removal on a randomly generated dataset.

3.3.2 Nearest-neighbor undersampling

The idea behind the Tomek link removal is simple, and can easily be justified by intuition, but

as mentioned in the previous section it does not restore the balance to a desired level. Especially in the case where the level of imbalance is very high, so there are much more negatives than positives in the training set, this method does not result in a large reduction of the majority class. This as, by definition, it removes at most the number positive sample points from the majority class. A class of methods that may be useful to overcome this are the *nearest-neighbor undersampling* methods.

Condensed Nearest Neighbor

Another commonly used distance-based undersampling method is *Condensed Nearest Neighbor* [35]. The key idea behind this method is as follows: form a subset S of the original (unbalanced) data set C such that all original samples are correctly classified by S when using a 1-Nearest Neighbor classification rule. Usually this subset is much smaller than the original data set, also resulting in less required computation time and storage capacity. Roughly, the way that such a subset is generated by the original CNN algorithm can be described as follows (pseudo-code displayed in algorithm 2):

1. At the beginning, let S be one randomly chosen sample point.
2. Select another point at random and classify using the 1-NN algorithm with S as reference. If the point is classified correctly according to its true label, set aside. Otherwise, include the point in subset S.
3. Repeat step 2 until each point is either set aside or included in subset S.
4. Continue by looping through all points that are set aside until either all points are in subset S, or one loop is completed without adding extra points to S
5. Discard leftover points, use S as desired subset

Now as discussed with Tomek link removal, this procedure can be used as undersampling by, in step 1, using the entire minority class C+ and one randomly chosen point from C-. This ensures that only sample points from the majority class are discarded by the CNN algorithm. An example of such output generated by the CNN undersampling method is shown on the left side of figure 3.3.

Algorithm 2 Condensed Nearest Neighbor

Require: data $X = (x_1, ..., x_N)$ with labels $y = (y_1, ..., y_N) \in \{0, 1\}^N$

procedure CNN(X, y)
  $S, D \leftarrow \emptyset$ ▷ Create a set for the stored and discarded points
  Pick $x \in X$ randomly
  $S \leftarrow S \cup \{x\}$, $D \leftarrow D \cup (X \setminus \{x\})$
  $added \leftarrow$ `True` ▷ Create boolean for continuing/stopping
  while $added =$ `True` **do**
    $added \leftarrow$ `False`
    for all $x_i \in D$ **do**
      $\bar{y_i} \leftarrow nnclass(xi, S)$ ▷ Classify point x_i by the 1-NN rule, with set S as reference
      if $\bar{y_i} \neq y_i$ **then**
        $S \leftarrow S \cup \{x_i\}$, $D \leftarrow D \setminus \{x_i\}$ ▷ Move misclassified points to set S
        $added \leftarrow$ `True`
    if $D = \emptyset$ **then**
      $added \leftarrow$ `False` ▷ Always stop when all points are in set S
  $y_S \leftarrow \{y_i | i : x_i \in S\}$
  return S, y_S

As pointed out in literature [34] this procedure introduces a certain amount of randomness, especially in the beginning of the algorithm. This can result in variations when repeating the algorithm,

and the fact that it will most probably not generate the smallest possible subset S which satisfies the desired condition.

Another possible drawback [36] is the sensitivity to class-label noise. This can be interpreted as follows: noisy data points usually have two common characteristics hindering the performance of this specific method of undersampling. First, data points that can be considered as noise tend to lie in the neighborhood of opposite class points, and therefore be misclassified by the 1-NN algorithm. This means the noisy observations, as well as their surroundings, will be retained by the CNN algorithm resulting in less data reduction. Second, by definition, noisy points do not represent the underlying class distribution very well so by both retaining them, and removing a lot of other points, this negative effect in correct classification for the learner is even amplified.

OSS, ENN and NCL

In attempt to overcome these drawbacks of the CNN data removal method, and to make these described distance based undersampling methods more suited for cases with extreme imbalance a combination is proposed that is often referred to as One-Sided Selection [32]. The OSS undersampling algorithm starts by using a CNN (-like) algorithm on only the negative sample points to remove a significant amount of majority class points. It then proceeds by removing all negative sample points participating in Tomek links. The term *One-Sided* in the title refers to the fact that only majority class points are removed from the training set and thus a representative subset of the negatives is combined with all the positives.

Another commonly used nearest-neighbor undersampling method is the Edited Nearest Neighbor [37]. The ENN considers a neighborhood of k sample points in determining whether a point should be retained or omitted. A point can be removed from the test set if it does not agree with the class of the majority of its k neighbors (algorithm 3). The idea is that this procedure removes both noise (by the argument made earlier), as well as points close to the theoretical decision border. This procedure shows similarities to the removal of Tomek links, but tends to result in more removed training data (middle of figure 3.3). For an even larger amount of data removal, the Repeated Edited Nearest Neighbor undersampling methods is proposed [38]. This method sequentially applies the original ENN method.

Algorithm 3 Edited Nearest Neighbor

Require: data $X = (x_1, ..., x_N)$ with labels $y = (y_1, ..., y_N) \in \{0, 1\}^N$

procedure ENN(X, y, k)
  for all $i : y_i = 0$ **do** ▷ Every point belonging to the majority class
    for j=1,k **do**
      $x_{nj} \leftarrow nn(x_i, j)$ ▷ Find the k nearest neighbors of point x_i
    $\bar{y_i} \leftarrow \frac{1}{k} \sum_{j=1}^{k} y_{nj}$
    if $\bar{y_i} \geq 0.5$ **then** ▷ Classify point i as majority vote over its k-nearest neighbors
      $X \leftarrow X \setminus \{x_i\}$
      $y \leftarrow y \setminus \{y_i\}$
  return X, y

On all of these discussed targeted undersampling methods, there are numerous of variations and tweaks. One particular adaptation to mention is the Neighborhood Cleaning rule [39]. As with OSS, the NCL focuses entirely on the majority class, making it a one-sided method. But, as opposed to OSS which focuses mostly on data reduction by using CNN, NCL aims more at data cleaning, as we saw with ENN and Tomek link removal. In fact, NCL is based on the former as it uses neighborhood areas of size k to identify 'noisy' data points.

The way it deals with such noisy data differs slightly. Again it starts by removing all majority class points that are wrongfully classified by the majority of its k neighbors, but then it proceeds

by also removing all negative samples in the neighborhood of each positive sample point that would be considered by the ENN method. The pseudo -code is shown in algorithm 4. This results in an increased level of data cleaning, as opposed to the ENN method, visualized on the right frame of figure 3.3. Of course, an obvious drawback can be the risk of overfitting, which is a trade-off that should be made at any time when performing data cleaning.

Algorithm 4 Neighborhood Cleaning rule

Require: data $X = (x_1, ..., x_N)$ with labels $y = (y_1, ..., y_N) \in \{0, 1\}^N$

procedure NCL(X, y)
  $A_1, A_2 \leftarrow \emptyset$
  $S \leftarrow (X, y)$
  for $i = 1, N$ **do**
    for j=1,3 **do**
      $x_{nj} \leftarrow nn(x_i, j)$ ▷ Find the 3 nearest neighbors of point x_i
    $\bar{y_i} \leftarrow \mathbb{1}\{y_{n1} + y_{n2} + y_{n3} > 1.5\}$
    if $\bar{y_i} \neq y_i$ **then** ▷ All misclassified points according to the 3-NN classifier
      if $y_i = 0$ **then**
        $A_1 \leftarrow A_1 \cup \{(x_i, y_i)\}$ ▷ Identical to ENN
      else
        for all $y_{nj} = 0$ **do** ▷ Select all majority class points in the neighborhood
          $A_2 \leftarrow A_2 \cup \{(x_{nj}, y_{nj})\}$
  $S \leftarrow S \setminus (A_1 \cup A_2)$
  return S

As mentioned earlier, these are by no means all possible methods of undersampling, but can be viewed as more of a framework on which most methods are based. And of course (as seen by OSS) one method absolutely does not excludes the others, and often properly chosen combinations achieve the best results.

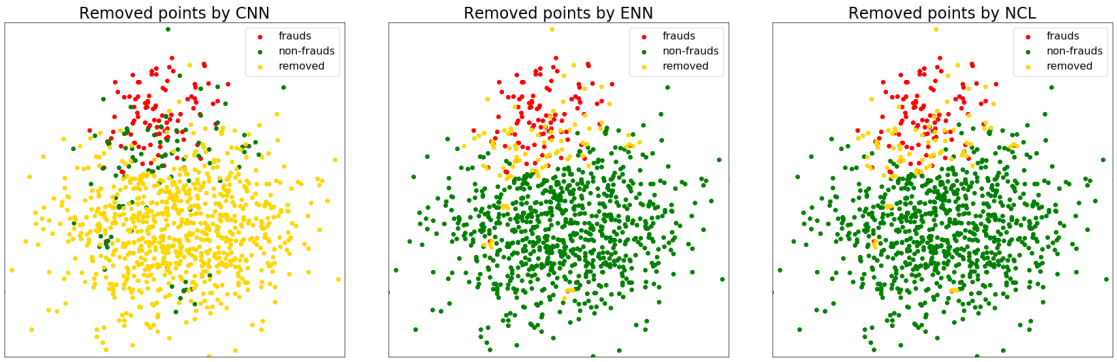

Figure 3.3: Result of performing random undersampling and Tomek Link removal on a randomly genearted dataset.

3.4 Informed oversampling: synthetic data generation

As mentioned in the first section, the main issue when learning with relative rarity is that the usual learning algorithms can't handle this very well. This is an important thing to always keep in mind

when performing oversampling. You do not actually obtain new information, but rather 'stretch' the currently available information in such a way that the algorithm can better use it. This, as opposed to all the undersampling methods, without actually ignoring or removing information from the data.

The already discussed method of random oversampling has an obvious risk, especially when the level of imbalance increases: exactly replicating sample points numerous times can make algorithms (especially the more complex ones) believe that certain specific configurations of the predictor space link directly to the outcome of interest. This of course does not generalize well to new and unseen data points. This risk of overfitting is the main point of attention when performing oversampling, and should be kept under consideration at any time. To force algorithms to focus more on the patterns in stead of individual configurations, certain techniques of **generating synthetic training data** are proposed.

3.4.1 Cluster-based oversampling: SMOTE

The first and probably the most commonly referenced method for generating synthetic new training data is called Synthetic Minority Over-sampling Technique [40]. This SMOTE method is inspired by earlier work on the recognition of handwriting where training data was generated by performing altering operations on existing data. The steps in the algorithm for generating a new synthetic sample point are as follows:

1. Consider a minority class sample
2. Compute its k nearest neighbors (in terms of Euclidean distance within the predictor space)
3. Pick one of these k points at random
4. For each feature:
    a) Compute the difference between the two points in terms of that feature
    b) For the synthetic sample point define the value of this feature as a random combination of the value of the two points

This process is done for each sample point (repeatedly) until the desired level of balance is achieved. To summarize, the algorithm generates new points within the *feature space* by connecting a point to one of its nearest neighbors and placing the new point somewhere on that line. Each 'new' sample point thus is generated as a **convex combination** of two existing points. The entire process of oversampling by SMOTE is shown in algorithm 5.

Algorithm 5 SMOTE

Require: data $X = (x_1, ..., x_N)$ with labels $y = (y_1, ..., y_N) \in \{0, 1\}^N$

procedure SMOTE(X, y, k, α)
  $S \leftarrow (X, y)$
  $S_+ \leftarrow \{(x_i, y_i) | y_i = 1\}$ ▷ Distinguish entire data set and data minority class data
  $N \leftarrow \alpha * |S| - |S_+|$ ▷ Amount of new points needed to obtain desired ratio α
  while $N \neq 0$ **do**
    for $i = 1, |S_+|$ **do**
      $r \leftarrow random(\{1, ..., k\})$ ▷ Pick random number from 1 to k
      $x_{nn} \leftarrow nn(x_i, S_+, r)$ ▷ Consider r-th nearest neighbor of point x_i within S_+
      $\delta \leftarrow random([0, 1])$ ▷ Pick random number between 0 and 1
      $x_{syn} = x_i + \delta(x_{nn} - x_i)$ ▷ Generate synthetic data point
      $S \leftarrow S \cup \{(x_{syn}, 1)\}$
      $N \leftarrow N - 1$
  return S

By the amount of randomness induced (both in selecting the neighboring point, as well as computing the new feature values) it is highly unlikely that this new training set contains to equal (or very similar) sample points, even when it is repeated for a long time. This reduces the risk of overfitting on the training set and should guide in enhancing the influence of the minority class in determining the decision boundary. On the other hand, (only) applying SMOTE can in some cases come with a downside. Most common risks are the increase of class overlap and overgeneralization [41].

By design, SMOTE generalizes the positive decision region as it generates new positives within existing ones, despite the possibility that there may be one or more majority class points in between them. This disregard for the majority class does come with a risk. Consider a set where there are one or more 'clusters' of positive with very small disjuncts (smaller than k), so a lot of sparsity. These points can either be noise or just very rare regions within the underlying class distribution. Now this means that, when looking at the set of k nearest neighbors, some (or even all) of them lie very far away in terms of feature-distance. If any of these points is selected in step 3 of the algorithm, it is very likely that this specific synthetic point lies in an area that previously contained only negatives. This, when being repeated, leads to overgeneralization as classes are being mixed. This could lead to a significant increase in false positives when learning on the training set. To address these issues, SMOTE could be combined with certain data cleaning techniques as described in the previous section. Alternatively, there are numerous variations on the original SMOTE algorithm.

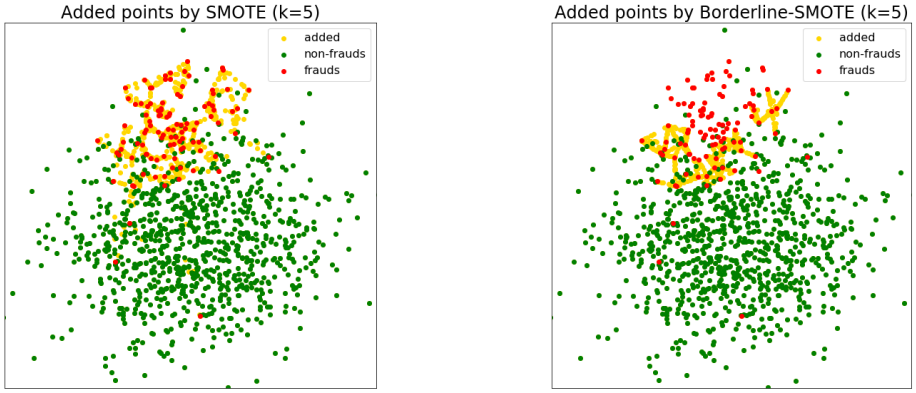

Figure 3.4: Result of performing SMOTE (algorithm 5) and borderline-SMOTE on a randomly generated dataset.

borderline-SMOTE

One, in particular, that removes some of the randomness of the original SMOTE by considering both classes in the neighborhoods is Borderline-SMOTE [42]. As the name suggests, this technique aims to focus at the borderline minority class sample points near the because they, as the authors argue, are most important for correct classification. Equivalent to the original algorithm, it uses the k (typically 5) nearest neighbors of a minority class point, but now while considering every remaining sample point in the training set. It uses this neighborhood to categorize the minority class point as follows:

a) If all of the k neighbors belong to the majority class, it is considered to be *noise*.

b) If less than half of its neighbors belong to the majority class, it is considered to be *safe*.

c) If at least half of it's neighbors belong to the majority class (but not all of them) , the point is labeled as *danger*.

Now all positives labeled danger must, according to the authors, be borderline samples from the minority class. This means that, by specifically oversampling those points, it should aid the learner in correctly retrieving the decision bound. This is done by the applying algorithm 5, but only for the points labeled danger .

Concluding remarks

All above described methods, both for under- and for oversampling, are considered established methods of restoring balance in a dataset. As mentioned, numerous tweaks and slight altercations are being designed and published every day. Most do however approach the problem in a similar way a the methods in this chapter. In the following chapter a method for pre-processing will be introduced that aims to do this a little different.

Chapter 4

Bump Hunting

The problem when facing imbalanced data, to summarize, is that throughout the entire feature space the underlying probability distribution of both classes are skewed. This on itself, as mentioned before, is by no means unusual nor problematic, but it resulted in an increased difficulty in the learning process and thereby a decreased performance of the classification task. The way it was previously attempted to overcome these difficulties was by specifically altering the distributions of the training sample that was presented to the learner by either downsampling the majority class, or upsampling the minority class, or a combination of both.

While this does a good job at rebalancing the two classes, and therefore reducing/removing the skewness of the two distributions, this does not tackle the underlying problem. This means that, when facing a different sample drawn from the same distribution within the same feature space, one would expect to find the same level of imbalance between the two classes. An alternative approach would thus be to, instead of specifically alter the effective distributions within your training sample, **redefine the feature space** from which instances are sampled in such a way that the resulting classes are more balanced.

An approach of trying to achieve this is through bump hunting, which will be described in this section. The bump hunting method does not find its origin within the imbalanced classification domain, so the section will start by describing the methods and theory involved with bump hunting. After this, the application to imbalanced learning will be further explored. Precisely this application of bump hunting, or subgroup discovery, to the setting of imbalanced classification has not yet been reported and hence is a new contribution of this work.

4.1 Formal definition

The original concept of bump hunting has applications beyond just a binary labeled setting [43, 44], but as this is the focus throughout this work, this section will be restricted to the two-class case. This means, just as in chapter 2, data of the following form will be used:

$$\{(X, y)\}^N \text{ where } y \in \{0, 1\} \text{ and } X = (x_1, x_2, ..., x_p). \tag{4.1}$$

Now bump hunting, as opposed to for instance resampling, focuses on the feature domain rather than specific sample points (X_k, y_k), let S denote the domain such that:

$$S = S_1 \times S_2 \times ... \times S_p \text{ where } \forall j \in \{1, ..., p\} : \ x_j \in S_j. \tag{4.2}$$

The goal of bump hunting is to find a set of subregions within this feature domain, in which the value of the target variable is relatively high compared to the value over the entire input space. Formally, for a given value function (usually the mean), a subregion $R \subset S$ is pursued such that

$$\bar{y}_R \gg \bar{y},$$

where y_R is the mean within region R and $\bar{y}$ is the global mean of target label y. In this application, these subregions, or 'rules', can be described as a union of hyper-rectangles or boxes each having the form

$$B = s_1 \times s_2 \times ... \times s_p \text{ where } s_j \subseteq S_j, \tag{4.3}$$

such that sample point $X = (x_1, ..., x_p)$ is considered to be in box B when:

$$X \in B \iff \bigcap_{j=1}^{p} (x_j \in s_j). \tag{4.4}$$

The reason that these subregions are called rules is that combined they form a specific set of rules to which new sample points can either qualify, or not qualify. An example of some of such rules, when being applied to credit card data can be:

$$\begin{gathered} 30 < \textit{AMOUNT} \\ 11{:}00 < \textit{TIME\_OF\_DAY} \\ \textit{RECIPIENT} \neq \textit{supermarket} \\ \textit{TIME\_OF\_DAY} < 23{:}00, \end{gathered}$$

where clearly each individual transaction either does or does not qualify to the rules. These rule sets are easily interpretable and can be traced back to human knowledge, meaning that they can also guide with the process of mapping reasons to a certain outcome and explaining why a certain outcome can be observed in terms of the given predictors. This extra level of **explainability** also speaks for using this method in for instance fraud detection.

The way in which these rules are constructed and combined is through a method called *covering* [45], which means that the same method of constructing individual boxes is applied sequentially to subsets of the dataset. The process starts by constructing box B_1 on the entire data set, resulting in a collection $\{(y_i, X_i) | X_i \in B_1\}$ which is a subset of the total amount of available training observations. The set consists of all pairs of feature vectors which fall within the constructed box and their corresponding class label (4.1). This subset is then removed from the entire data set, and a new box is constructed on the remaining sample points. This means that, after $K-1$ iterations, the K-th box is constructed on the set:

$$\left\{ (y_i, X_i) | X_i \notin \bigcup_{k=1}^{K-1} B_k \right\} \tag{4.5}$$

To know when this process should be stopped, two important quantities must be considered after constructing each box. First of all, the target mean within the newly constructed box, defined by

$$\bar{y}_K = \text{mean} \left(y_i | X_i \in B_K \text{ and } X_i \notin \bigcup_{k=1}^{K-1} B_k \right), \tag{4.6}$$

and the support of the data within the newly constructed box, defined by

$$\beta_K = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}\{X_i \in B_K \text{ and } X_i \notin \bigcup_{k=1}^{K-1} B_k\}. \tag{4.7}$$

As mentioned earlier, the final rule R is defined by the union of all individual boxes (4.3), which on itself has target mean and support.

4.2 Patient Rule Induction Method

The original method that will be used within this research for the induction of the rules is called Patient Rule Induction Method [45], or PRIM, and originally constructs each box in two different phases:

1. Top-down peeling, sequentially removing small subregions within a certain feature domain
2. Bottom-up pasting, successively expanding certain sub boxes.

4.2.1 Top-down peeling

In the peeling stage, small subboxes are removed from the initial box B until the support fall below a certain threshold. This is achieved through the following steps:

1. A collection of candidate boxes eligible for peeling is selected: $C(b)$
2. The box which yields the highest target value mean after removal is chosen: $b^* = \underset{b \in C(b)}{\arg\max}\, \hat{y}_{B-b}$
3. The current box is updated accordingly: $B \leftarrow B - b^*$
4. Steps 1-3 are repeated until support falls below threshold: $\beta_B = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}\{\mathbf{x}_i \in B\} \leq \beta_0$

Of course, the main point of attention that arises from these steps is how to select the set of candidate boxes $C(b)$. For a categorical feature, this is pretty straightforward. By definition, each of such features S_j had a finite number of possible values within its domain, which can all be peeled of, one by one, hence a finite number of subboxes

$$b_{jm} = \{X | x_j = s_{jm}\} \text{ where } s_{jm} \in S_j. \tag{4.8}$$

For a numerical feature, two candidate subboxes can be distinguished, each on one end of the input domain:

$$\begin{aligned} b_{j-} &= \{X | x_j < x_{j(\alpha)}\} \\ b_{j+} &= \{X | x_j > x_{j(1-\alpha)}\}. \end{aligned} \tag{4.9}$$

In equation (4.9), the values of $x_{j(\alpha)}$ and $x_{j(1-\alpha)}$ correspond to the α-th and the $(1-\alpha)$-th quantile of the values of x_j within the box that is currently considered, hence

$$\mathbb{P}(x_j < x_{j(\alpha)}) = \alpha \text{ and } \mathbb{P}(x_j > x_{j(1-\alpha)}) = 1 - \alpha. \tag{4.10}$$

Now the class $C(b)$ that is used in the peeling algorithm consists of all these subboxes (4.8)(4.9) on all different sub domains (4.2) combined.

Within this entire peeling procedure, two meta-parameters are involved:

1. The peeling fraction α that appears when defining eligible subboxes for numerical input (4.9), often referred to as the *degree of patience*.
2. The minimum support β_0 to guarantee a certain remaining amount of data within the final box.

Both parameters should be considered when defining the final rule set and can be determined either by statistical analysis, preference when applying peeling to a specific situation, or for instance a more pragmatic approach like cross validation. This second approach will be explored further in the following section

4.2.2 Bottom-up pasting

This second step of the original PRIM algorithm, especially in the case where the resulting rule sets are used as regions of peaked interest rather then acting as an approximate classifier for the target variable, is of less relevance. Nevertheless, for completeness, it will briefly be discussed anyway.

The idea behind the pasting procedure is as follows: the removal of subboxes in step 2 of the peeling algorithm is done in a very greedy way, in the sense that each split is made by only considering the best choice at that specific moment disregarding possible future choices. This means that it is possible that a suboptimal final box is reached that can be improved by readjusting some of its boundaries.

The pasting algorithm works similar as the peeling algorithm from the previous section, but it starts with the final (smallest) box. It, again, selects a set of candidate boxes similarly to equations (4.8) and (4.9), but now by either selecting an input that is not represented by the current box or by extending the upper/lower bound of the domain for categorical or numerical features, respectively. From all of these candidate boxes, the box b^* that results in the largest increase in target mean (if existent) is then selected. The new, larger, box is formed by

$$B \leftarrow B \cup b^* \tag{4.11}$$

This process is repeated until none of the candidate boxes in the candidate set results in an increase of $\bar{y}_{B+b^*}$.

4.3 Application to Classification

As mentioned earlier, rule induction or bump hunting has previously not been linked to data preprocessing, especially in the case of classification on imbalanced data. Instead, typical applications could be direct classification of new data through the rule set, or for instance in the situation of credit scoring when applying for a loan [46]. The resulting box implies a high risk area, so when an applicant falls within this region, extra investigation on whether or not a loan should be granted could be considered. This means that, as opposed to fraud detection, an accurate prediction of the ceases of interest is of much less importance, as long as little of the higher risk cases are missed.

Now for the case of imbalanced learning, bump hunting merely is one stage in the entire classification process, as observed with for instance resampling. If assumed that the box induction is done correctly, and that all data, present and future, is drawn from the same distribution, this would imply that when restricting to the domain within the final box, a new and unseen observation has a much larger probability of belonging to the minority outcome class. This would as mentioned earlier, as opposed to resampling the training data, directly **change the underlying skewness** of the class distributions. Thus when restricting the training process to just this subspace of the input data, an algorithm would theoretically have much less difficulty with properly handling the data even without the need of resampling. This concept is being examined a little more thorough in a further section.

This initial idea of preparing and classifying can be seen as a two-staged model for training a classifier on the available, highly imbalanced data set:

Tr1. Run the subgroup discovery algorithm (PRIM) on the entire data set to create one/multiple sub box(es) (4.3).

Tr2. Discard all training samples that do not fall within these rule sets corresponding to the box(es) generated in step 1. Train classifiers on the remaining data within each of the subregions of the initial training set.

A two-dimensional visualization of this training process, when one box is produced in Tr1, is displayed in figure 4.1. It may sound a bit trivial, but it is important to realize that step Tr2 implies that, when more then one, non overlapping, subregions are obtained (4.3) from the initial feature domain (4.2) by step Tr1, on each of these regions different classifiers with completely different hyper-parameters can be trained. It would even be possible to use different learners, each with their own strengths and weaknesses, on the different distinct subregions of interest. This can be another advantage of this particular method of handling imbalanced data as opposed to resampling.

When presented with unseen data point (X_{te}, y_{te}) with unknown class label y_{te}, again a two-staged testing (classification) process can be distinguished:

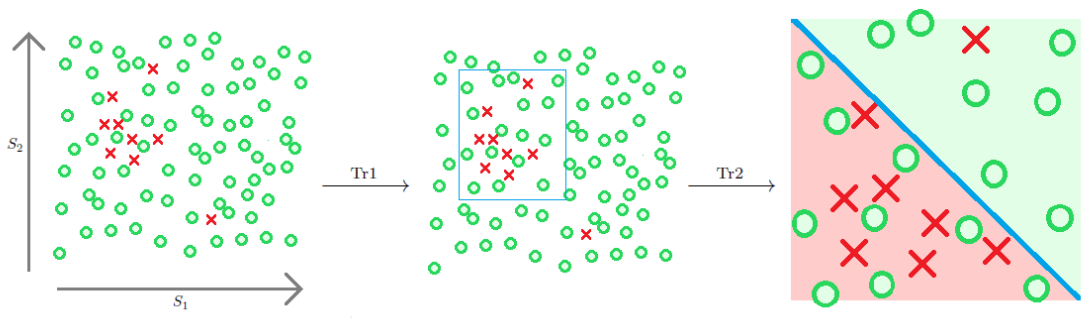

Figure 4.1: Visualizition of the training process when combining bump hunting with a classifier.

Te1. Observe whether X_{te} falls within (4.4) one of the subboxes generated in training step Tr1. If not, immediately classify this new point as negative.

Te2. Otherwise, classify this new data point according to the specific classifier trained in Tr2 that corresponds to the subbox that the point belongs to.

Again, this process can be systematically visualized, shown in figure 4.2.

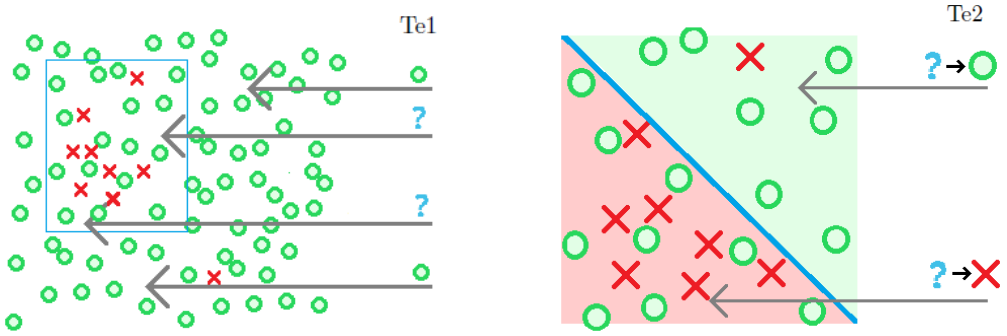

Figure 4.2: Visualization of the two-staged classification process, as if four new points (gray arrows) would be classified by the learner shown in figure 4.1

This process automatically implies that all (potentially noisy) new data points that do not fall within a certain subbox resulting from Tr1 will automatically be (mis)classified as belonging to the negative class. This speaks in favor of carefully and conservatively constructing the boxes and defining their characteristics. On the other hand, one could argue that, especially in an environment that by itself is harder to learn, such points would almost never be retrieved by any method available, unless a great number of false positives would be accepted on forehand. Nevertheless, defining the proper way of constructing the 'interesting' subboxes, should be considered the main focus of attention. However, there are more things to considered when attempting to fit the method of bump hunting in the imbalanced classification pipeline.

4.3.1 Adaptations to PRIM

The original PRIM algorithm features two meta-parameters: the peeling fraction α and the minimal support β_0. Before going into detail about how the value of these two could be determined, there is another important consideration to be made. That is, instead of restricting the size of each peel

(4.9) and the amount of data left in each box (4.7) in the construction process, reconsidering the idea behind each peel and why a particular subbox out of the candidate set $C(b)$ should be selected for removal.

Target function

The whole principle behind PRIM is to find a subregion within which the target mean is significantly higher. As a result, as observed in step 2 of the peeling process, out of all candidate subboxes the one with the largest target mean after a specific peel is selected. This is a very reasonable process when applied to a somewhat balanced domain; as long as a split removes more negatives than positives, it is a sensible choice to make.

But in the current case, as sketched above, it may be much more sensible to remove certain areas of the feature space that, although they may contain less majority class than others, contain no minority class points or positives. In that case purely maximizing the target mean turns out to be a worse choice. It is extremely important to realize what the goal of subgroup discovery in this frame work is:

> *specifying certain regions within the feature space such that, when classifiers are trained exclusively on data falling within these regions as opposed to the entire data set, it improves their performance.*

Also, as mentioned earlier within this work, it is often the case that falsely classifying a positive as being a negative is much more costly. This too is an important argument in favor of altering the target function in such a way that it removes subboxes containing positives with more caution.

4.3.2 Validation structure

As with regular machine learning, in subgroup discovery the risk of overfitting should always be taken under consideration. Continuously peeling of small parts of the feature space (4.8)(4.9) with the sole purpose of raising the value of the chosen target function may lead to individual boxes each with a very high target function value (4.6), but without controlling the support of each box (4.7), one may as well end up with a large number of boxes all enclosing precisely one single sample point belonging to the target class. It is evident that this probably does not generalize well to a similar set, but with different sample points.

To overcome this, as mentioned earlier, the support control parameter β_0 is included in the peeling method. This parameter is responsible for, on one hand determining how specific the box(es) are allowed to capture the finer structures on the available test data, but on the other hand constraint the model into going too much into specific details of the test set and therefore losing its generalization properties. This *bias-variance like* trade off means that, as with model complexity in the general learning case, this support parameter should be set with caution.

In the original paper [45], this is accomplished through a cross-validation like approach. The entire training set, on which the boxes should be built, is divided in a train and validation part. The peeling procedure is performed as usual, with a fixed very small β_0. Of course, by definition the peeling algorithm proceeds by sequentially increasing the target mean (4.6), thereby reducing the remaining support (4.7). As mentioned above, it is expected that this process will enter the overfitting-domain. This means that eventually, when the same set of rules is applies to a new set, the value of the target function will cease on to increase, or even decrease. To determine this point, at each peel, the target output of each box (4.6) is calculated on the remaining partition. This process, if the data indeed is generated by the same underlying distribution function as assumed, will produce an unbiased estimator for the optimal value of the target function.

As in this case the overall objective is not maximizing the target function, this validation process is altered to make it better suited for fitting in the classification pipeline. A schematic representation of the new proposed validation structure is shown in figure 4.3.

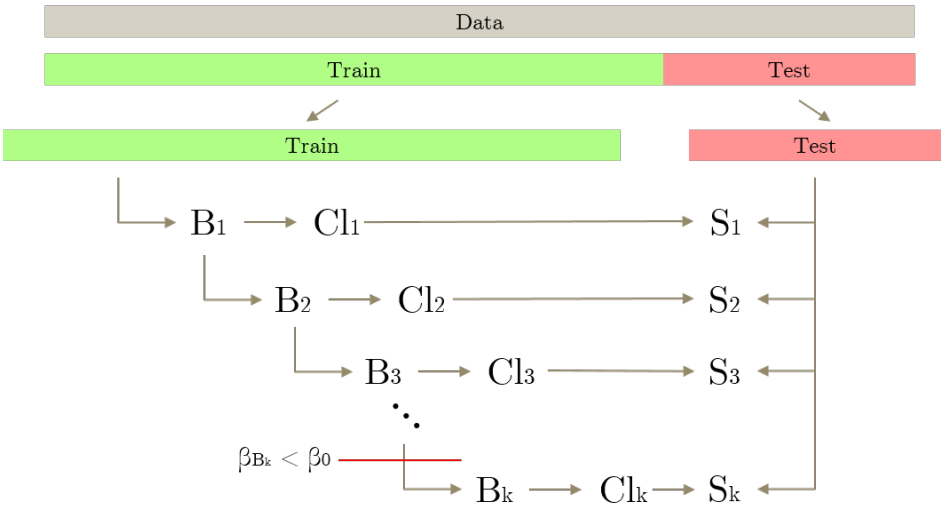

Figure 4.3: The newly proposed validation structure introduced in this research. On the training partition through the original way boxes B_i are selected and classifiers CL_i are trained on the corresponding set of observations. Then assign a *learnability score* S_i to each box, the classifier is evaluated on the test partition according to the procedure described in section 4.3.

Within this process, two main components can be distinguished. On one hand, there is the regular peeling and thereby creation of subboxes, denoted by the B_i's in figure 4.3. This is done in the same manner as described in the PRIM section, thus by greedily maximizing the target function on the training partition of the data set at each step. The way this procedure varies from the original PRIM algorithm and its corresponding validation process, is the part that comes after each peel. In this validation approach for each subbox B_i, a classifier is trained on the remaining points that correspond to the rules that define B_i. Using these classifiers Cl_i, a score S_i is computed by using the testing method on the remaining partition of the data, described in the previous section, so both steps Te1 and Te2 (figure 4.2). This process stops after k steps when the support of box B_k falls below the user defined parameter β_0. The value of this parameter could, and should, be taken quite small.

Theoretically, this process should provide the subbox which is an unbiased estimator for the subset of the data that is **best learnable** by the chosen classification algorithm used in the Cl_i. This statement automatically raises two important questions. First of all, best learnable is a rather vague statement. What makes a subset best learnable, or more important, which scores S_i should be computed at each iteration? This question will be further examined in chapter 6, as it features more than just selecting the proper performance metric, but also how to combine multiple boxes, and how to incorporate the earlier mentioned covering in this framework. The second question is a little more straight-forward: which classification algorithm should be used for training each Cl_i?

Of course, the classifier that is used must be representative for the classifier that will be used within the final predictor, otherwise there is absolutely no guarantee that the boxes that are selected will be close to the best choice. On the other hand, when selecting a very thorough and computationally intensive classifier, the process will probably require a lot of computational time and power. As a first test, the method shown in figure 4.3 is tested with both the Logistic Regression classifier and the Random Forest classifier where the number of trees is capped at ten. For this test, a peeling fraction of 0.05 is used. The time needed for training the classifiers Cl_i at each peel i are plotted in figure 4.4.

As expected, due to the exponential decrease in remaining data points on which each classifier is being trained, the training time at each iteration decreases rapidly. For the logistic regression and

Figure 4.4: Time required for training a classifier on each subbox resulting from a peel according to the process from figure 4.3. Red lines indicate when the process passed the half of the total time.

random forest, half of the total training time is reached at just 13 and 10 peels, respectively (indicated by the red lines in figure 4.4). If the assumption that this process of peeling away the less interesting regions of the feature space leads to improved performance for classifiers, it could therefore be sensible to skip the first few classifiers and start the scoring after a certain number of iterations. This would not only, as figure 4.4 indicates, reduce the time that the entire process consumes, but also allows the user to use more sophisticated and intensive classifiers or hyper-parameter configurations, as each classifier effectively has to be trained on a much smaller set of observations. This would of course only work if it is guaranteed that the chosen score function in non-decreasing (preferably increasing) over the first part of the validation process.

4.4 Bump hunting versus resampling

Previously it has been stated briefly that, as opposed to resampling, bump hunting should be able to rebalance the distributions of both the training set on which the actual method is applied, and all unseen data that will be encountered in the future. In the following section, this concept, together with the one-to-one comparison with resampling, will be explored further. For visualization and better interpretation of the actual implications on the data of each direction of methods, the credit card fraud dataset will be used.

4.4.1 Binary classification

Throughout this research, the main area of focus is binary classification: can one predict the occurrence of fraud (or non-fraud) based on a given 'configuration' of the data. This can be formalized and applied to the statistical learning setting by using Bayes' Theorem of conditional probabilities:

$$\mathbb{P}(y|X) = \frac{\mathbb{P}(X|y)\mathbb{P}(y)}{\mathbb{P}(X)}. \tag{4.12}$$

In the classification setting this quantity, or the *posterior probability*, corresponds to the probability that an observation belongs to a certain label y, given data X. For each *discriminative* classification algorithm described, this is the property that is attempted to model. Note that X is continuous while y is discrete. As equation (4.12) shows, the posterior is defined by three different components. On one

hand there are the two individual *prior probabilities* $\mathbb{P}(X)$ and $\mathbb{P}(y)$, corresponding to the distributions of either the data or the classes. On the other hand there is the *conditional probability* $\mathbb{P}(X|y)$ which captures the distribution of the data given that it belongs to a certain class.

In the case of binary classification (for instance fraud detection), y can be either one or zero. Including this in equation (4.12) yields the probability of finding a fraud ($y = 1$) given a certain feature configuration $X = x \in \mathcal{X}$:

$$\mathbb{P}(1|X = x) = \frac{\mathbb{P}(X = x|1)\mathbb{P}(1)}{\mathbb{P}(X = x)}. \tag{4.13}$$

For convenience in notation $\mathbb{P}(X = x)$ will be abbreviated to $\mathbb{P}(x)$. Because y ranges over a limited number of discrete values, the class probability of X can be further rewritten as:

$$\mathbb{P}(x) = \sum_y \mathbb{P}(x|y)\mathbb{P}(y) = \mathbb{P}(x|1)\mathbb{P}(1) + \mathbb{P}(x|0)\mathbb{P}(0). \tag{4.14}$$

Combining equations (4.13) and (4.14) yields

$$\mathbb{P}(1|x) = \frac{\mathbb{P}(x|1)\mathbb{P}(1)}{\mathbb{P}(x|1)\mathbb{P}(1) + \mathbb{P}(x|0)\mathbb{P}(0)}. \tag{4.15}$$

Imbalanced data

It is noted before that imbalance of data, or the rarity of one of the classes, actually implies that individual class priors are skewed. To put this in the same context as above, this means that

$$\mathbb{P}(0) \gg \mathbb{P}(1), \tag{4.16}$$

or simply the fact that one class is more rare than the other. It is important to know that this does not affect the conditional probabilities $\mathbb{P}(x|1)$ and $\mathbb{P}(x|0)$, as they are defined within a specific class. Combining this and (4.16) with equation (4.15) this immediately implies that, unless the data finds itself in a region where the conditional probability of the fraud class is highly dominant, the overall posterior probability of observing an observation belonging to the 0 class is much higher.

To visualize this, the left side of figure 4.5 shows a histogram of an individual feature (V14 to be specific) over its domain. It requires a very good eye for detail to be able to distinguish the small red

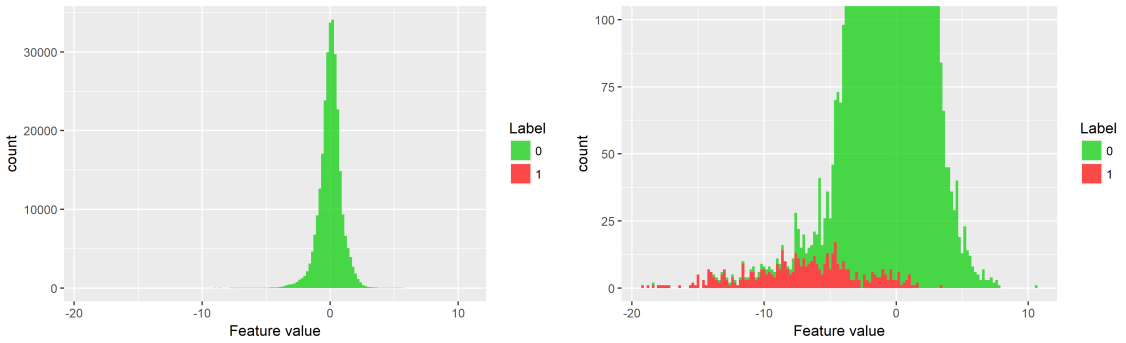

Figure 4.5: Count plots of the both classes over the feature range.

spots in the lower region of the feature space. To be able to observe both of the outcome classes, on the right side the y-axis is cut off at 100 (about .28% of the original range). This figure confirms that, although on most of the area the actual conditional class probability of the fraud class dominates (visualized further in figure 5.8) even here the actual number of observed sample points belonging to class 0 is higher.

4.4.2 Random undersampling

So what is the influence of randomly removing majority class sample points from the available pool of data on the situation sketched above? The result on the same feature is shown in figure 4.6. Here it clearly shows that, while the fraud class remains identical, the overall balance between the two classes is restored in the sense that the total amount of instances belonging to both outcome classes is the same.

The effect can be explained by again using the statistical framework. First note that both class priors can easily be estimated by using the proportion of sample belonging to the class:

$$\mathbb{P}(i) = \frac{N_i}{N} \quad \forall i \in \{0, 1\}, \tag{4.17}$$

where N_i represents the total number of observations with label i. Undersampling, denoted by * from here forward, means removing points from class 0, hence $N_0^* < N_0$. By using equation (4.17) in combination with (4.16) this results in:

$$\frac{\mathbb{P}^*(0)}{\mathbb{P}^*(1)} = \frac{N_0^*/N^*}{N_1/N^*} = 1 \ll \frac{\mathbb{P}(0)}{\mathbb{P}(1)}, \tag{4.18}$$

hence their difference is reduced greatly. Of course this applies to the case where the dataset is rebalanced completely (so $N_0^* = N_1$), but for the case where the balance is equalized up to a certain factor the inequality remains valid.

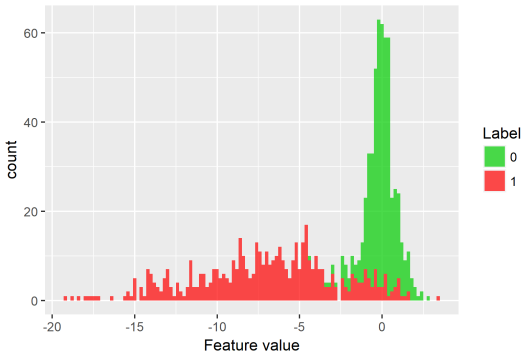

Figure 4.6: Distribution of the data after performing random undersampling.

On the other hand, by randomly selecting each point for removal, it can be assumed that the conditional data distribution within the majority class remains unchanged. Formally this means that

$$\mathbb{P}^*(x|0) = \mathbb{P}(x|0). \tag{4.19}$$

To verify this assumption empirically, a density plot of both dataset is shown in figure 4.7. The plot clearly shows that, as intended, in both cases the conditional constructions within both classes appear (relatively) unchanged.

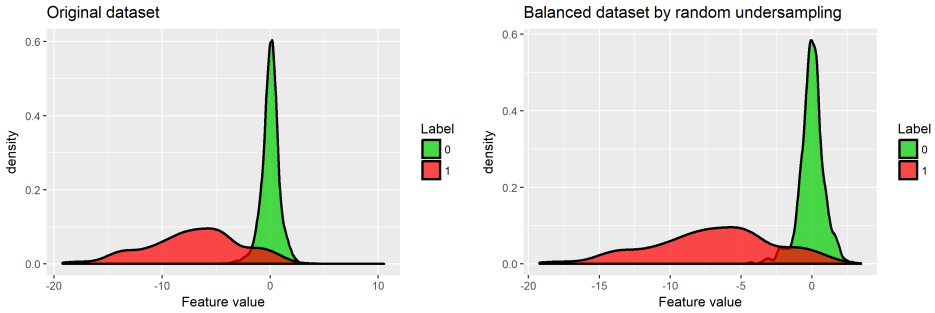

Figure 4.7: Density plots of one feature of the credit card fraud dataset before (left) and after (right) performing random undersampling

Generalizing beyond training data

It was previously concluded that the issue with between-class imbalance was that, while on itself it should not be a problem as long as there are sufficient observations available in both classes, the

difficulties in learning arise because algorithms **can't effectively handle** imbalance. For that reason the above sketched situation appears to be ideal: both classes are properly balanced when presented to the learner and one merely has to pick a classification algorithm that models the posterior such as the logistic regression (see equations (2.1) and (2.2)) and according to equation (4.15) combined with (4.18) and (4.19) the minority class will be much less overlooked and more dominant within the class label predictions. However, this is not immediately the case.

The original goal of machine learning is to extend the knowledge within a certain available pool of data to new, unencountered and unlabeled data and thereby generalize this beyond the training set. This can be mimicked by dividing the dataset into a train and test partition, and leave the test set out of the entire learning process. This allows the user to independently validate the methods used. One of the main assumptions beyond statistical learning is that observations are **iid** in the sense that it can be expected that past patterns will represent future instances. This can be formalized as

$$\mathbb{P}_{tr}(y = i|X) \overset{d}{=} \mathbb{P}_{te}(y = i|X) \ \ \forall i \in \{0, 1\}, \tag{4.20}$$

where *tr* and *te* represent the train and test partitions, respectively. This implies that, if the learner is able to accurately estimate the posterior over the training set, this can **directly** be used to predict $\mathbb{P}_{te}(i|x_n)$ for an unseen observation x_n.

However, by undersampling the training data to balance both outcome classes, the equality in equation (4.20) does not hold. To see this, consider the original and the 'new' posterior of the fraud class over the training set, after the same procedure as in the previous section is applied:

$$\begin{aligned} \mathbb{P}_{tr}(1|x) &= \frac{\mathbb{P}_{tr}(x|1)\mathbb{P}_{tr}(1)}{\mathbb{P}_{tr}(x|1)\mathbb{P}_{tr}(1) + \mathbb{P}_{tr}(x|0)\mathbb{P}_{tr}(0)} \\ \mathbb{P}^*_{tr}(1|x) &= \frac{\mathbb{P}_{tr}(x|1)\mathbb{P}^*_{tr}(1)}{\mathbb{P}_{tr}(x|1)\mathbb{P}^*_{tr}(1) + \mathbb{P}^*_{tr}(x|0)\mathbb{P}^*_{tr}(0)} = \frac{\mathbb{P}_{tr}(x|1)}{\mathbb{P}_{tr}(x|1) + \alpha \cdot \mathbb{P}_{tr}(x|0)}, \end{aligned} \tag{4.21}$$

where for the the last equality (4.19) is used. Here $\alpha := \mathbb{P}^*_{tr}(0)/\mathbb{P}^*_{tr}(1)$ represents the desired new (im)balance ratio, which was previously set to one in (4.18). As after undersampling these two clearly are not equal, the equality in (4.20) does not hold, hence

$$\mathbb{P}^*_{tr}(1|X) \overset{d}{\neq} \mathbb{P}_{te}(1|X). \tag{4.22}$$

Evidently identical arguments apply to the majority class case.

This inequality implies that the commonly followed procedure of undersampling the dataset, training a classifier on this subset of the original dataset and use this to label new observations systematically **over- and underestimates** the true underlying class-posteriors.

Overestimation

The fact that undersampling thus, according to the above computations, systematically overestimates the conditional probability of an unencountered observation belonging to the positive minority (fraud) class does not have to be a problem and may even be the goal of performing (random) undersampling. After all it has previously been stated that often misclassification costs are highly skewed, and thereby somewhat overestimating the chance of observing a hit can be interpreted as a case of *better safe than sorry.*

Therefore it may be useful to investigate and quantify the amount of overestimation in this *theoretical framework* as concluded in (4.22). For simplicity in notation lets introduce the following abbreviations:

$$p_+ := \mathbb{P}_{tr}(1|x), \quad p_- = \mathbb{P}_{tr}(0|x), \text{ and } \ \alpha_0 = \mathbb{P}_{tr}(0)/\mathbb{P}_{tr}(1). \tag{4.23}$$

Then equation (4.21) becomes

$$\begin{aligned} p_+ &= \frac{\mathbb{P}_{tr}(x|1)}{\mathbb{P}_{tr}(x|1) + \alpha_0 \cdot \mathbb{P}_{tr}(x|0)} \\ p_+^* &= \frac{\mathbb{P}_{tr}(x|1)}{\mathbb{P}_{tr}(x|1) + \alpha \cdot \mathbb{P}_{tr}(x|0)}. \end{aligned} \tag{4.24}$$

Rewriting and combining yields:

$$\left.\begin{aligned} p_+ &= \tfrac{1}{1+\alpha_0 \cdot \frac{\mathbb{P}_{tr}(x|0)}{\mathbb{P}_{tr}(x|1)}} \\ p_+^* &= \tfrac{1}{1+\alpha \cdot \frac{\mathbb{P}_{tr}(x|0)}{\mathbb{P}_{tr}(x|1)}} \end{aligned}\right\} p_+^* = \frac{p_+}{p_+ + \frac{\alpha}{\alpha_0}(1-p_+)} \tag{4.25}$$

Obviously, inserting $\alpha = \alpha_0$ (no undersampling) into (4.25) yields the original probability and, because undersampling results in an increase in imbalance ($\alpha \leq \alpha_0$), this shows the overestimation of the minority class. Evidently, as $p_- = 1 - p_+$, this immediately implies an underestimation of the majority class. Both α and α_0 depend on the two class priors of the two outcome classes. These can, similar to (4.18), be estimated by using the proportions of the two classes within the training set:

$$\begin{aligned} \frac{\alpha}{\alpha_0} &= \frac{\mathbb{P}^*(0)}{\mathbb{P}^*(1)} \cdot \frac{\mathbb{P}(1)}{\mathbb{P}(0)} \\ &= \frac{N_0^*/N^*}{N_1/N^*} \cdot \frac{N_1/N}{N_0/N} = \frac{N_0^*}{N_0}. \end{aligned} \tag{4.26}$$

This means that, in theoretical framework stated above, the level over overestimation of the fraud class posterior probability directly depends on the amount of majority class observation removed. This also implies that, for less rigorous methods previously denoted as *data cleaning*, which remove only a small number of points which presumably hinder the classifiers performance the most, this level of overestimation is relatively small. Especially with large datasets such as one would expect to encounter when detecting credit card fraud. Of course in this case the assumption in (4.19) does not strictly hold, as points are no more being removed at random, but due to the small amount it is not expected that this distribution changes drastically.

On the other hand, with methods that forcibly rebalance the dataset completely (such that $N_0^* = N_1$) this level of overestimation can be significantly higher.

4.4.3 Bump hunting

To be able to compare these theoretical results to the newly proposed method of using bump hunting as a method for imbalanced classification, it must be analyzed in the same framework. This means that, as stated in equation (4.12), the goal is to predict the posterior probabilities

$$\mathbb{P}(1|X) \quad \text{and} \quad \mathbb{P}(0|X). \tag{4.27}$$

The main difference in this setting is that each observation within the dataset X can either belong to a box ($X \in B$) or not belong to a box ($X \notin B$). Conditional on these two events, (4.27) can be rewritten as

$$\begin{aligned} \mathbb{P}(1|X) &= \mathbb{P}(1|X, X \in B) \cdot \mathbb{P}(X \in B) + \mathbb{P}(1|X, X \notin B) \cdot \mathbb{P}(X \notin B) \\ \mathbb{P}(0|X) &= \mathbb{P}(0|X, X \in B) \cdot \mathbb{P}(X \in B) + \mathbb{P}(0|X, X \notin B) \cdot \mathbb{P}(X \notin B). \end{aligned} \tag{4.28}$$

Thus far no assumptions have been made. The only difference is that the previous setting featured the entire data-space $\mathcal{X}$, where this now has been separated into two disjoint sets $\mathcal{X}_i$ and $\mathcal{X}_o$ for the part that is either in or out of the constructed subboxes, respectively. Just as described in section

2.1, by finding two mappings $f^i : \mathcal{X}_i \to \{0,1\}$ and $f^o : \mathcal{X}_o \to \{0,1\}$ one can construct a combined mapping

$$f : \mathcal{X}_i \cup \mathcal{X}_o \to \{0,1\} \text{ where } f(x) = f_i(x) \cdot \mathbb{1}(x \in B) + f_o(x) \cdot \mathbb{1}(x \notin B) \tag{4.29}$$

which serves as a predictor over the entire feature space.

In a normal setting, this is rather a questionable thing to do. If one would randomly split the feature space in two pieces, train two classifiers and combine them through (4.29), as each of the two classification algorithm receives less of the total available pool of training data, the overall performance can be assumed to be reduced. In this case however, as the devision is obviously not random, this should not apply.

By following the procedure described in chapter 4, it should be the case that for the data region within the subboxes, the level of imbalance is greatly reduced. Evidently this immediately implies that the level of imbalance is even higher for the domain outside these boxes, or $\mathcal{X}_o$. Together with the knowledge from section 3.1, this means that it is expected that predictor f^i performs much better than a predictor trained over the entire imbalanced dataset, especially on the minority class. Consequently, when using a predictor of the form defined in equation (4.29), the performance on the class of interest should be improved as opposed to a classifier trained over the entire data space $\mathcal{X}$.

Due to the fact that the level of imbalance is even higher on $\mathcal{X}_o$, together with the assumption that these regions overall contain a little amount of minority class observations, it is expected that the second classifier f^o performs poorly. To not run the risk that by including this predictor, the total number of false positives increases significantly, this predictor is discarded and, as was previously indicated in the testing procedure in section 4.3, all observations that fall within $\mathcal{X}_o$ are automatically classified as non-fraud. Formally, this implies

$$\mathbb{P}(1|X, X \notin B) = 0 \text{ and } \mathbb{P}(0|X, X \notin B) = 1. \tag{4.30}$$

Through empirical testing it is confirmed that indeed this benefits the overall performance.

Again, by considering a specific observation x, combining (4.28) and (4.30) yields

$$\begin{aligned} \mathbb{P}(1|x) &= \mathbb{P}(1|x, x \in B) \cdot \mathbb{1}(x \in B) \\ \mathbb{P}(0|x) &= \mathbb{P}(0|x, x \in B) \cdot \mathbb{1}(x \in B) + 1 \cdot \mathbb{1}(x \notin B). \end{aligned} \tag{4.31}$$

Here, the indicators appeared as, at all time during the training and testing phase, it is known whether a certain observation either does or does not belong to a box.

One of the key elements of using bump hunting in the imbalanced case is that algorithms are only trained on sample points within a certain box. The means that, instead of attempting to reconstruct the overall posterior probabilities in (4.27), this methods aims to learn both

$$\mathbb{P}(1|x, x \in B) \quad \text{and} \quad \mathbb{P}(0|x, x \in B), \tag{4.32}$$

Next step is to further investigate this situation. Again using Bayes' Theorem yields

$$\begin{aligned} \mathbb{P}(1|x, x \in B) &= \frac{\mathbb{P}(x|1, x \in B)\mathbb{P}(1|x \in B)}{\mathbb{P}(x|x \in B)} \\ &= \frac{\mathbb{P}(x|1, x \in B)\mathbb{P}(1|x \in B)}{\mathbb{P}(x|1, x \in B)\mathbb{P}(1|x \in B) + \mathbb{P}(x|0, x \in B)\mathbb{P}(0|x \in B)} \\ &= \frac{\mathbb{P}(x|1, x \in B)}{\mathbb{P}(x|1, x \in B) + \alpha_B \cdot \mathbb{P}(x|0, x \in B)}. \end{aligned} \tag{4.33}$$

Here α_B represents the **within-box imbalance**, which can be approximated by the proportion of training samples belonging to both classes, but within the box:

$$\alpha_B = \frac{\sum_{i=1}^N \mathbb{1}(X_i \in B)\mathbb{1}(y_i = 0)}{\sum_{i=1}^N \mathbb{1}(X_i \in B)\mathbb{1}(y_i = 1)} =: \frac{N_{B,0}}{N_{B,1}}. \tag{4.34}$$

Now by definition of the PRIM algorithm, as it optimizes over the within box mean, it can be assumed that

$$\alpha_B < \alpha_0, \tag{4.35}$$

hence the within box fraud ratio is higher.

Generalizing beyond training data

The above inequality in equation (4.35) implies that again, as with undersampling, a more balanced training set to learn the classifier on is obtained. As expected from the theoretical part about imbalanced classification, this should then result in improved performance, especially on the minority class. However, there is a fundamental difference between this method and undersampling the training set. To see this, it is again important to investigate what happens when the newly acquired training set is generalized to new and unseen observations.

First of all, it again has to be assumed that both present and future observations and their resulting outcome are drawn from the same distribution, hence the iid assumption from (4.20) still holds. This immediately implies that, both for the train and test set all components in equation (4.33) are equally distributed, together with both the imbalance ratios. From this it can immediately be concluded that learning $\mathbb{P}(1|x, x \in B)$ over the training set and using this to estimate future instances does **not** introduce a systematical error.

To show this empirically, together with the assumption the decrease in imbalance ratio is preserved when a box is evaluated on an unobserved dataset, the procedure of train/test partitioning can be used again. Consider the following steps: first a training partition is being taken at random. Then the PRIM algorithm is used to construct a box on the training set in which the ratio of minority to majority is significantly larger. Finally the remaining partition of the data is being evaluated by using the same rules that constructed the box. These two resulting datasets can be compared. This is shown in figure 4.8.

This figure shows clearly that, although the rules are entirely created on the trainset, it results in both a similar distribution as a similar imbalance ratio when being generalized to unseen data. This speaks in favor of the theoretically drawn conclusion that it does not result in systematic under- or overestimation when $\mathbb{P}_{tr}(1|X, X \in B)$ is used to estimate the probability that a certain unseen observation, falling within the box, belongs to the positive outcome class.

However, there is one specific remark to be made. While this method allows the user to obtain a more balanced dataset to train its algorithm on and generalize this in an unbiased manner to unseen data, the objective as stated in (4.27) is slightly different. Because, as mentioned before, the boxes are clearly defined, the conditional probabilities $\mathbb{P}(1|x_{te}, x_{te} \in B)$ and $\mathbb{P}(1|x_{te})$ for an unlabeled observation x_{te} can be linked easily using (4.31). However this still required the assumptions made in (4.30). If, at any time, the PRIM algorithm is able to neatly eliminate all areas of lesser interest, while remaining *each* minority class observation in both the train- and test set, these would be met completely. However, this is obviously not always the case.

This implies that, as opposed to the undersampling case, by making the assumptions in (4.30) and using equation (4.31) to reconstruct the overall posterior distribution, this method actually slightly **underestimates** the overall positive class posterior probability. This means that, unless indeed the performance benefits from the reduced level of imbalance (as the theoretical background would suggest) this method runs the risk of actually performing worse on the class of interest. Of course, by the design of the validation structure (4.3) in practice this is prevented. Furthermore, to fully eliminate this effect, the possibility still remains to also construct a predictor on the feature space outside the boxes.

Concluding remarks

Throughout this chapter, the novel bump hunting method for preprocessing the data is introduced and described. The motivation behind this method is that by operating not on the level of individual

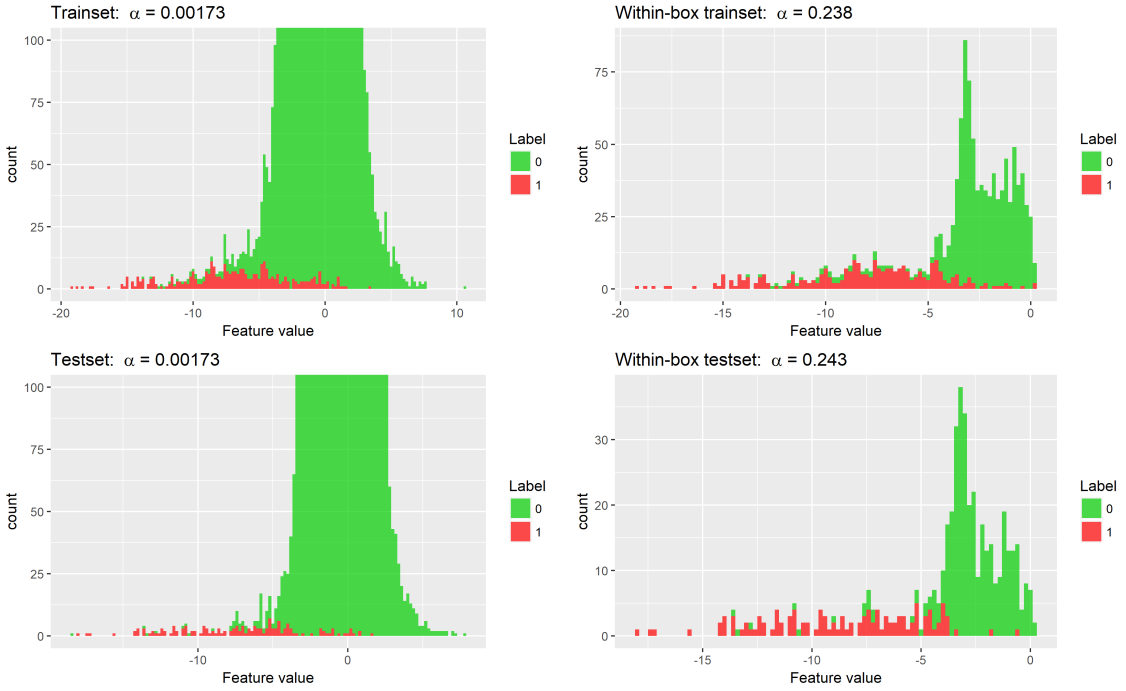

Figure 4.8: Counts of both outcome classes on the complete train and test partition (left) and within the remaining subset when the rules generated by the PRIM algorithm are evaluated on both sets (right). All imbalance ratio's α_0 and α_B are shown in the titles.

observations, but in stead reducing the volume of the feature space, a resulting more balanced situation can be extended to unseen data. If this is indeed the case, future instances within this restricted feature space are expected to have a much higher chance of belonging to the original minority class. This then should result in a situation in which classifiers are able to more accurately classify these instances without having to introduce a systematical overestimation in the process. Both the empirical results in figure 4.8 as well as the statistical analysis in equations (4.28)-(4.35) seem to confirm this conjecture.

It should be noted that, due to the potentially available skewed misclassification costs, a slight overestimation as mentioned above does not have to be undesirable. In this case however, it is still very possible to either combine bump hunting with resampling, or for instance lower the probability threshold in the classifiers to manually introduce and control this overestimation.

Chapter 5

Experimental framework

Before all described classifiers, resampling techniques and the novelty bump hunting method can be evaluated empirically, some additional experimental framework needs to be specified. First of all, a uniform measure of performance needs to be introduced to be able to analyze the functioning of individual methods and, more importantly, allow comparison between them. This may seem obvious but especially in the case of imbalance this is a crucial consideration to make.

Furthermore a dataset must be presented in order to implement and test all resulting *classification pipelines* on. As the main area of focus of this thesis is credit card fraud, a corresponding transaction data set is acquired. Finally, for benchmarking purposes, the effect of using and comparing multiple implementations across multiple platforms on this specific dataset is explored.

5.1 Evaluation

Thus far the classification setting has been introduced, together with imbalanced data and the challenges it brings. But a fundamental question throughout the entire domain that had been introduced remains: *how do we evaluate the performance of a classifier?*

Of course, performance on itself is a vague notion. The goal of a classifier, or its main point of focus, is how well it *generalizes* to unseen and unlabeled observations and how accurately the outcome of these observations can be predicted. Obviously, unseen data is rather useless for directly assessing the performance of the trained classifier, as the true outcomes are yet unknown and may take quite some time (if possible) to obtain. Hence the most common practice to reserve a holdout- or testset from the available pool of data. This test data is then used to evaluate the trained classifier on, by comparing its true labels to the ones the classifier predicted. The partitioning of the data already had been used in the classification on the example dataset of the Pima Indians. Quantifying the performance of a classifier can be done through other methods, such as cross-validation, but (especially when N is large enough) partitioning the available data into a training and testing set is common practice and hence will be used here.

5.1.1 Performance measures

Earlier in this document, evaluation of a classifier has briefly been mentioned. For instance in figure 2.4 the *evaluation metric* test error was used. This metric, together with a number of other performance metrics, follow directly from the **confusion matrix**, which directly compares the true labels to the labels as predicted by the classifier. This results (for the binary case) in a $2x2$ matrix, as displayed in figure 5.1. Test error as used previously, or the partition of observations that were predicted wrongly, is equivalent to the most commonly used metric: accuracy. Accuracy is defined as the proportion of

correctly classified sample points, which can be formalized trough the confusion matrix by:

$$accuracy = \frac{TN + TP}{TN + TP + FN + FP} \tag{5.1}$$

Intuitively, the accuracy (or test error rate which equals 1-accuracy) seems like the obvious choice for evaluating the performance of a classifier as it directly assesses the overall performance of the classifier on both classes combined (hence over the entire domain of y).

However,as argued earlier, especially in the case of imbalanced classification, accuracy can be misleading and it may be reasonable to include metrics that more directly involve one of either classes. For the performance of the classifier on the positive class, the recall is used:

$$recall = \frac{TP}{TP + FN} \tag{5.2}$$

which directly computes the proportion of actual positives that the classifier identified correctly. On the other hand, for evaluating the accuracy rate of the predictor on the negative class, the specificity is defined as:

$$specificity = \frac{TN}{TN + FP} \tag{5.3}$$

The specificity is the direct equivalent to the recall, but for the opposing class. To further assess the performance of a classifier on predicting the positive class (which often is the class of interest), the precision is defined:

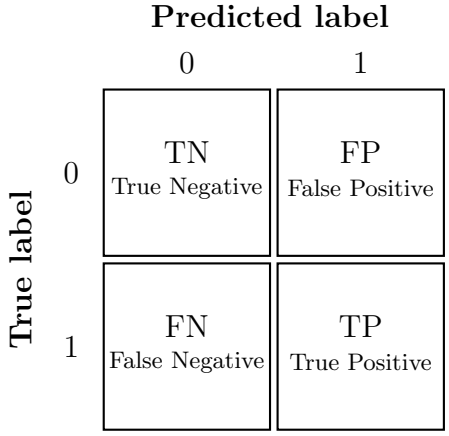

Figure 5.1: Confusion matrix for binary (0/1) classification.

$$precision = \frac{TP}{TP + FP} \tag{5.4}$$

which, opposed to the recall (5.2) which involved the proportion of true positives identified, focuses on how many of the observations that are labeled positive by the classifier, indeed belong to the positive class.

In the case where there specifically is one class of interest, which obviously is the case for fraud detection, the metrics in (5.2) and (5.4) are essential. A *good* classifier has to find the proper balance between recall and precision which means that it can detect a substantial amount of the true fraud cases, while also limiting the amount of samples that are wrongly labeled fraudulent. The most common metric that considers both and attempts to capture this trade-off is the **f-score** (or f_1 score)

$$f = 2 * \frac{precision * recall}{precision + recall} = \frac{2 * TP}{2 * TP + FN + FP} \tag{5.5}$$

In practice it is very common that misclassification costs are non-uniform, hence one might want to put more magnitude on recall (finding all positives) than on precision (reducing the amount of false positives), hence for this reason the more general f-measure is defined through

$$f_\beta = (1 + \beta^2) * \frac{precision * recall}{(\beta^2 * precision) + recall} = \frac{(1 + \beta^2) * TP}{(1 + \beta^2) * TP + \beta^2 * FN + FP} \tag{5.6}$$

where a larger value (>1) for β results in more emphasis being put on recall as opposed to precision (or more precisely on the amount of false negatives which follows from the right side of equation (5.6)). For that reason for instance the f_2 score (or even higher) is often being used when misclassification costs are non-uniform, for instance in the case of fraud. It immediately follows that using $\beta = 1$ returns the general f-score (5.5).

ROC analysis

Another strong and widely adopted tool for assessing the performance of a classifier is ROC analysis. Instead of assessing performance through metrics that follow directly from comparing the true labels to the classifier's predictions (figure 5.1) this method defines a two dimensional ROC (short for *receiver operating characteristics*) space. This space is spanned by both the true positive rate, which was previously referred to as recall (5.2), and the false positive rate

$$false\ positive\ rate = \frac{FP}{FP + TN}\ (= 1 - specificity)\,. \tag{5.7}$$

This two-dimensional ROC space is shown in figure 5.2.

Of course, by definition of this space, an optimal classifier would be located at the upper left corner, meaning that it has a true positive rate close to one (all positives can be identified by the model) while the false positive rate is close to zero (no observations wrongly classified as positives, or equivalently all negatives correctly classified as well). It is common practice, as is done in figure 5.2 by the dashed line, to include the diagonal in the graph. This corresponds to a random guess, where by definition the false positive rate is equal to the true positive rate. The points in a ROC graph can therefore also be interpreted as an answer to the question: *how much better does my model perform than random guessing?*

If the classification model that is being selected for training on the available data and for predicting future instances produces a direct class prediction (for instance a single decision tree defined in equation(2.6)), then it can be represented as a single point on the graph. Generally speaking, the further away this point is located in the upper or left direction with respect to the diagonal, the better is the performance of the corresponding predictor. However, this isn't as straightforward as it seems, and as it would be in for instance comparing two classifiers based on their accuracy or f-score. To illustrate this difference, consider the points 1 and 2 in figure 5.2. Based on false positive rate, the classifier with performance corresponding to point 1 would be favorable, while the classifier corresponding to point 2 scores significantly better on recall (true positive rate). In this case, selecting the optimal classifier could depend on the underlying setting or the user's preferences.

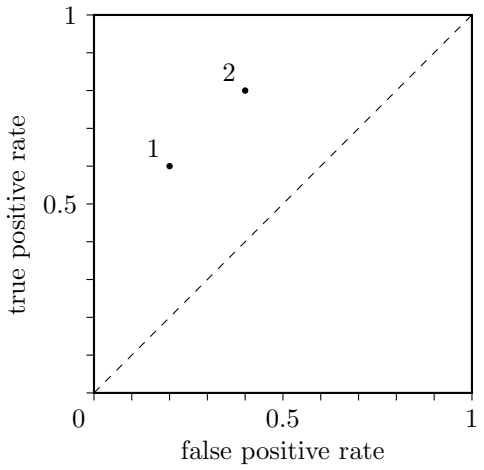

Figure 5.2: ROC space

ROC curve

On the other hand, as mentioned in the introduction from the section about classification, a lot of classifiers produce two *class probabilities*, the estimated probability of an observation belonging to either one of the classes. Examples of such classifiers are the logistic regression (which directly models this probability by (2.2)) or the different discussed ensemble methods where a large number of individual predictions are combined. By picking the class corresponding to the highest probability, these outcomes can easily be mapped to actual class label predictions, from which a confusion matrix and the corresponding relevant metrics can be computed. Obviously selecting the class with the

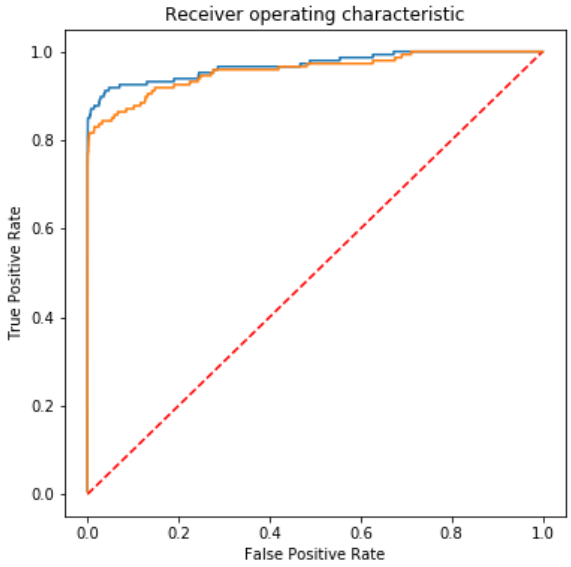

Figure 5.3: Example of two ROC curves generated generated by the `scikit-learn` library in `python`. The blue and orange lines correspond to two different classifiers trained on the same dataset.

highest probability is equivalent to setting a probability threshold of θ=0.5 and applying:

$$\hat{y} = \begin{cases} 1 & \text{if } f(X) > \theta \\ 0 & \text{if } f(X) \leq \theta \end{cases} \tag{5.8}$$

where $f(X)$ corresponds to the produced class probabilities by the classifier. While a threshold value of 0.5 may be intuitive, it is not guaranteed that it will produce the best results. This effect is magnified when misclassification costs are unequal or there is a lot of imbalance between the two classes within the data. Therefore, varying this threshold can improve performance.

Setting this threshold to the largest possible value, so equal to one in the probabilistic case, results in a classifier that labels every observation as belonging to the positive class. This specific classifier will end up in the lower-left corner of the ROC graph; each positive instance will be missed, while each negative instance will be identified correctly. On the other hand, a threshold value of zero will produce a classifier which correctly labels each positive, while simultaneously missing all negatives.

Varying this threshold between 0 and 1 thus results in different locations on the ROC space, which, when connected form the *ROC curve*. An example of a ROC curve is shown in figure 5.3. The ability to generate such curve is one of the strengths of ROC analysis, as the user does not have to pre-specify a threshold to be able to compute a confusion matrix and its corresponding metrics. Instead the outcomes over the entire threshold spectrum can be included. As with the discrete class predictions, one would prefer a classifier with a ROC curve that 'touches' the upper-left corner.

Determining the favorable classification algorithm based solely on the ROC curve can still be a difficult and somewhat arbitrary task. In the case where one curve (almost) completely exceeds the other, as is the case with the blue curve in figure 5.3, this is trivial. But more often than not, two curves may intersect an some point(s). In order to incorporate the above described ROC analysis, but still be able to rank classifiers accordingly, the AUC metric is introduced. The AUC, short for *area under curve*, is defined as the proportion of the area under the ROC curve. By definition, the AUC is between 0 and 1, where the random guess diagonal corresponds to a AUC of 0.5. Statistically,

the AUC corresponds to the probability that the classifier will assign a higher class probability to a positive, than a negative class observation [47].

For data with a skewed class distribution, the AUC may be a good choice in measuring the performance of a classifier, as it considers the relative performance on both classes separately, as opposed to for instance accuracy.

5.2 Credit card transaction dataset

5.2.1 Dataset description

For the main part of this research a dataset containing credit card transactions is being used. This dataset is made publicly as part of another research [48], and openly available through the platform Kaggle (`https://www.kaggle.com/mlg-ulb/creditcardfraud`). Each observation within the dataset contains information about a credit card transaction made in September 2013. The datasets consists of **284.807 rows**, or transactions, and **31 columns** or features.

Feature	Type	Mean	Range
V1 - V28	*double*	0	figure 5.4
Amount	*double*	88.35	[0,25619]
Time	*integer*	94814	{0,1,...,172792}
Class	*binary*	0.001728	{0,1}

Table 5.1: Description of the credit card fraud dataset

As table 5.1 indicates, all features apart from the target class label are numerical, which is convenient in selecting the model. Due to obvious privacy and client protection regulations, all features except for Time and Amount are anonymous. This is achieved through performing Principle Component Analysis transformations [49] on the original features. This process generated features V1 to V28. All are distributed around zero, where their individual ranges for both outcome classes are displayed in figure 5.4. The meaning and contents of the features from which V1 to V28 are constructed are unknown as they can not be disclosed. Of course, as another result of the PCA transformation, all of the features are complete, in the sense that they do not contain any missing values.

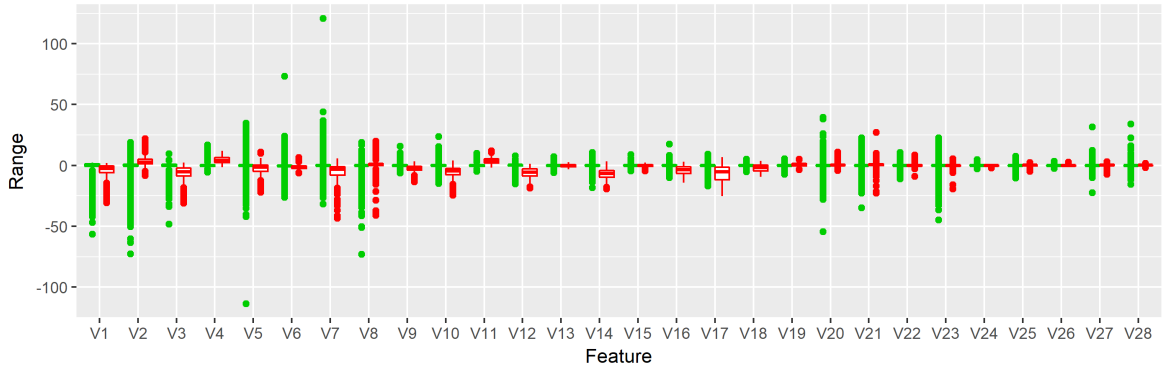

Figure 5.4: The range of each of the anonymized features for the fraud (red) and non-fraud (green) outcome class.

Apart from the 28 different unnamed features, there are two more features from which the meaning is known: Time and Amount. Amount corresponds to the height of the individual transaction. As one would expect, most of the transactions are for a rather low amount (for instance groceries or

gas). As a result, almost 80% of all transactions are below 100 Euro. On the other hand, the data shows some extreme outliers up to over 25.000 Euro. For better visualization, three different regions of transaction height are investigated and their distributions are shown in figure 5.5. The columns Time corresponds to the number of seconds elapsed since the first transaction in the dataset. As with the previous features, for each transaction both values are known, hence the entire feature set contains **no missing values**.

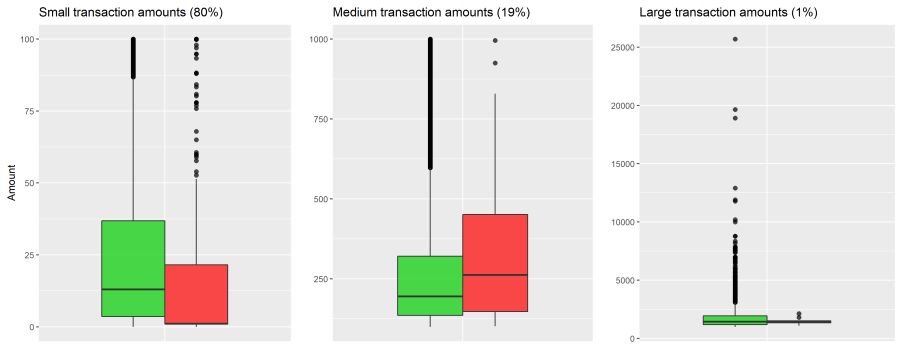

Figure 5.5: The distribution of the feature Amount, separated into small,medium and large transactions. The green columns correspond with legitimate transactions, while the red ones were marked fraudulent.

The remaining columns in the class label Class, which is binary coded where a 0 corresponds to legitimate and a 1 to fraud. The distribution of the two classes is shown in figure 5.6. As can be concluded from the numbers, only **0.17%** of all transaction are marked fraudulent, which corresponds to about 600 legitimate transaction to 1 fraudulent one.

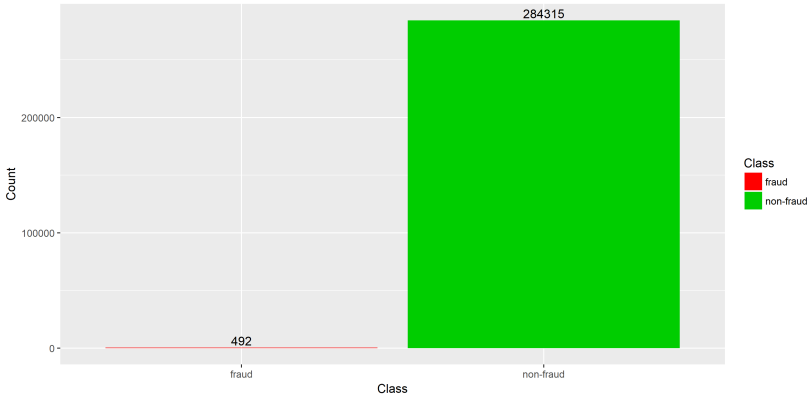

Figure 5.6: Distribution of the two outcome classes.

5.2.2 Data exploration

Before actually applying the described classification methods to this (highly imbalanced) dataset, it may be helpful to do some exploratory analysis on the available data. The most important question to ask is: *can the outcome of an unlabeled new transaction be predicted based on the available features?* If for instance variables are **correlated** to the outcome, or their distribution differs significantly for both outcome classes, this could indicate a higher chance for success. If, on the other hand, there is

no clear relationship between (some) features and the outcome, this could potentially make the task much harder.

Figure 5.7 shows the individual correlation of each of the features with the label. The values in this figure are Pearson correlation coefficients, computed by the `corr` library in `R`. Although the question of relevance remains, due to the fact that correlation is designed for continuous variables and hence treats the class labels as such, it could be an indicator that some of these variables could function as a strong indicator for fraud. Special interest goes to the features on the left and right end of the figure.

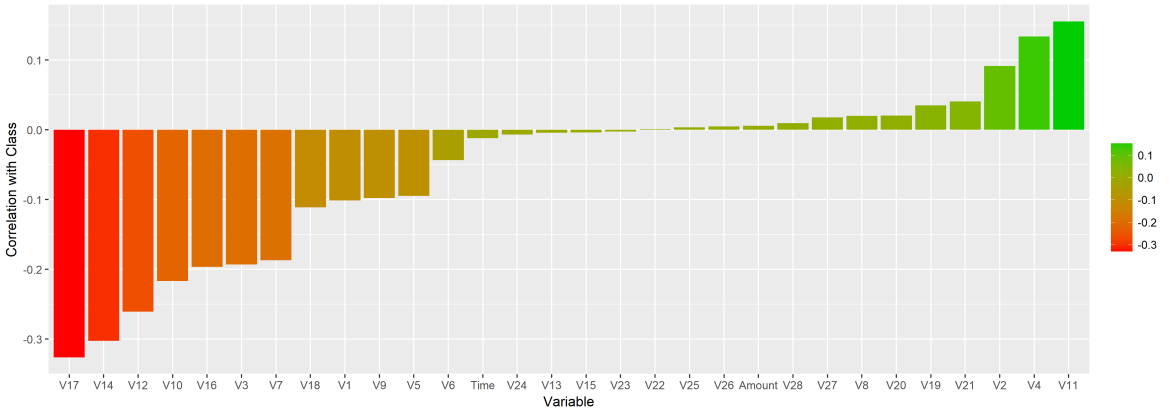

Figure 5.7: Pearson correlation coefficients for each feature specified in table 5.1 against the outcome label.

For further exploring these potentially interesting features, it is more reasonable to display its distribution for both outcome classes. This can be visualized nicely by using the `ggplot2` library in `R` (figure 5.8).

In all four of the plots, there is a clear difference in distributions visible amongst the two different outcomes. While, as mentioned, just using correlation in this case could be misleading and hence should be treated with caution, the fact that amongst these features a clear shift can be observed may be an indicator that based on the available information a functioning classifier can be trained.

5.3 R versus Python

Fortunately nowadays, implementing and running different types of classification algorithms doesn't require for the user to personally code them. This as there are a large number of open-source implementations available and ready to use. Of course directly copying and using each source available without some further investigation increases the risk of including certain results that where generated wrongfully. As a result, incorrect conclusions can be made by using (some of) these flawed of suboptimal implementations, especially when multiple different sources/developers are included.

To overcome this potential problem, the first step in selecting a specific method to use is research. Usually it does not take a lot of effort to find, throughout either literature or digital sources, which implementation is the most commonly used or best suits the user's wishes. But differences in implementation can be more subtle than mistakes made by the creator. It can also be that slightly different (default) settings or a different underlying optimization strategy can result in slight skewed outcomes. Therefore, doing some testing may be relevant as well.

Within this research, two main programming languages will be used: `R` and `Python`. Which of the two is used where mainly depends on specific applications such as which has a better functioning implementation or which has a more suitable library for for instance making certain visualizations. For machine learning purposes, `Python` has the clear advantage that it contains the `scikit-learn`

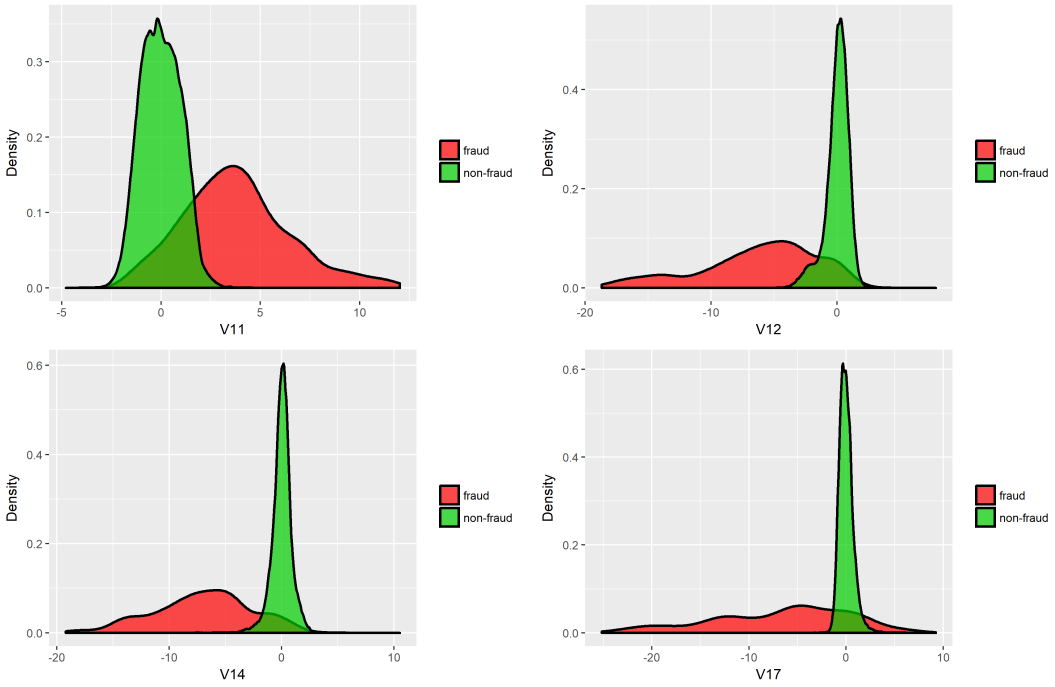

Figure 5.8: Density plots of four features that are potentially of a bigger interest due to the correlation analysis.

library, which features all state-of-the-art machine learning algorithms and tools in one place, which means that the user can skip the research for each specific method, as mentioned above. Furthermore, they all require similar data structures so prepping the data for each application involves much less work.

Although this makes it very tempting to only use the `Python` programming language, as it will turn out, there will be situations where, while using the same classification algorithm, but combined with a different method, using `R` is preferable. For this situation, comparing baseline performance on the data is essential.

Logistic Regression

For a large part, the Logistic Regression will be used. This due to the fact that it is easy to implement and, most of all, does require relatively little computational needs and training time. This can be crucial when processes involve a lot of trained classifiers. On the other hand however, using the logistic regression does not mean that prediction automatically have to be much worse. The logistic regression obviously is included in the above mentioned `scikit-learn` library. It is also included as a default function in `R`.

In order to compare different methods, where the classification is done through either of the implementations, it is important that both perform similar. To test this, both versions are trained and tested 100 times on the credit card fraud dataset (table 5.1). First metric used for comparing the results of both implementations is recall (5.2), as this directly monitors the proportion of frauds detected correctly. The results after 100 trained classifiers using both libraries, displayed in figure 5.9, indicate that it, in the case of these two logistic regression implementations, is safe to assume that they perform similarly in terms of recall. As the left figure indicates, the means seem to converge to a similar value, and following from the box plots this conclusion is confirmed.

To make sure that the two implementations also perform similarly on the non-fraud class, the f-

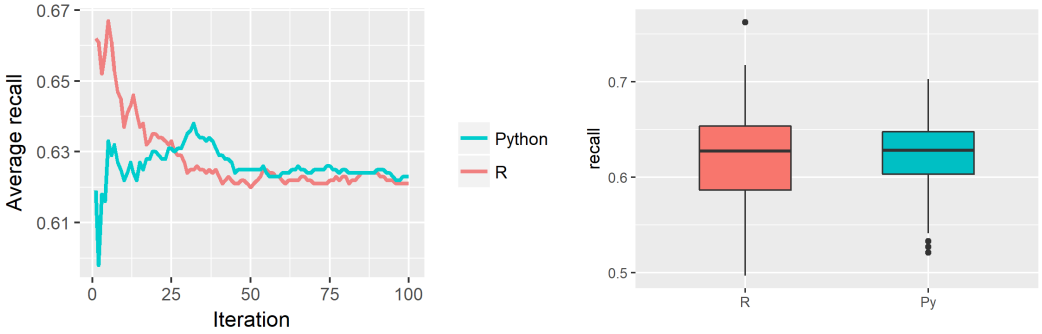

Figure 5.9: Comparison of two logistic regression implementations. On the left the progression of the average after each iteration is tracked, while on the right the situation after all 100 classifiers is visualized.

score (5.5) is briefly analyzed as well. As figure 5.10 indicated, again it is safe to assume that the two classifiers perform equally. This means that comparing different data-preprocessing methods while using either of the two implementations in the classification stage, does not result in conclusions with a certain bias due to using one of the two.

Figure 5.10: The similar plots as in the previous figure, but now by considering f-score (5.5) as measure of quality.

Random Forest

The second algorithm that, in a later stage, will be used across multiple programming platforms is the random forest classifier. As mentioned before, the random forest algorithm requires relatively little iterations to converge and the underlying decision tree classifier is quick and asks for little computational needs. For that reason it makes it particularly suitable for methods that require a lot of trained classifiers, such the Bump Hunting method (figure 4.3).

In `Python`, similar to the logistic regression part, the implementation from `scikit-learn` will be used. For using the random forest classifier in `R`, the library `randomForest` is used [50]. The library is based on the original random forest implementation in Fortran written by Breiman and Cutler [11]. In order to be able to accurately conclude if both implementations can safely be compared, it is essential that the hyper-parameters of both implementations are synchronized. As mentioned before, for the most part random forests are non-parametric (apart for some smoothing and regularization) but the most important choice is the number of estimators to use. Of course, more estimators result in an increased training time, but also possibly increased accuracy.

Strangely, according to the documentation from both implementations [51, 52] the default number of trees that is being trained varies greatly amongst the two. The `R` implementation used 500 trees by default, while the `Python` alternative uses 10. Because the number of observations is rather large, the algorithm is required to be executed multiple times and the level of improvement is more relevant than finding and tuning the optimal predictor a setting with significantly less trained classifiers would be preferred. For that reason, the first comparison between the two implementations is drawn by using the `scikit-learn` default configuration of 10 trees. To investigate the effect of amplifying the number of estimators, a second configuration of 25 trees is used as well. For both settings across both platforms the recall scores are shown in figure 5.11.

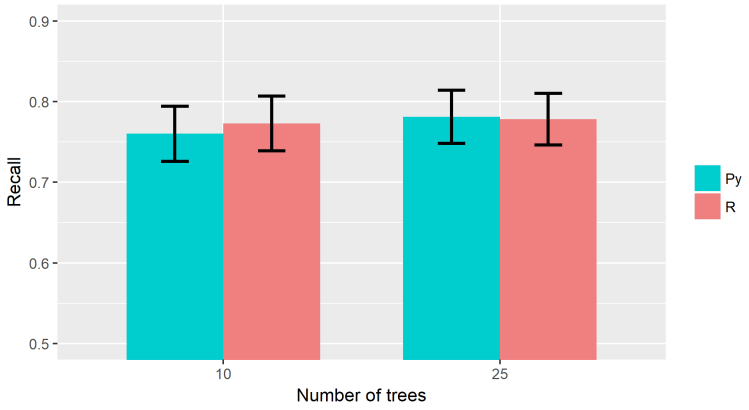

Figure 5.11: The recall (5.2) for two different configurations across both platforms.

To obtain the results from figure 5.11, the dataset is split at random, trained and evaluated for a total of 100 times. The scores are then averaged. The figure clearly shows that both implementations produce similar results across both platforms. Again, to be completely sure that results are equivalent, the f-score is being compared as well in a similar setting. The results are displayed in figure 5.12.

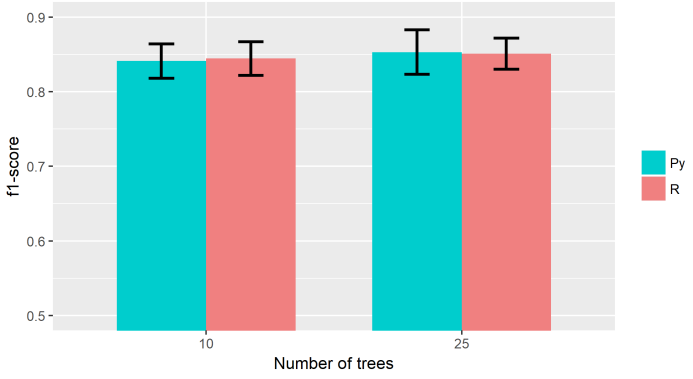

Figure 5.12: The f-score (5.5) for two different configurations across both platforms.

As again the two implementations match very closely, it can be concluded that it is safe to assume that comparing results generated by the two implementations across the two platforms does *not introduce extra bias* in the comparison of the results and effects of different methods.

Boosting

Boosting, and to be more precise the Adaboost implementation described in section 2.4.3, again is featured in the `scikit-learn` library in `Python`, hence this will be used. For `R` this is a less straightforward choice. There exist multiple libraries, but not a lot are capable of handling datasets of such magnitude in a reasonable timespan. By that restriction, the most suitable implementation comes from the `gbm` library [53], which features a number of gradient boosting models, one of which is Adaboost.

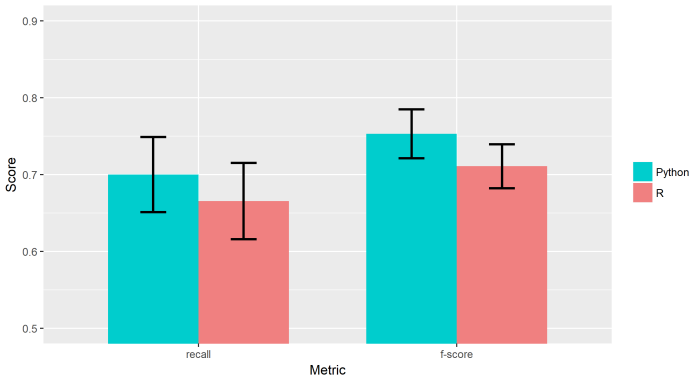

Figure 5.13: Comparison of recall and f-score for two different implementation of the Adaboost algorithm.

Strangely, while the same configurations are being used, the results are slightly different according to figure 5.13. In both performance metrics, the Python implementation scores better. Both do however have the same standard deviation, which on itself is quite high.

Unfortunately, the search for a more equal resulting implementation for either of the platforms was not successful. Of course this does not mean that the Adaboost algorithm should be discarded entirely, but that these differences should be kept in mind and comparisons should be drawn with a little more caution.

Chapter 6

Results

A whole lot of methods to overcome the difficulties when learning with imbalanced data through *resampling* the training data are presented. Some work very specific on small regions of the data (for instance Tomek link removal, visualized in figure 3.2) and others approach this more rigorously by either adding or removing observations until the balance is restored. After explaining how each of the methods functions, together with the intuitive thought process and reasoning behind them, the next step is to empirically test their performance.

This section starts off by **combining** most of the described methods in chapter 3 with the state-of-the-art classification algorithms specified in chapter 2, tested with the credit card fraud data from table 5.1. All methods are shown in figure 6.1. From there it should be able to address the overall feasibility of learning with this specific imbalanced dataset together with the potential (dis-)advantages of each classifier and method. Furthermore, apart from looking for how well each methods performs and which is the most suited combination, another main question will be very important: *does the outcome of each method correspond to the way that would be expected from the theoretical background?*

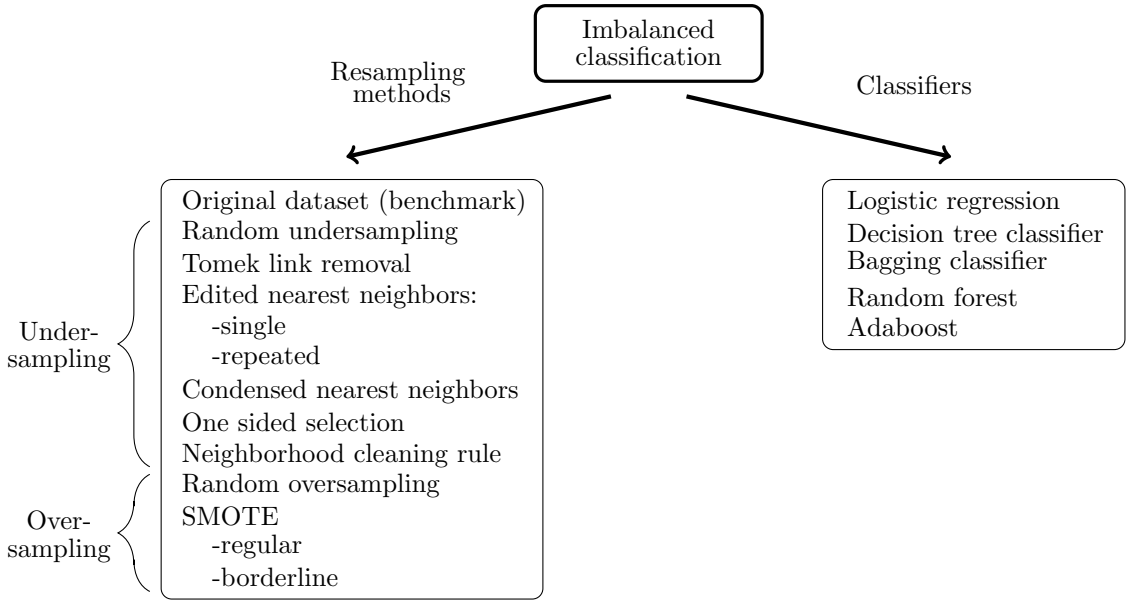

Figure 6.1: Methods and classifiers used for empirically testing the performance of each resampling method.

6.1 Benchmark

Before combining each treated under- and oversampling method (left side of figure 6.1) with all described classifiers (right side of figure 6.1) it is essential to first asses the performance on the *original dataset*. This can then be used as a benchmark to compare the (improved) performance with. Here can also be useful to do a little deeper investigation on how classifiers behave compared to each other on for instance performance on individual classes or speed.

As a starting point, to be able to fully comprehend the classifiers performance, on top of computing some of the described metrics such as recall (5.2) or the f-score (5.5) it can be relevant for display the entire confusion metric (introduced in figure 5.1). For two classifiers, logistic regression and random forest, this is shown in figure 6.2.

Figure 6.2: Confusion matrices for one instance of both logistic regression (left) and random forest (right) on the credit card fraud dataset.

To be able to compute these matrices, a **70/30** division is made, which will be repeated throughout this entire section. From this single instance the common metrics can be calculated as shown before, done in table 6.1. This table immediately shows why accuracy as a measure of performance is not

classifier	accuracy	recall	precision	f-score
LR	1.0	0.61	0.89	0.72
RF	1.0	0.76	0.97	0.85

Table 6.1: Common metrics computed for the two situations shown in figure 6.2.

suited in the imbalanced case, as it capture none of the obviously present differences whats however. For this reason, from now on, it will be omitted.

In the evaluation section, the ROC curve together with the AUC metric were also introduced as a state of the art methods for evaluating performance in the imbalanced setting. While this thereby very useful as a skew-insensitive measure to address and compare total performance, it is a lot harder to interpret the underlying implications as opposed to the metrics in table 6.1. An in- or decrease in AUC can be through a difference in performance on either one of the classes, and therefore investigating what exactly is the effect of a certain method is less obvious. On top of that it is not always straightforward to compute for instance the AUC for all different classifiers (especially across multiple libraries as will be done later). For these reasons, measuring performance through ROC analysis too is omitted in this section. It is thereby assumed that enough information about improvement and *relative performance* is captured in the remaining three metrics included in table 6.1.

For a more accurate comparison it is important that the process shown above is repeated multiple times. This to reduce the influence of random effects in for instance the partitioning in the data or the individual classifiers. For this reason throughout this section, each combination of method and classifier (figure 6.1) is repeated 10 times, after which metrics are aggregated. For all classifiers in this benchmark comparison, as well as all (resampling) methods in the proceeding sections, the implementation from the previously mentioned `Sci-kit learn` and `imblearn` libraries in `Python` are used. Furthermore, by fixing the random state over the iterations, for each method-classifier combination the same ten partitions are considered to make the comparison even more accurate.

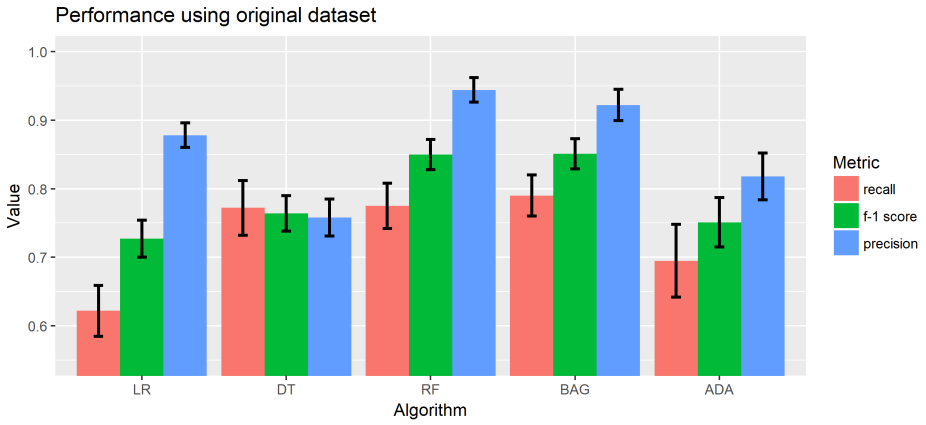

Figure 6.3: Aggregated results and standard deviations for all classifiers on the original (imbalanced) dataset.

The benchmark results on the original and imbalanced dataset are shown in figure 6.3. The figure shows a similar trend as in figure 6.1, where the more extensive random forest classifier overall significantly outperforms the simpler logistic regression. This is also the case for the two associated classifiers; decision tree, which scores a little less on precision, and the bagging classifier. On the other hand, the Adaboost algorithm seems to perform a little less, which could be seen as surprising given common knowledge.

As mentioned before, computational speed could also be taken into account, especially when a large number of classifiers must be trained or the number of transactions increases. The training time of a single algorithm is shown in figure 6.4. This clearly shows some significant differences, not all that unexpected from the theoretical background. Logistic regression remains by far the fastest algorithm, followed by the single decision tree (pruned back). Between the three remaining *ensemble methods*, random forest appears to be the fastest, with a training time of less than a fourth of that of the bagging classifier. As it was stated that a random forest by design takes relatively little iterations to converge to a solution, this can not be seen as a big surprise. It is important to note that, for all classifiers, the **default configuration** is being used.

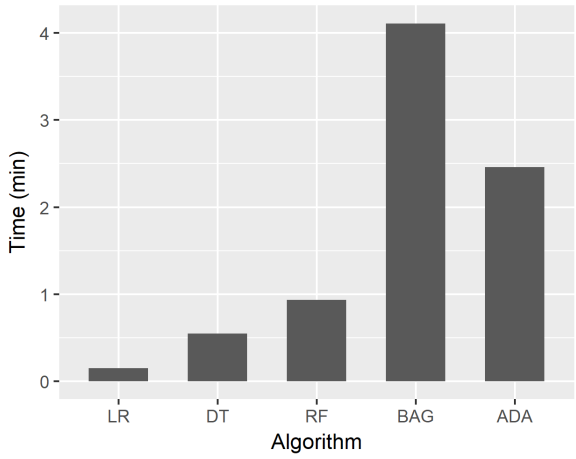

Figure 6.4: Training time for each of the classifiers on the original dataset.

This, as mentioned before, as this survey is more about investigating relative performances by using different methods than finding the optimal predictor for this specific dataset.

6.2 Resampling

Now that the benchmark performance is set, the following section will investigate how much each of the methods (potentially) improves on these scores. This is done in a similar order as in chapter 3, where again some methods will be grouped due to their similar nature.

6.2.1 Undersampling

The first and most trivial way of undersampling is by doing it randomly until the balance is completely restored. The results are shown in figure 6.5. For each classifier the recall increased to above **0.9**, but

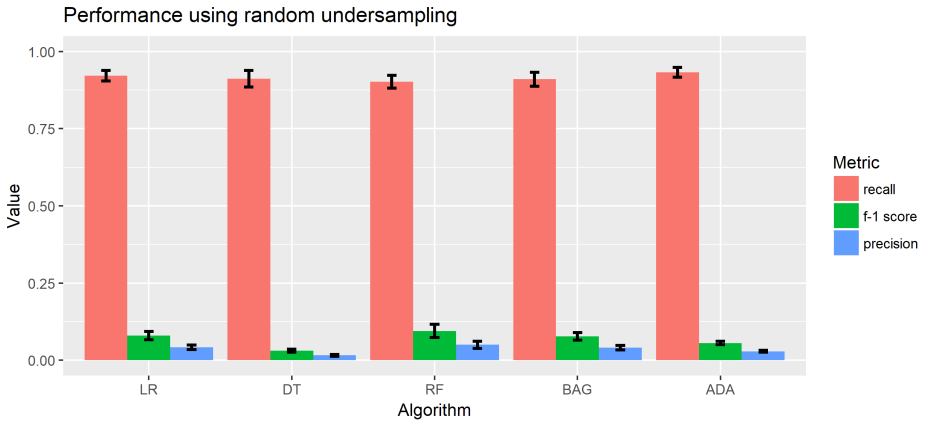

Figure 6.5: Aggregated results and standard deviations for all classifiers after performing random undersampling.

on the other hand the precision decreased drastically resulting in an overall very low f-score with none above 0.1. To be able to have a closer look at the precise influence of performing random oversampling the same two confusion matrices as in figure 6.2 can be computed.

These confusion matrices, displayed in figure 6.6, clearly show what is going on in this situation. While, especially for logistic regression (left matrix), the amount of correctly identified frauds is increased, this comes with a huge increase in false positives. Comparing the specific situations of figures 6.2 and 6.6, both evaluated on exactly the same test partition, the random forest classifier went from a false discovery rate (1 - precision) of about 3.5% to a staggering 96%! On the other hand, the random forest did discover 21 more true frauds. This means that, in this case, if misclassification costs of a fraud are over 125 times that of a non fraud, the overall costs still are less. Of course, this kind of very specific analysis requires a more thorough approach (as done in for instance section 4.4) and serves merely as an illustration.

Now how surprising are these observations? As was derived in equation (4.25) randomly undersampling the training set tends to systematically overestimate the probability of finding a fraudulent transaction. This, although it was in a purely theoretical framework, corresponds to what is observed here.

Figure 6.6: Confusion matrices for one instance of both logistic regression (left) and random forest (right) after performing random undersampling.

Targeted undersampling

The idea behind targeted undersampling is that, as opposed to removing observations with the main goal of restoring balance, specific sample points that complicate performance the most are discarded. In this section these informed undersampling methods are sorted and grouped roughly according to the amount of observations they remove, which is a slight deviation from the theoretical section.

The first informed undersampling method is Tomek link removal. Because of the scarcity of the fraud class (especially when you consider only a partition of it) and the fact that it can at most remove the amount of positives in the dataset, it is expected that this method will result in a limited change in imbalance. The results are shown in the left side of figure 6.7.

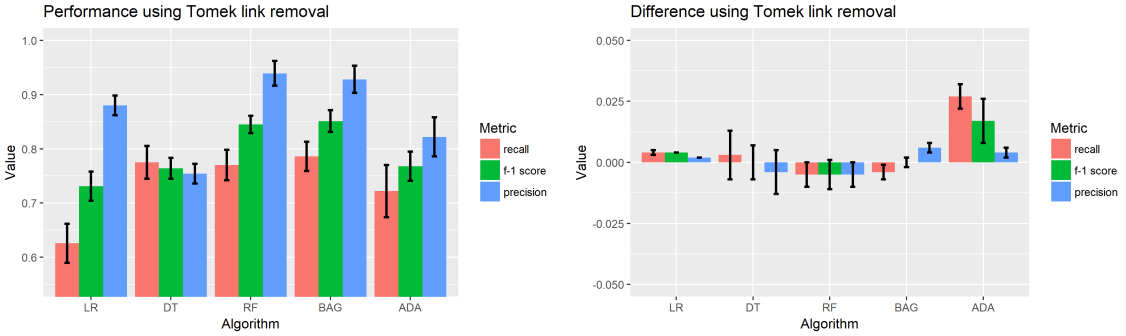

Figure 6.7: Aggregated results and standard deviations for all classifiers after performing Tomek link removal(left) and their differences as compared to the benchmark results.

As opposed to random undersampling, in this case as would be expected, the influence is much less significant and, when being compared to figure 6.3 it is hard to distinguish any difference with the naked eye. For that reason it may be more useful to calculate the difference as opposed to the original (benchmark) result and aggregate these values. This is shown on the right side of figure 6.7. Indeed, for most classifiers, the influence of removing Tomek links is minimal. In some situations it even (slightly) worsens the overall performance.

Apart from Tomek link removal, several other so-called *data-cleaning* methods where introduced in the theoretic survey. In particular one-sided selection, neighborhood cleaning and (repeated) edited

nearest neighbor. Figure 6.8 contains the results of these four methods. By similar arguments as above, the differences are more insightful then the overall performance scores, hence the aggregated scores are omitted. In table A.2 from appendix A the exact results are given.

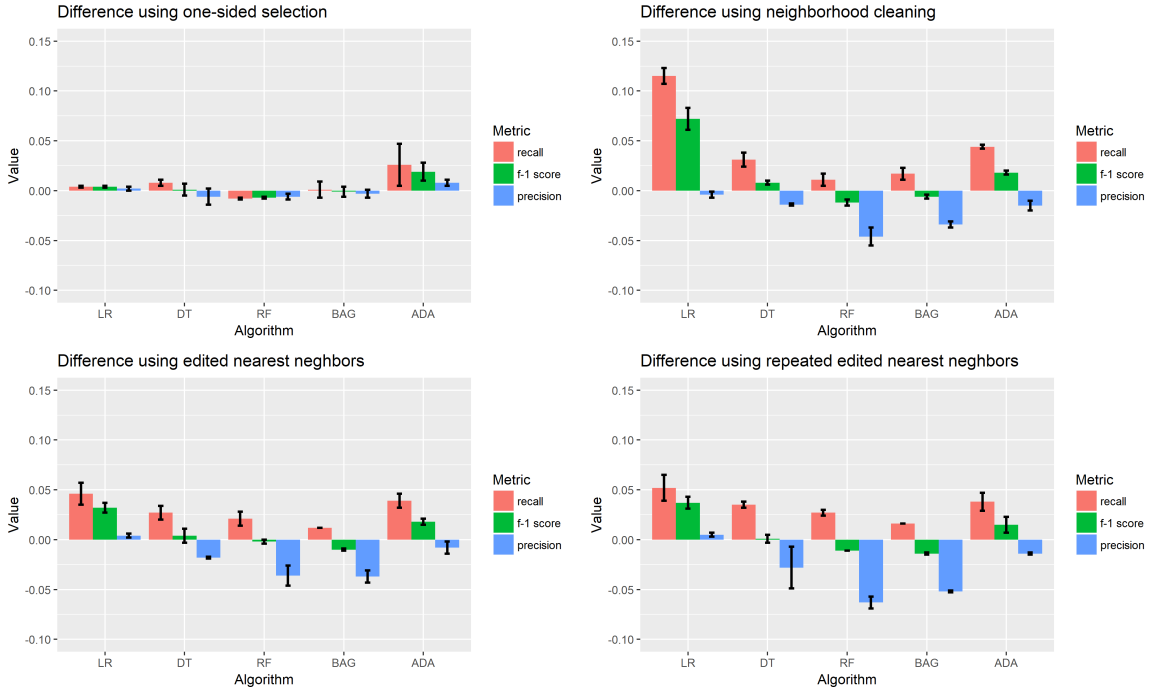

Figure 6.8: Aggregated differences between four different targeted undersampling methods and the benchmark performance.

While the effects of one sided selection show great resemblance with those observed after Tomek link removal, the other methods show different results. Neighborhood cleaning seems to work particularly well for the algorithms which initially performed the least on the fraud class (logistic regression and Adaboost). On the other hand, the edited nearest neighbors work relatively well on the other methods. All three methods do seem to slightly worsen the precision of the classifier. However, this level of decrease is still much less then observed after performing random undersampling and, by keeping in mind that the misclassification costs are often highly nonuniform, those differences could very well being considered profitably in terms of cost minimization.

The thing that all of these five undersampling methods have in common, is that they attempt to remove several specific observations, all in their own way. For that reason, these results are not quite unexpected. Indeed the classifiers perform better on the harder positive class, but this comes at some price as certain legitimate transactions will be falsely flagged due to the cleaning methods.

The remaining undersampling method, the condensed nearest neighbors methods, works slightly different. Instead of targeting the more difficult boundary cases, it aims to remove a lot of redundant points. As these are usually much more frequent, this method will automatically remove a lot more observations from the training set, and hence restore the balance a lot more. While for the other methods, the new level of balance did not exceed the original 0.17%, the condensed nearest neighbors method increased this ratio to over **43%**. This is still slightly less than with random oversampling, which by definition sets the ratio to 100%.

The results of the CNN method, as shown in figure 6.9, show that for each classifier the recall increases. However, due to the (expected) decrease in precision, in most cases a lower f-score is being observed. Although the amount of removed points with this method is of the same order of magnitude

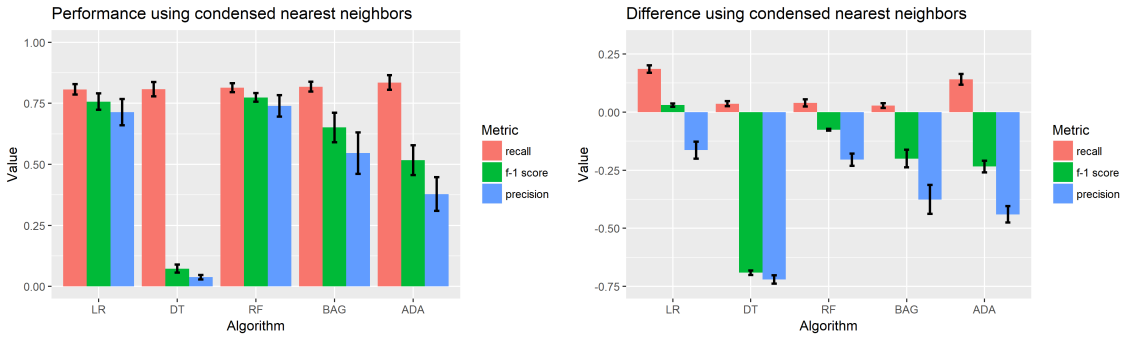

Figure 6.9: Aggregated results and standard deviations for all classifiers after performing condensed nearest neighbors (left) and their differences as compared to the benchmark results.

as with random undersampling, comparing figures 6.9 and 6.5 learns that the decrease in precision is decreased significantly for most classifiers. This does however comes at the cost of less gain in successful fraud detection. The exact differences can be observed in table A.1.

So which method works best in this setting? After considering all described undersampling methods, there is definitely no uniform answer to this question. This all depends on user preferences, misclassification costs and which algorithm is used. Even amongst the more conservative methods, this is very much dependent by the specific setting. The scattering of best scores (bold numbers) in table A.2 illustrates this nicely.

6.2.2 Oversampling

Analogous to the previous section, the first oversampling method considered is random oversampling, from which the results are displayed in figure 6.10.

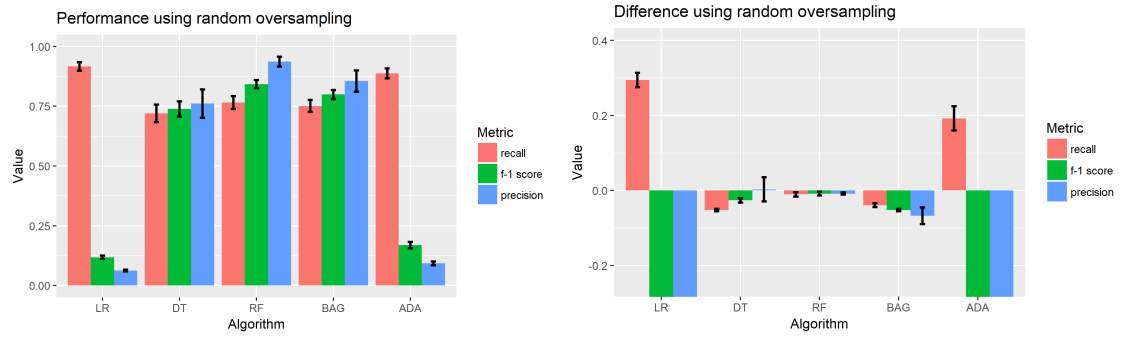

Figure 6.10: Aggregated results and standard deviations for all classifiers after performing random oversampling(left) and their differences as compared to the benchmark results.

The logistic regression, whose mechanics fit rather closely in the framework of section 4.4 as it directly attempts to model the conditional probability from equation (4.15), appears to have a very similar result as with random undersampling which is thereby not that unexpected. The Adaboost classifiers also behaves very similar to the random undersampling case. On the other hand, for the tree-based classifiers this is completely different. Randomly repeating similar points seems to have very little effect, and even overall slightly worsens the performance.

The two SMOTE implementations are, as stated earlier, methods of amplifying the minority dataset without actually exactly replicating existing sample points. By default, they do however

continue until balance is completely restored. Figure 6.11 shows the result of both configurations of the SMOTE method.

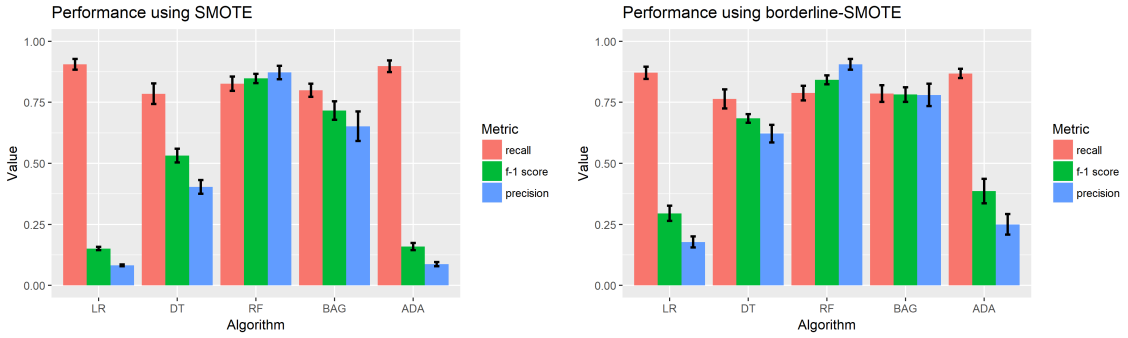

Figure 6.11: Aggregated results and standard deviations for all classifiers after SMOTE and borderline-SMOTE.

Both oversampling methods seem to show a similar pattern. To be able to accurately compare the three methods, it may be more useful, instead of comparing them to the benchmark, in this case compare them to each other as they do not appear to behave very different. Both the difference between the two SMOTE methods, as well as the comparison of SMOTE and oversampling randomly are shown in figure 6.12

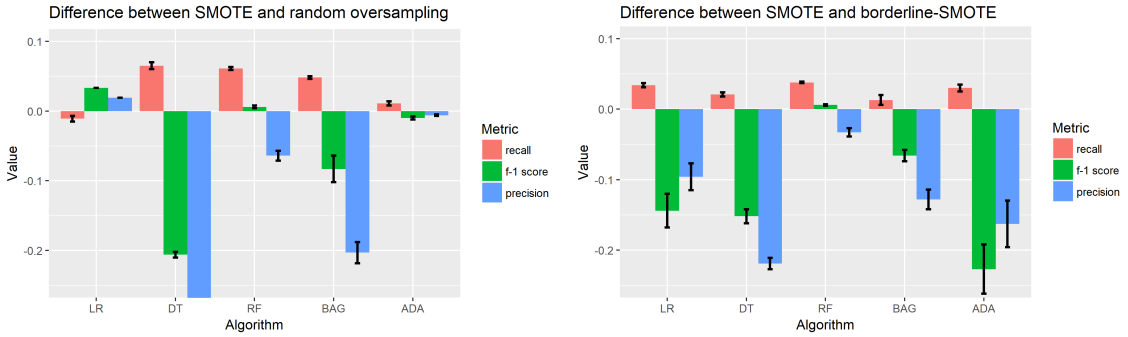

Figure 6.12: Aggregated differences between the three oversampling methods.

The result of the left image in 6.12 could also be deducted by comparing the aggregated results: SMOTE works pretty similar as random oversampling for logistic regression and Adaboost, but has a lot more significant effect on the remaining three classifiers. This comes, as been observed in a lot of methods, at the cost of some precision decrease however. The comparison between the two SMOTE implementations is a little bit more interesting. It appears that, in terms of identifying the positives correctly, the original SMOTE implementation outperforms on each classifier. On the other hand, by introducing the borderline ruling as explained in section 3.4.1 the decrease in precision is greatly reduced.

Especially in the highly imbalanced case, SMOTE tends to add a lot of noise due to the fact that it often attempts to connect small disjuncts within the positive class. This means that regions within the dataspace where originally no fraud would be expected, can now contain one or multiple fraudulent transactions. This could serve as an explanation for the big increase in the number of false positives. The borderline setting aims to avoid these specific new and synthetic sample points by only enlarging the borderline region.

All performance scores, as well as their standard deviation, are shown in table A.3 in appendix A.

6.3 Bump Hunting

In the coming section, the results of the introduced bump hunting method for overcoming difficulties with imbalanced learning will be presented.

6.3.1 Logistic regression

The first classifier which will be used is the logistic regression classifier. This is done for multiple reasons. First of all, as figure 6.4 shows, it has by far the shortest training time, which is useful as this method requires a lot of trained classifiers (see figure 4.3). Secondly, results from the previous section suggest that the performance of this particular classifier decreases most due to the high level of imbalance. This can be assumed by for instance comparing the gaps between benchmark performance (recall in figure 6.3) between logistic regression and for instance random forest, and the same gaps after resampling methods are performed (for instance in figures 6.5 or 6.9) where they seem to perform equally.

The complete method involves multiple steps. Through the PRIM algorithm [45] a series of rules is computed where each *box* of remaining training observations is being evaluated by the classifier on the other partition of the data. After the most suited rule set is selected, though covering, this process is repeated until the most suitable set of boxes is found. On these subregions of the feature space, unobserved sample points will be classified according to figure 4.2.

Single box

To be able to have some more insight in the functioning of this method, the first step is limiting the process to creating and scoring a single box. The PRIM algorithm for finding subgroups is implemented in `R` as part of the `subgroup.discovery` library developed for a different research project [54]. Although the possibility of using different target functions is mentioned, the mean will still be used here. The other two parameters, the peeling fraction and the minimal support, are kept very low. In this case regions are 'peeled off' very slowly until a very small box remains. Of course, this will traditionally probably be considered as overfitting, but as the individual boxes are scored differently, this does not matter. The progress of the scoring function for one instance, evaluated on the entire training partition, is shown in figure 6.13. By definition of the algorithm, this value increases up to one, where one or multiple fraudulent transactions are isolated.

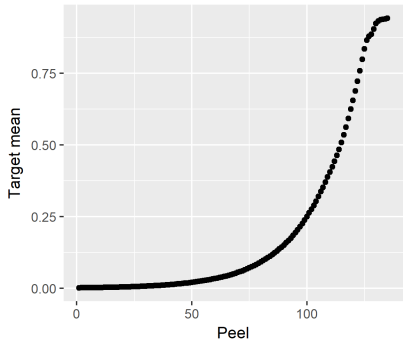

Figure 6.13: Target mean after each peel.

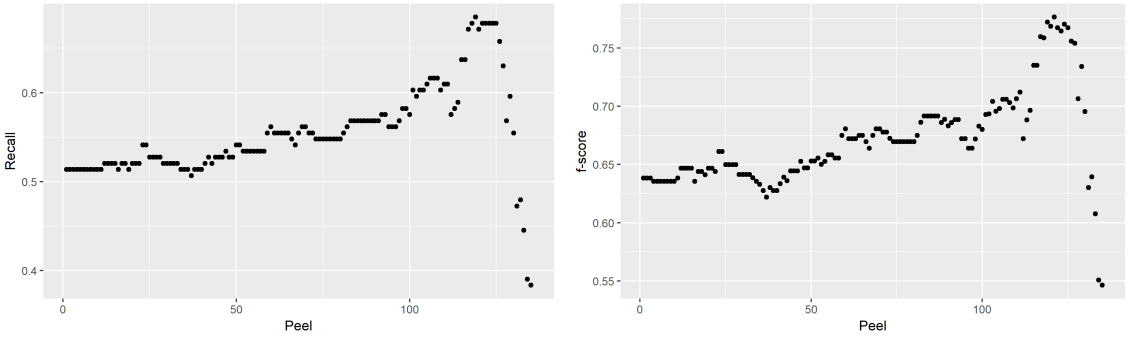

Figure 6.14: The result of two scoring metrics on the sequence of dataset.

The next step is scoring each individual box based on the classifiers performance on the resulting test set, and selecting the optimal ideal subgroup based on a preferred metric. Options for this metric are recall, which puts all focus on the minority class, or for instance the f-score, which also involves the precision. For this instance, the score of both of these metrics can be tracked.

In this specific case they follow a very similar trend. As the f-score is a direct trade-off between true positive and false positive, this means that the amount of false positives stays relatively constant over the entire trajectory. In other cases however, these could be different. Both do show a very promising increase as the box becomes smaller. It may also be relevant to, before and after selecting a preferred box, compare the confusion matrices as done before. This gives some insight into the behavior of the classifiers on both classes.

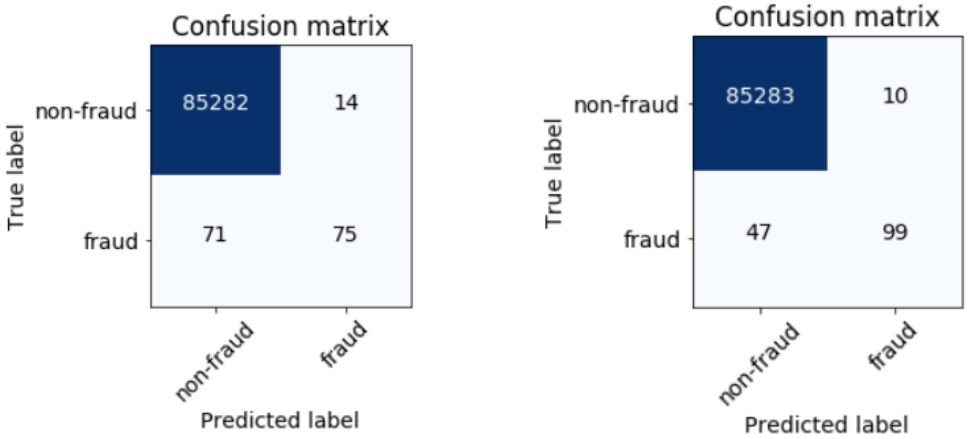

Figure 6.15: Confusion matrices of the initial results and the results of the optimal subbox. Classifiers are evaluated on the test partition.

The results in figure 6.15 show that, by applying the bump hunting procedure with just one box, the overall performance of the logistic regression classifier improved. Not only on the minority class, but also on the majority class. To gain a more robust result, this process is repeated 10 times, as was done with all other methods. Figure 6.16 shows that indeed this methods shows a significant increase in both recall and f-score.

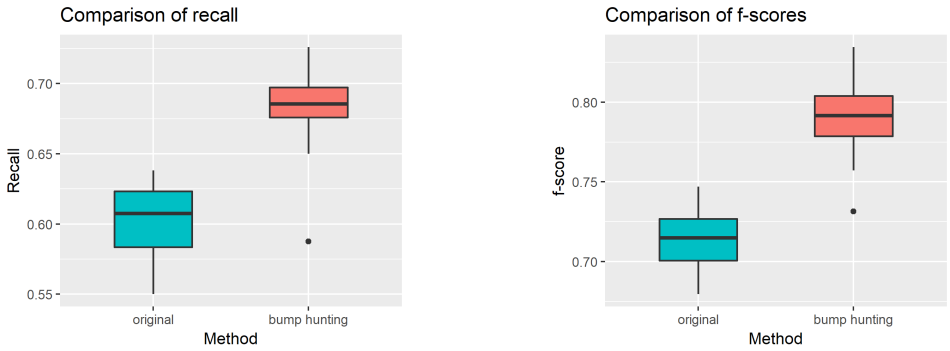

Figure 6.16: Initial scores versus the maximized scores for the two proposed evaluation metrics for the single box bump hunting method.

Multiple boxes

The single box section above can be seen as first conceptual evidence that, indeed, by restricting the feature space the classifiers performance can be improved. But of course for maximal performance, this process must be automatically repeated until the ideal amount of specific subboxes are selected. This is done analogous to the original covering, meaning that after selecting the proper subbox, each observation which it contains is removed both from the train, and the test partition. Next the same procedure is repeated. This until a specific situation arises where, after performing one peel, there are immediately no more fraudulent transaction left in the corresponding test set.

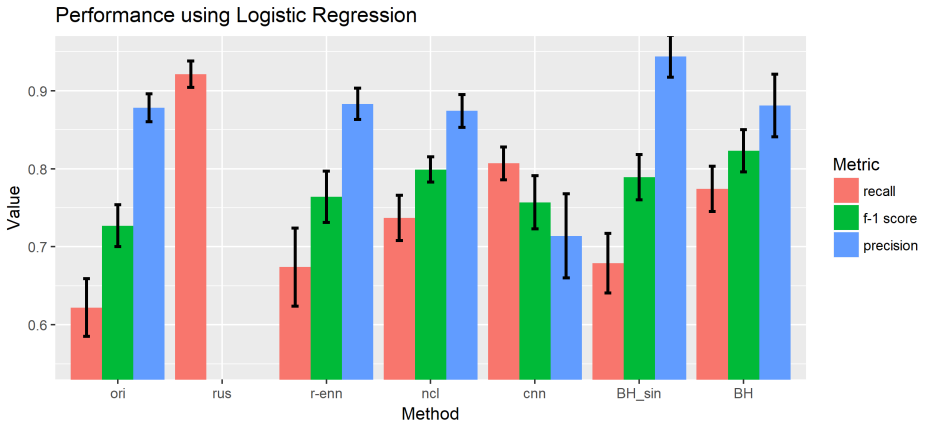

Figure 6.17: Aggregated results and standard deviations for the single box method (BH_sin) and the complete bump hunting method (BH) combined with the logistic regression classifier. As comparison, the figure also features some relevant methods from the previous section.

Figure 6.17 shows the results if this procedure, again repeated 10 times and aggregated, as compared to some of the most important 'competitors' for the previous section as well as the benchmark results. For completeness, this figure also includes the results of repeating the single box procedure from figures 6.15 and 6.16. The used scoring metric to optimize over in this case was recall but, due to the large correlation between f-score and recall (as visualized in figure 6.14) optimizing over f-score resulted in very similar scores. For this reason it is omitted in the results.

The results clearly indicate that, in terms of recall, the bump hunting method outperforms the other more conservative targeted undersampling methods and, although precision is slightly decreased when multiple boxes are included, the complete bump hunting method still outperforms **every other method** in terms of f-score.

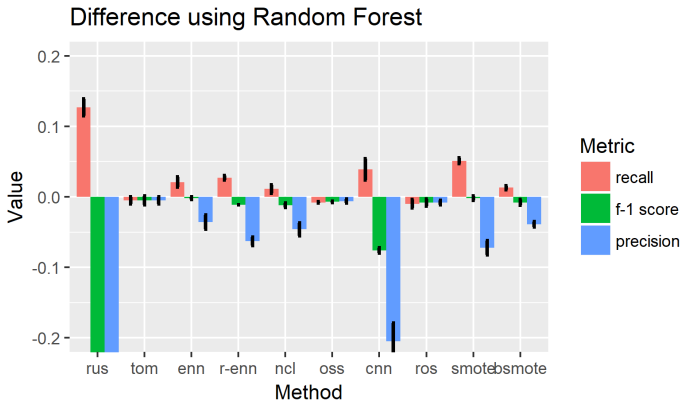

Figure 6.18: Difference in performance between all described resampling methods and the benchmark performance when classifying with random forest.

6.3.2 Random forest

The second algorithm that the bump hunting method is being tested on is random forest. Just as with the logistic regression, the `R` implementation is being used which, according to section 5.3 performs similar in comparison to

the `Python` equivalent which is used for the other results. According to figure 6.4, on the entire dataset, this classifiers requires significantly more training time which means that the entire process will be a little more time consuming.

By considering each of the results from the resampling survey for the random forest classifier from the figures above or the tables in Appendix A, it is evident that the random forest algorithm shows much less significant differences when being compared to the benchmark results in figure 6.3. These differences are all collected in figure 6.18, where the most significant improvements in recall from methods apart from random oversampling are of the order of 0.05. This is about a third of the improvement which was observed for methods applied to the logistic regression classifier. This could mean that, as the benchmark performance is on itself quite decent, it may be harder to improve upon and therefore it is interesting to see how the bump hunting methods functions in this specific pipeline.

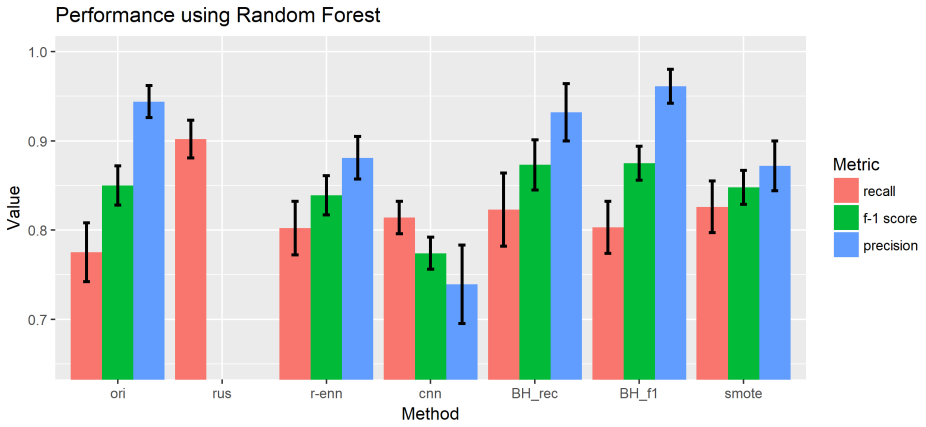

Figure 6.19: Aggregated results and standard deviations for the complete bump hunting method optimized over recall and f-score, combined with the random forest classifier compared with some relevant methods from the previous section.

When classifying with random forest, it turns out that the metric on which the algorithm selects the optimal box does make a difference, as the results in figure 6.19 show. Optimizing over f-score (denoted by BH_f1) does recover slightly less true fraudulent transactions, but does have the highest precision, resulting in the overall highest f-score amongst all methods. The other bump hunting implementation (BH_rec) does too outperform all other methods in terms of f-score, and apart from that, is able to compete with state-of-the-art methods such as SMOTE when it comes to correct fraud detection.

For completeness, figure 6.20 on the right shows the differences between the benchmark results and the most relevant methods, together with the two bump hunting methods. As it turns out, both bump hunting implementations are in fact the only two methods on which **all** three performance metrics actually increase.

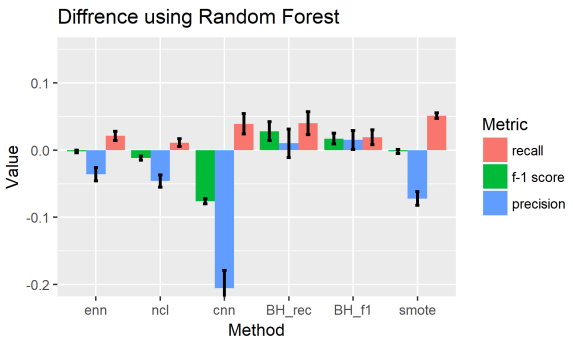

Figure 6.20: Most relevant methods of figure 6.18, together with bump hunting.

6.3.3 Adaboost

Next to generalized linear methods and tree-based (ensemble) methods, the third class consisting of the boosting classifiers is described in section 2.4.3, from which the Adaboost algorithm was explained in detail. As the benchmark

results from figure 6.3 suggest, the Adaboost classifier performed worse than the tree based methods. On the other hand, throughout theory it is assumed that boosting classifiers score high in terms of predictive capacity. For that reason it can be interesting to observe how the bump hunting method could potentially improve this for the highly imbalanced case.

It is important to realize that, by using the `gbm` implementation in `R` to validate and score each subgroup in the bump hunting process, and to eventually classify the entire testset, the benchmark performance is slightly worse than that of the `sci-kit learn` implementation shown in figure 6.3. This could potentially mean that directly comparing both of the outcomes introduces a certain bias in the analysis. On the other hand, as the results in figure 5.13 were not completely different, as long as this is being kept in mind, the investigation of the potential influence of the method can still be relevant.

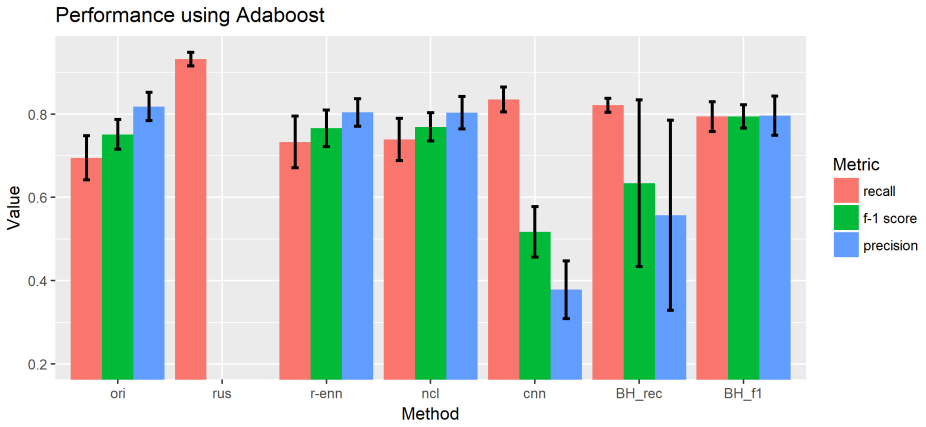

Figure 6.21: Aggregated results and standard deviations for the complete bump hunting method optimized over recall and f-score, combined with the Adaboost algorithm compared with some relevant methods from the previous section.

The result of repeating the complete bump hunting method ten times, together with some of the more relevant resampling methods used previously, are shown in figure 6.21. Just as with random forest, the results distinguish two scores over which the algorithm optimizes: recall and f-score. Both show interesting results. Optimizing over f-score yields a increase in true positives (recall) without a reduced precision, such as previously observed when using for instance condensed nearest neighbors or random undersampling. As a result this method has the overall highest f-score. This is all the more surprising when considering the significantly lower benchmark f-score as shown in 5.13.

When optimizing over recall score as in figure 4.3, the amount of correctly recovered fraudulent transfers is even higher. On the other hand, the precision (and with that the f-score) shows a surprisingly large variance. This indicates that, in some settings, the bump hunting algorithm picks a subgroup or box which induces a lot of false positives. For further use, this could probably be filtered out by including a certain precision threshold. It can be noted however that, despite this potential flaw, on average both the f-score and the precision still succeeds that of the condensed nearest neighbor method (which is the best performer in terms of recall between all the targeted undersampling methods).

Because of the difference in benchmark performance, it can also be relevant to investigate some of the differences between a certain methods performance and the corresponding original result. By doing this, visualized in figure 6.22, it can more clearly be seen that performance in greatly improved by the bump hunting method. Both implementation increase the amount of recovered fraudulent transfers by **over 10%**. Apart from that, similar to the random forest analysis, bump hunting with optimization over f-score is the only method which increases the score of each of the performance metrics.

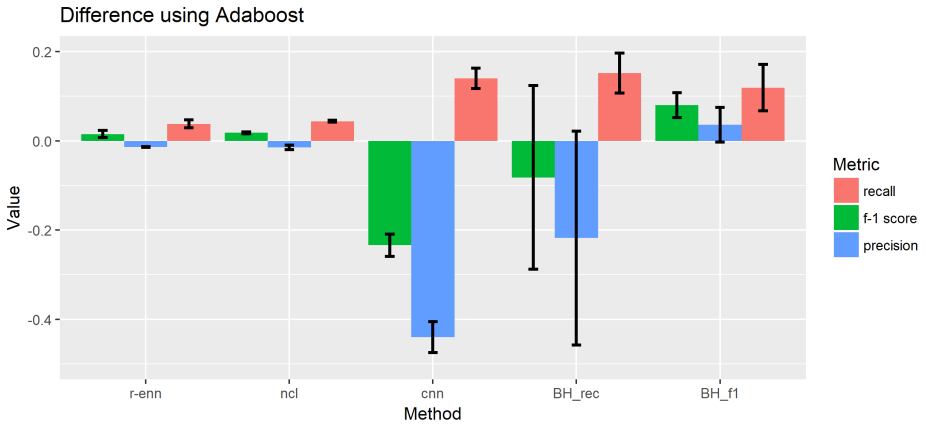

Figure 6.22: Aggregated differences to benchmark performance with corresponding standard deviations for the complete bump hunting method optimized over recall and f-score, combined with the Adaboost algorithm compared with some relevant methods from the previous section.

6.3.4 Further comparison

By considering all three classifiers on which the bump hunting method is tested, it appears that based on the proposed evaluation metrics they are able to compete with the state-of-the-art resampling methods that can be found throughout literature. For a more precise comparison, table A.4 of Appendix A provides all exact result of the bump hunting method on each classifier.

Apart from performance, the time consumption and computational needs can also be a factor in selecting the proper method for handling highly imbalanced data. Especially when the size of the dataset increases significantly. Random resampling for instance requires only index sampling. As this is equivalent to randomly picking a small amount of numbers out of a larger set, this can scale and be fast up to a very high amount of available observations. For methods that consider distance, this is much less the case, as by definition an increase in size results in an exponential increase in distances. To see how, with using the current dataset, the bump hunting method ranks compared to the established methods, a single execution is timed together with the logistic regression classifier. Of course exact values are of lesser importance here, as some might be variable sometimes while other repeat the same process over and over, but the order of magnitude can be expected to be representative for a given method.

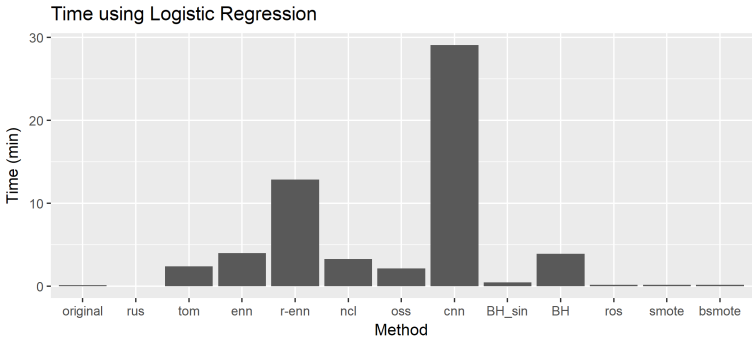

Figure 6.23: Time required for applying each method shown in figure 6.1 and the proposed bump hunting method (single-box and complete). For the classification, logistic regression is used.

As figure 6.23 shows, multiple groups appear. All oversampling methods, together with random

undersampling and the **single -box bump hunting** method are the fastest, all with a total time of under half a minute. Next to that, there is a group of methods with a time passed in the order of several minutes. To this group, the complete bump hunting method belongs. Next to these methods, the repeated nearest neighbors and especially the condensed nearest neighbors method both consume a significantly larger amount of time. This is to be expected as both require that mutual distance are calculated and tracked at any time.

Of course, due to the differences in training time and the large number of trained classifiers, the bump hunting method is particularly sensitive to the classifier that is being used in the validation process. However, due to the theory motivated in the end of section 4.3.2, accompanied by figure 4.4, it is usually not necessary to validate each subgroup. And, as with each subgroup the total amount of training observations decreases with a fixed factor, the differences in required time when changing classifier turn out to be less significant than figure 6.4 would suggest.

Chapter 7

Conclusion

Comprehending both the underlying mechanics, as well as the potential solutions of learning with highly imbalanced data has proven to be a relevant topic throughout literature with widespread applications far beyond just the detection of credit card fraud, the setting on which this research is focused. For that reason, gaining a deeper understanding of this turned out to be a major part of this work.

Throughout the entire analysis one statement continuously returned: learning with high imbalance is an issue because, by design, common classifiers cannot effectively process such datasets. To overcome this, obviously one could either alter the classification algorithm or somehow reduce the encountered level of imbalance. Due to generality arguments, it can be applied to any dataset and any classifier in an identical way, the second option is explored.

The extensive survey features most renown methods for both amplifying the minority class and compressing the size of the majority class. Furthermore, the reasoning behind each resampling method is explored. For undersampling the negative class, methods range from either randomly restoring complete balance, omitting certain specifically selected sample points that are expected to particularly increase learning difficulty, to more rigorous methods that aim to remove a lot of redundant data. For oversampling the class of interest, apart from randomly replicating instances to again restore balance, the main point of focus for the described methods is artificially generating synthetic minority class observations to add to the available pool of data.

Additionally, accompanied by a statistical argument, a novel method is introduced. This method, based on the the field of subgroup discovery or bump hunting, approaches data pre-processing in an alternative way. Instead of directly altering the data set on which a classifier is trained, restrictions are made on the data-space. By ignoring regions in which the majority class is exceptionally dominant, a more balanced situation can be achieved. Through this method however, as opposed to resampling, this more balanced subset can be generalized to new data. This as it is expected that future instances within these sub-regions will have a generally higher probability of belonging to the class of interest.

The effects of all described methods are experientially tested by applying them on a credit card transaction dataset. Here special attention is given to comparing the results of the introduced bump hunting method as opposed to the other established methods. As a result of the statistical analysis of section 4.4, it is expected that methods that remove a large number of points tend to highly overestimate the occurrence of a fraudulent transaction. This indeed is confirmed by the results. As for instance figure 6.5 shows, although randomly removing most majority class sample points does result in a large increase in correctly identified frauds, the increase of false positives is even larger. Similar behavior is observed in the oversampling methods. On the other hand, the more conservative methods such as shown in figure 6.8 overall make much less of an impact on the ability to accurately detect fraud. They do however mostly maintain the high level of precision of the benchmark performances.

Comparing the results of these resampling strategies to those of the presented bump hunting

method shows some promising results. It is the only method that is able to consistently realize an increase in all three of the considered performance metrics. For the logistic regression classifier, which has the worst benchmark performance, applying the complete bump hunting methods shows a significant increase in recall and the highest overall f-score amongst all methods. On the more advanced classifiers, random forest and Adaboost, it is even the only method that is able to improve on the benchmark f-score. These results all speak in favor of further exploring the possibilities of this concept.

Further research

For further exploring the possibilities of the introduced bump hunting method, numerous topics can be explored. The version that has been implemented in this work can be viewed as a conceptual proof-of-concept. It contains the basic mechanisms behind the idea that is given in section 4.4, but some things can be improved and fine-tuned.

To be able to increase efficiency, automatically selecting the first step, and more importantly when to stop, in the validation process (figure 4.3) can reduce the computation time significantly. Furthermore, by comparing results from section 6.3, in some cases the bump hunting method tends to have a particular high variance. More carefully selecting which box is considered optimal or putting a threshold on the allowed amount of new falsely classified observations may reduce this.

This last statement introduces probably the most interesting potential topic of further research: the scoring function. Throughout this work recall and f-score have been used which, as it turned out, often resulted in quite different results (especially figure 6.21). A more uniform way of selecting the boxes could potentially improve the effectiveness of the overall method. It may for instance often be the overall better choice to build a lot of small boxes as opposed to a few large ones. Greedily optimizing over the current metrics for each box individually will never achieve this situation. A cost proportional scoring may potentially be another way to more precisely score subgroups according to the specific situation and the user's preferences.

Overall, all these possible improvements plead for an overall more uniform implementation to be able to progress from a conceptual idea to an established method for learning with high imbalance. Furthermore, to build more trust in the method, applying it different datasets and testing it with even more classifiers is essential.

Appendix A

Numerical results

method	metric	LR	DT	RF	BAG	ADA
original	recall	0.622(0.037)	0.772(0.04)	0.775(0.033)	0.79(0.03)	0.695(0.053)
	f-1 score	0.727(0.027)	**0.764(0.026)**	**0.85(0.022)**	**0.851(0.022)**	**0.751(0.036)**
	precision	**0.878(0.018)**	**0.758(0.027)**	**0.944(0.018)**	**0.922(0.023)**	**0.818(0.034)**
random	recall	**0.921(0.017)**	**0.912(0.027)**	**0.902(0.021)**	**0.91(0.023)**	**0.932(0.016)**
us	f-1 score	0.08(0.013)	0.031(0.005)	0.095(0.021)	0.077(0.012)	0.056(0.006)
	precision	0.042(0.007)	0.016(0.003)	0.05(0.012)	0.041(0.007)	0.029(0.003)
cnn	recall	0.807(0.021)	0.808(0.029)	0.814(0.018)	0.818(0.02)	0.835(0.03)
	f-1 score	**0.757(0.034)**	0.073(0.016)	0.774(0.018)	0.651(0.06)	0.517(0.061)
	precision	0.714(0.054)	0.038(0.009)	0.739(0.044)	0.546(0.085)	0.378(0.069)

Table A.1: Performance of random undersampling and the condensed nearest neighbors method together with the benchmark results on the original dataset. The overall best score for each classifier is bold.

method	metric	LR	DT	RF	BAG	ADA
original	recall	0.622(0.037)	0.772(0.04)	0.775(0.033)	0.79(0.03)	0.695(0.053)
	f-1 score	0.727(0.027)	0.764(0.026)	**0.85(0.022)**	0.851(0.022)	0.751(0.036)
	precision	0.878(0.018)	**0.758(0.027)**	**0.944(0.018)**	0.922(0.023)	0.818(0.034)
tomek	recall	0.626(0.036)	0.775(0.03)	0.77(0.028)	0.786(0.027)	0.722(0.048)
	f-1 score	0.731(0.027)	0.764(0.019)	0.845(0.016)	**0.851(0.02)**	0.768(0.027)
	precision	0.88(0.018)	0.754(0.018)	0.939(0.023)	**0.928(0.025)**	0.822(0.036)
oss	recall	0.626(0.036)	0.78(0.037)	0.767(0.034)	0.791(0.022)	0.721(0.032)
	f-1 score	0.731(0.026)	0.765(0.02)	0.843(0.021)	0.85(0.017)	**0.77(0.027)**
	precision	0.88(0.02)	0.752(0.035)	0.938(0.015)	0.919(0.027)	**0.826(0.037)**
ncl	recall	**0.737(0.029)**	0.803(0.033)	0.786(0.027)	**0.807(0.036)**	**0.739(0.051)**
	f-1 score	**0.799(0.016)**	**0.772(0.024)**	0.838(0.019)	0.845(0.024)	0.769(0.034)
	precision	0.874(0.021)	0.744(0.028)	0.898(0.027)	0.888(0.02)	0.803(0.039)
enn	recall	0.668(0.048)	0.799(0.033)	0.796(0.026)	0.802(0.03)	0.734(0.046)
	f-1 score	0.759(0.032)	0.768(0.019)	0.848(0.02)	0.841(0.023)	0.769(0.033)
	precision	0.882(0.02)	0.74(0.026)	0.908(0.028)	0.885(0.029)	0.81(0.04)
r_enn	recall	0.674(0.05)	**0.807(0.037)**	**0.802(0.03)**	0.806(0.03)	0.733(0.062)
	f-1 score	0.764(0.033)	0.765(0.03)	0.839(0.022)	0.837(0.023)	0.766(0.044)
	precision	**0.883(0.02)**	0.73(0.048)	0.881(0.024)	0.87(0.024)	0.804(0.033)

Table A.2: Performance of the *data-cleaning* undersampling methods Tomek link removal, one-sided selection, neighborhood cleaning, and (repeated) edited neighbors together with the benchmark results on the original dataset. The overall best score for each classifier is bold.

method	metric	LR	DT	RF	BAG	ADA
original	recall	0.622(0.037)	0.772(0.04)	0.775(0.033)	0.79(0.03)	0.695(0.053)
	f-1 score	**0.727(0.027)**	**0.764(0.026)**	**0.85(0.022)**	**0.851(0.022)**	**0.751(0.036)**
	precision	**0.878(0.018)**	0.758(0.027)	**0.944(0.018)**	**0.922(0.023)**	**0.818(0.034)**
random	recall	**0.916(0.018)**	0.72(0.037)	0.765(0.027)	0.751(0.025)	0.887(0.021)
os	f-1 score	0.118(0.007)	0.738(0.032)	0.842(0.017)	0.799(0.019)	0.169(0.013)
	precision	0.063(0.004)	**0.761(0.059)**	0.936(0.021)	0.855(0.045)	0.093(0.008)
smote	recall	0.905(0.022)	**0.785(0.042)**	**0.826(0.029)**	**0.799(0.027)**	**0.898(0.024)**
	f-1 score	0.151(0.007)	0.532(0.028)	0.848(0.019)	0.716(0.038)	0.159(0.015)
	precision	0.082(0.004)	0.403(0.028)	0.872(0.028)	0.652(0.06)	0.087(0.009)
bsmote	recall	0.871(0.025)	0.764(0.039)	0.788(0.03)	0.786(0.034)	0.868(0.019)
	f-1 score	0.295(0.031)	0.684(0.018)	0.842(0.018)	0.782(0.03)	0.386(0.05)
	precision	0.178(0.023)	0.622(0.036)	0.905(0.022)	0.78(0.046)	0.25(0.042)

Table A.3: Performance of random undersampling and the condensed nearest neighbors method together with the benchmark results on the original dataset. The overall best score for each classifier is bold.

classifier	method	recall	f-score	precision
logistic regression	BH_sin	0.679(0.038)	0.789(0.029)	0.944(0.027)
	BH	0.774(0.029)	0.823(0.027)	0.881(0.04)
random forest	BH_rec	0.823(0.041)	0.873(0.028)	0.932(0.032)
	BH_f1	0.803(0.029)	0.875(0.019)	0.961(0.019)
Adaboost	BH_rec	0.821(0.017)	0.634(0.200)	0.557(0.228)
	BH_f1	0.794(0.036)	0.794(0.028)	0.796(0.047)

Table A.4: Performance of the bump hunting method when using different classifiers

Bibliography

[1] Capgemini and BNP Paribas. World payments report, 2017. . https://www.worldpaymentsreport.com/.

[2] Tej Paul Bhatla, Vikram Prabhu, and Amit Dua. Understanding credit card frauds. *Cards business review*, 1(6), 2003.

[3] Richard Sullivan. The changing nature of us card payment fraud: Issues for industry and public policy. In *WEIS*, 2010.

[4] Bruno Buonaguidi. Credit card fraud: what you need to know. *The Conversation*, 2017.

[5] European Central Bank. Fourth report on card fraud. July 2016.

[6] The nilson report, Issue 1118. https://nilsonreport.com.

[7] Roy Wedge, James Max Kanter, Santiago Moral Rubio, Sergio Iglesias Perez, and Kalyan Veeramachaneni. Solving the "false positives" problem in fraud prediction. *arXiv preprint arXiv:1710.07709*, 2017.

[8] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[9] Leo Breiman, Jerome Friedman, RA Olshen, and Charles J Stone. Classification and regression trees. 1984.

[10] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[11] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[12] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[13] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.

[14] Ajinkya More. Survey of resampling techniques for improving classification performance in unbalanced datasets. *arXiv preprint arXiv:1608.06048*, 2016.

[15] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002.

[16] Xu-Ying Liu and Zhi-Hua Zhou. Ensemble methods for class imbalance learning. *Imbalanced Learning: Foundations, Algorithms, and Applications*, pages 61–82, 2013.

[17] Gary M Weiss. Mining with rarity: a unifying framework. *ACM Sigkdd Explorations Newsletter*, 6(1):7–19, 2004.

[18] Samir Al-Stouhi and Chandan K Reddy. Transfer learning for class imbalance problems with inadequate data. *Knowledge and information systems*, 48(1):201–228, 2016.

[19] Haibo He and Yunqian Ma. *Imbalanced learning: foundations, algorithms, and applications*. John Wiley & Sons, 2013. Chapter 2.

[20] Nathalie Japkowicz. Concept-learning in the presence of between-class and within-class imbalances. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 67–77. Springer, 2001.

[21] Taeho Jo and Nathalie Japkowicz. Class imbalances versus small disjuncts. *ACM Sigkdd Explorations Newsletter*, 6(1):40–49, 2004.

[22] Gary Mitchell Weiss. *The effect of small disjuncts and class distribution on decision tree learning*. PhD thesis, Rutgers University, 2003.

[23] Horst Forster. Card fraud report 2015. Payments Cards & Mobile.

[24] Sori Kang and Kotagiri Ramamohanarao. A robust classifier for imbalanced datasets. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 212–223. Springer, 2014.

[25] Ling Zhuang and Honghua Dai. Parameter optimization of kernel-based one-class classifier on imbalance learning. *Journal of Computers*, 1(7):32–40, 2006.

[26] CX Ling and VS Sheng. Cost-sensitive learning and the class imbalance problem. Springer: Encyclopedia of machine learning, 2008.

[27] Charles Elkan. The foundations of cost-sensitive learning. In *International joint conference on artificial intelligence*, volume 17, pages 973–978. Lawrence Erlbaum Associates Ltd, 2001.

[28] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[29] Gary M Weiss, Kate McCarthy, and Bibi Zabar. Cost-sensitive learning vs. sampling: Which is best for handling unbalanced classes with unequal error costs? *DMIN*, 7:35–41, 2007.

[30] Alexander Liu, Cheryl Martin, Brian La Cour, and Joydeep Ghosh. Effects of oversampling versus cost-sensitive learning for bayesian and svm classifiers. In *Data Mining*, pages 159–192. Springer, 2010.

[31] Chris Seiffert, Taghi M Khoshgoftaar, Jason Van Hulse, and Amri Napolitano. Resampling or reweighting: A comparison of boosting implementations. In *Tools with Artificial Intelligence, 2008. ICTAI'08. 20th IEEE International Conference on*, volume 1, pages 445–451. IEEE, 2008.

[32] Miroslav Kubat, Stan Matwin, et al. Addressing the curse of imbalanced training sets: one-sided selection. pages 179–186, 1997.

[33] Krystyna Napierala and Jerzy Stefanowski. Types of minority class examples and their influence on learning classifiers from imbalanced data. *Journal of Intelligent Information Systems*, 46(3):563–597, 2016.

[34] Ivan Tomek. Two modifications of cnn. *IEEE Trans. Systems, Man and Cybernetics*, 6:769–772, 1976.

[35] Peter Hart. The condensed nearest neighbor rule (corresp.). *IEEE transactions on information theory*, 14(3):515–516, 1968.

[36] D Randall Wilson and Tony R Martinez. Reduction techniques for instance-based learning algorithms. *Machine learning*, 38(3):257–286, 2000.

[37] Dennis L Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, (3):408–421, 1972.

[38] Ivan Tomek. An experiment with the edited nearest-neighbor rule. *IEEE Transactions on systems, Man, and Cybernetics*, (6):448–452, 1976.

[39] Jorma Laurikkala. Improving identification of difficult small classes by balancing class distribution. In *Conference on Artificial Intelligence in Medicine in Europe*, pages 63–66. Springer, 2001.

[40] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[41] BX Wang and Nathalie Japkowicz. Imbalanced data set learning with synthetic samples. In *Proc. IRIS Machine Learning Workshop*, volume 19, 2004.

[42] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. Borderline-smote: a new over-sampling method in imbalanced data sets learning. In *International Conference on Intelligent Computing*, pages 878–887. Springer, 2005.

[43] Andrew E Jaffe, Peter Murakami, Hwajin Lee, Jeffrey T Leek, M Daniele Fallin, Andrew P Feinberg, and Rafael A Irizarry. Bump hunting to identify differentially methylated regions in epigenetic epidemiology studies. *International journal of epidemiology*, 41(1):200–209, 2012.

[44] Rui Jiang, Hua Yang, Fengzhu Sun, and Ting Chen. Searching for interpretable rules for disease mutations: a simulated annealing bump hunting strategy. *BMC bioinformatics*, 7(1):417, 2006.

[45] Jerome H Friedman and Nicholas I Fisher. Bump hunting in high-dimensional data. *Statistics and Computing*, 9(2):123–143, 1999.

[46] Ad Feelers. Rule induction by bump hunting. Lecture notes from course *Advanced Data Mining*, Universiteit Utrecht, 2011/2012.

[47] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[48] Andrea Dal Pozzolo, Olivier Caelen, Reid A Johnson, and Gianluca Bontempi. Calibrating probability with undersampling for unbalanced classification. In *Computational Intelligence, 2015 IEEE Symposium Series on*, pages 159–166. IEEE, 2015.

[49] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[50] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

[51] Sci-kit learn - random forest documentation. `http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`.

[52] Documentation randomForest R library. `https://cran.r-project.org/web/packages/randomForest/randomForest.pdf`.

[53] Documentation Adaboost R library. `https://cran.r-project.org/web/packages/gbm/gbm.pdf`.

[54] Jurian Baas. Global models and local patterns for crime prediction from weather data. *Master Thesis Artificial Intelligence, Universiteit Utrecht*, 2017.