SEMI-SUPERVISED LEARNING FOR IMBALANCED CLASSIFICATION WITH LABEL SCARCITY IN THE DOMAIN OF FINANCIAL FRAUD DETECTION

by

LUUK VAN SON

836381

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Business Analytics and Operations Research

Tilburg School of Economics and Management

Tilburg University

Supervisor

Dr. Moslem Zamani

Second reader

Dr. Maxence Delorme

August 2022

ABSTRACT

The application of statistical methods for financial fraud detection often appears to be hampered by label scarcity and existing class imbalance within the utilized dataset. Now, semi-supervised learning has shown to be potentially effective in mitigating negative outcomes originating from these data properties. This study investigates the performance of semi-supervised learning method, self-training within the context of financial fraud detection. To this end, we apply a customized self-training algorithm called SelfTrain on input classifiers logistic regression, linear discriminant analysis, random forests and linear support vector machines. We evaluate the improvement ability of SelfTrain with regard to the classifier's supervised baseline results for a low, medium and high label scarcity scenario. Moreover, we specifically address class imbalance by investigating the impact of data augmentation method SMOTE on the performance results in the medium label scarcity scenario. In this study, we find that for linear discriminant analysis, random forests and linear SVM, SelfTrain is consistently able to improve upon supervised baselines, where the most robust improvement is observed for random forests. Lastly, we find that SMOTE is not able to significantly boost the performance of SelfTrain.

Keywords: Class imbalance, label scarcity, supervised learning, semi-supervised learning, self-training, input classifier, data augmentation

ACKNOWLEDGEMENTS

First of all, I would like to give my warmest thanks to my thesis supervisor Dr. Moslem Zamani, who gave me great guidance and was always available for advice during my thesis project. His unconditional optimism and expertise on the subject certainly carried me through this thesis project. Without him, I could not have done it. In anticipation of my defense, I would also already like to thank Maxence Delorme for making himself available as second reader.

Next, I would like to show my grattitude to my colleague Raffaele Battilomo from RiskQuest, with whom I spent several hours having fruitful discussions on the subject matter.

Last but not least, I want to thank my family and friends who always showed great love and support to me in the good and bad times that I have encountered the last couple of months. You are the best.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. Introduction

Nowadays, financial fraud detection plays a major role in countering financial crime. Financial fraud can be defined as "the intentional use of illegal methods or practices for the purpose of obtaining financial gain" and can be divided into three subcategories: bank fraud, corporate fraud, insurance fraud and cryptocurrency fraud [41]. Over the years, manual financial fraud detection has become more and more superfluous due to upcoming automated processes. Whereas manual fraud detection is mainly characterized by audit practices, new automated processes predominantly include statistical and computational methods. Especially the interest in statistical methods has increased significantly due to the rise of big data. According to Quah and Sriganesh [32], the use of statistical methods in fraud detection has many advantages when it is applied in a framework that classifies suspicious transactions for further investigation. They [32] argue that this approach results in reduced time, complexity and costs for processing a transaction. The majority of these statistical methods are supervised classification methods utilized to determine the likelihood of a transaction being fraudulent or genuine based on class labels and classifying it as such [41].

A problem that often arises in fraud detection in combination with supervised classification is *label scarcity*, which refers to a situation where only a small fraction of the data is actually labeled [26]. This scarcity occurs due to the fact that labeling data is an expensive and time-consuming exercise [24]. Label scarcity can make supervised learning challenging, due to the limited size of the training data, which may deteriorate the generalization ability of the classifier. This problem may be amplified when the financial fraud data is high-dimensional, due to the curse of dimensionality, referring to a situation where the data contains too many features relative to amount of observations [18].

Another typical problem in fraud detection is a skewed class distribution of the data, where one class is over-represented compared to the other class, also referred to as *class imbalance* [15]. In general, the under-represented class is called the minority class, while the dominant class is called the majority class [20]. First of all, it is argued by Ling and Sheng [23] that class imbalance is not problematic in itself. However, the assump-

tions underneath a specific classification application can make it a problem. Note that a traditional classifier is built upon the assumption of maximizing accuracy with equal misclassification costs among both classes, making the classifier biased towards the majority class in case of imbalance [31]. Yet, in financial fraud detection, the minority (fraudulent) class happens to be the class of interest. This imposes the risk of making costly misclassifications when using a traditional cost-insensitive classifier.

A relatively new type of statistical learning, called semi-supervised learning (SSL), seems to have the potential to tackle the problem of label scarcity and class imbalance simultaneously. Zhu and Goldberg [43] define semi-supervised learning as the learning paradigm concerned with the study of how computers and natural systems learn in the presence of both labeled and unlabeled data. SSL combines the unlabeled data with the labeled data in a specific learning method, mitigating the need for labeling all the data. Recent study suggests that exploiting unlabeled data trough SSL might also improve class imbalanced learning [42]. Furthermore, SSL can be combined with other methods to address both problems at the same time. These methods include among others over-sampling, under-sampling and cost-sensitive learning [40].

Through the recent years, several studies focused on the application of SSL in financial fraud detection and explicitly addressed the combination of label scarcity and class imbalance. Casalino et al. [4] proposed a clustering based algorithm in which an incremental semi-supervised fuzzy C-means was used to detect credit card fraud. Dzakiyullah et al. [11] implemented the use of T-SNE, a statistical method for visualizing high-dimensional data, and improved it with autoencoders. Furthermore, Melo-Acosta et al. [25] investigated balanced random forests through a SSL approach called co-training. Lastly, Salazar et al. [34] constructed a three stage SSL framework including a Fourier transform-based class balancing method in the first stage and a SSL model called self-training in the second and third stage. Even though these studies have shown promising results, the topic is still relatively under-examined.

In this study, we propose an exploratory investigation of the performance of self-training complemented with data re-balancing applied in a financial fraud detection context. Self-training is a supervised-wrapper method that uses its predictions on unlabeled data to teach itself through an iterative process of training, testing and label updating with pseudo-labels which stops whenever convergence is attained [43]. Logically, self-training depends on a supervised learning method that is selected as input. The main goal of this study is to evaluate whether self-training can improve upon supervised baselines. In addition, we want to investigate how separately addressing class imbalance by means of data re-balancing may affect the performance results. As a corollary of addressing these

goals, a new detailed and operational SSL framework can be established, which may serve as a foundation for further research.

In the first stage of the framework, a data preprocessing step is executed in order to make the data suitable for SSL. Subsequently, synthetic minority oversampling technique (SMOTE) is applied as data re-balancing method. SMOTE creates synthetic minority datapoints based on the original minority datapoints and their K-nearest neighbours [5]. Within the domain of financial fraud detection, SMOTE has proven to be an effective method. SMOTE was applied in addition to varying machine learning models in order to solve financial fraud detection problems [21, 27, 37]. They concluded that utilizing SMOTE as an oversampling technique can result in a significant improvement in predictive power of the machine learning models. In the second stage, input classifiers linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), logistic regression, linear support vector machines (linear SVM) and random forests are trained and tested as a initialization and benchmarking step for the self-training algorithm. First of all, logistic regression is selected because of its widespread use in the industry [41]. LDA and QDA are chosen since they have shown to be effective in combination with self-training for improving upon supervised baseline results within credit card fraud detection [34]. Lastly, SVM and random forests are generally perceived as high-performing models for detecting financial fraud and are therefore included as competing models [41]. In the third stage, the self-training algorithm is executed in an iterative way. The implementation of the algorithm is based on [34], from which the fundamental elements of self-training are incorporated. Moreover, some extensions are implemented in the algorithm. First, an extension on pseudo-label selection is adopted from [29]. Second, a gradual learning option is implemented, which enables the algorithm to gradually adapt to the label updates. Lastly, a performance metric maximizer is constructed that selects the classification model at the point where the highest metric score is attained during the self-training algorithm. The metric to be maximized is referred to as the principal performance metric. For convenience, we denote our constructed self-training algorithm by SelfTrain.

In this study, the SSL framework is applied on a credit card fraud dataset, consisting of 284,807 transactions of which 492 are labeled as fraudelent. This implies that the data distribution is heavily skewed, which is in accordance with the earlier mentioned class imbalance property of financial fraud data. The performance of SelfTrain is evaluated and compared among the input classifiers by means of three different performance metrics for three different scenarios of label scarcity: low, medium and high. In addition, supervised baseline results, referred to as SL-base, are reported and compared with the SelfTrain results. Lastly, the impact of SMOTE is evaluated in order to gain insight on

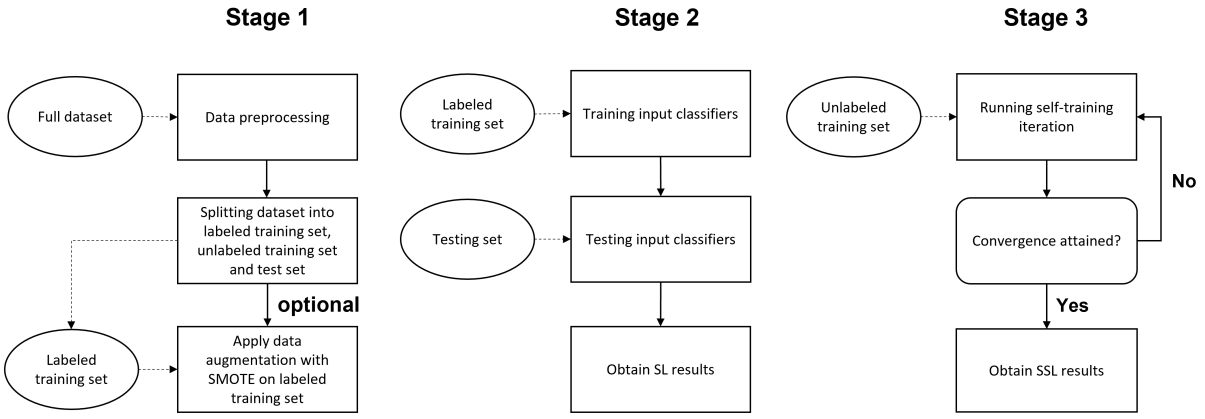
how re-balancing the data with SMOTE affects the results. In this study, the performance evaluation is based on assessing the models ability of detecting true fraudulent transactions, while limiting the amount of detected false fraudulent transactions. Under a fixed discrimination threshold, the first ability can be reflected by the metric recall, whereas the latter ability can be captured by the metric precision [35]. However, the relative importance of both recall and precision strongly depends on the business related context, which makes it difficult to fix the discriminant threshold at a certain value. For this reason, the principal performance metric is chosen to be the threshold-free metric average precision (AP), which is characterized by its ability to properly summarize precision against recall under different thresholds by a single number. For the sake of comparison, area under the receiver operating characterstic curve (ROC AUC) is included as a second metric, which has the same summarizing property as AP, albeit for recall against false positivity rate (FPR). Lastly, Brier score is included as the third metric in order to assess the accuracy of the predicted probabilities. Finally, a sensitivity analysis is conducted, in which the robustness of the SelfTrain is assessed on the basis of optimized SelfTrain parameters. These parameters include sample size (to be selected from the unlabeled data), confidence threshold (above which a pseudo-labeled datapoint is added to the training data) and a boolean paramater for implementing gradual learning.

The study is structured in the following way. In Chapter 2, the methodology framework is set out consisting of three stages which are elaborated in the subsequent subsections. Next, Chapter 3 outlines the selected performance metrics and the rationale behind this selection. In Chapter 4, the utilized financial fraud dataset is analyzed and an explanation of the data composition for results generation is provided. The evaluation of the results and sensitivity analysis are presented in Chapter 5. Lastly, Chapter 6 concludes the study by summarizing and interpreting the results, discussing the study's limitations and providing suggestions for further research.

2. Methodology

The purpose of this study is to investigate the performance of SSL with the scope of addressing the two problems related to financial fraud data: class imbalance and label scarcity. As mentioned in the introduction, the study will be conducted on the basis of a three stage SSL framework illustrated in Figure 2.1. The first stage includes a data preprocessing step, in which the data is scaled. Then, the full preprocessed dataset is partitioned into a labeled training set, unlabeled training set and test set. In the last step of this stage, synthetic data is created using SMOTE, which is a data augmentation method that can be used to re-balance the data to a specified degree. SMOTE is solely applied on the labeled training data, since it is believed that this resembles most with real-world applications.

Figure 2.1 A three stage semi-supervised learning framework.



The second stage can be considered as a preparatory stage for SelfTrain. It contains a supervised training and testing procedure using the set of input classifiers that will be integrated into SelfTrain. The set of selected input classifiers consists of logistic regression, LDA, QDA, random forests and linear SVM. The supervised learning results (SL-base) from this stage are used both as an initialization and a benchmark for SelfTrain. In the third and final stage, SelfTrain is executed in an iterative manner until convergence[1] is

[1]Note that in this context, convergence is not specifically defined and can refer to any stopping criterion of the algorithm.

reached, after which the results are obtained.

The following sections will cover some deeper insights on the most comprehensive steps within the framework in a logical order. First, we elaborate on how synthetic data is created with SMOTE. Next, SSL is further explained where the main focus lies on self-training and the construction of our specific SelfTrain algorithm. Finally, the supervised input classifiers which are used as input for SelfTrain are set out.

2.1 SMOTE

SMOTE is an over-sampling based data augmentation technique in which synthetic datapoints of the minority class are created using interpolation between real datapoints that are close in the feature space [5]. It can be used to restore the balance within a data set to a specific desired degree. The procedure of SMOTE is as follows. Initially, the amount of synthetic datapoints to be created is determined by choosing a value for N such that it is an integer multiple of 100. Then an iterative process starts. First, a datapoint x_i is randomly selected from the set of minority class datapoints T, after which its K nearest minority class neighbors (KNN) are determined. Subsequently, a nearest neighbour nn_R of x_i is randomly selected such that a line segment can be constructed between x_i and this randomly selected neighbour nn_R. On this line segment, a random convex combination can be derived. For each minority class datapoint x_i, this process is repeated $\frac{N}{100}$ times such that one obtains $\frac{N}{100}$ random convex combinations . Eventually, the output consists of $\frac{N}{100} * T$ synthetic datapoints. The formal SMOTE procedure is captured in Algorithm 1 proposed in [17]. Note that in lines 1-4 of Algorithm 1 the case of N being lower than 100 is handled. In this situation, the output must be less than T and so the parameters must be defined accordingly. To this end, the T minority class datapoints are put in a random order after which T is set to $T * (N/100)$, such that T becomes a fraction of its previous value. Furthermore, N is set to 100 in line 4 and then becomes equal to 1 by line 5. The reason for this step is that per minority class datapoint that we visit, we want to extract exactly one synthetic minority class datapoint. Furthermore, the process deriving convex combinations for each of the minority class datapoints in lines 6 - 11 of Algorithm 1 can be explained more clearly on the basis of Figure 2.2. Note that point x_i represents the i-th minority class datapoint that is selected. As can be seen in Figure 2.2a, nearest neighbours of x_i are determined (denoted by $\{nn_1, ..., nn_5\}$) and connected with x_i by different line segments. Subsequently, convex combinations can be constructed on these line segments, of which 5 examples are given by the red dots. The effect of one such procedure on the data can be observed in Figure 2.2b.

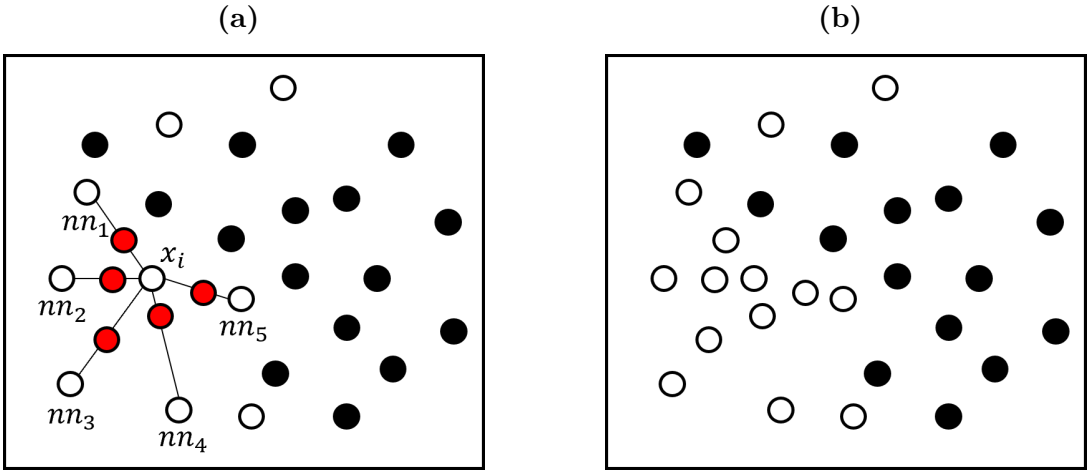
Algorithm 1: SMOTE

Input: Amount of minority class datapoints T
Synthetic datapoints factor N (as multiple of 100)
Amount of nearest neighbours K

Output: Array *Synthetic* consisting of $(N/100)*T$ synthetic minority class datapoints

```
if N ≤ 100 then
    Randomize the T minority class datapoints
    T = (N/100) * T
    N = 100
N = (int)N/100
KNNarray[][]: array for saving KNN for each minority class datapoint
Synthetic[][]: array for saving N synthetic datapoints for each minority class datapoint
for i = 1,...,T do
    Determine KNN for minority class datapoint x_i, save KNN in KNNarray[i]
    while N ≠ 0 do
        Generate a random number R out of {1,...,K}
        Choose the R-th element from KNNarray and call it nn_R
        Make convex combination between x_i and nn_R and save it in Synthetic[i]
        Let N = N − 1
```

Figure 2.2 A graphical representation of SMOTE showing a minority class datapoint, 5 nearest neighbours and 5 potential convex combinations (a) and the re-balancing effect of SMOTE (b).

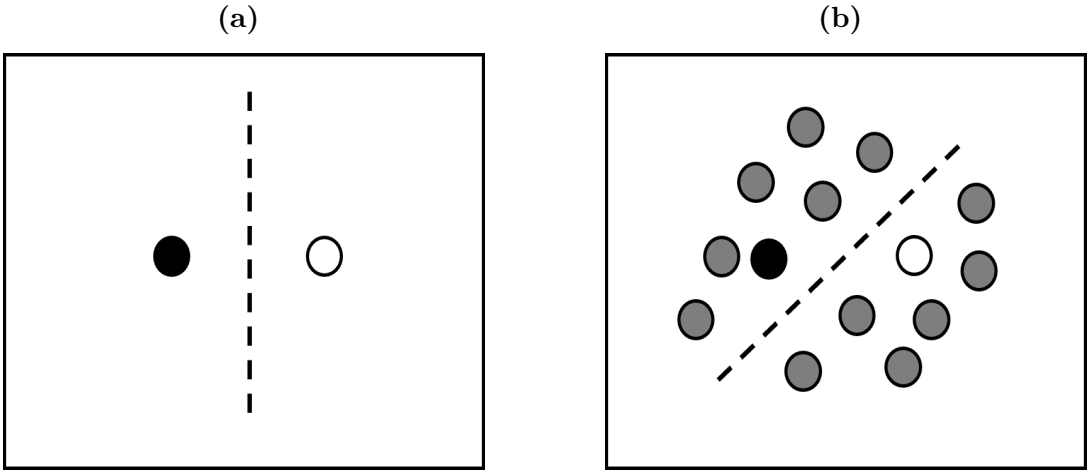


2.2 Semi-supervised learning

In a lot of real-world applications of statistical learning such as financial fraud detection, only a small part of a dataset is labeled, such that one ends up with a (possibly large) amount of unlabeled data [24]. This is often perceived as a problem, since it is a generally accepted understanding that training (complex) statistical learning models on small but high-dimensional datasets results in models with high variance which often leads to poor prediction performances [18]. This phenomenon is also known as 'overfitting'. The case of having (much) more unlabeled data than labeled data available raises the question whether one could exploit this unlabeled data together with the small portion of labeled data in order to improve prediction performance of a specific learning method. Semi-supervised learning (SSL) is a learning paradigm that addresses this question. SSL operates on the cutting edge of supervised and unsupervised learning in the sense that it extends one learning paradigm, either supervised or unsupervised, with another one [43]. Conventionally, statistical learning is only applied within either a supervised or unsupervised paradigm. It is crucial to first understand these concepts in order to get a grasp on the concept of SSL itself. In short, supervised learning represents learning through labeled datapoints, where the goal is to fit a model that relates feature values to labels such that labels can be predicted with future observations of features (a more detailed explanation of supervised learning can be found in Section 2.4). On the other hand, unsupervised learning can be considered as a form of pattern recognition of unlabeled datapoints in order to identify structures in the data, without the involvement of any supervising target label [2].

SSL combines these two learning paradigms by exploiting both the availability of labeled as well as unlabeled data. More specifically, it takes advantage of the unlabeled data by adding labels to unlabeled data observations based on the structure of the unlabeled data and knowledge that is obtained from the small amount of labelled data. By doing so, an improved statistical learning model can be build, without the need for manually labeling additional data, which may to be a very costly and time consuming exercise [24]. The working of this process can be explained on the basis of Figure 2.3. In Figure 2.3a, it can be seen how a supervised learning method would set the decision boundary based on one positive (black dot) and one negative (white dot) datapoint. If in addition, unlabeled data would become available, an unsupervised learning method named clustering could be used to separate regions. Clustering divides a set of unlabeled datapoints into a specific amount of clusters, such that the datapoints within the seperate clusters are similar to a certain extent. Finally, pseudo-labels could be added to the created clusters in accordance with the location of the original labeled data (Figure 2.3b).

Figure 2.3 Example of semi-supervised learning in which both labeled and unlabeled data (grey dots) is exploited to find an improved decision boundary.



SSL can be employed based on two different goals. The first goal is to assign the right pseudo-labels to the unlabeled data, which is also known as transductive[2] learning. The second goal is to fit a classifier in order to accurately predict future data, also referred to as inductive[3].

In this study, semi-supervised classification is of relevance, with the aim of training a classifier using both labeled data $\{(x_i^l, y_i^l)\}_{i=1}^n$ and unlabeled data $\{(x_i^{ul}, y_i^{ul})\}_{i=n+1}^{n+u}$, with $\{y_i^{ul}\}_{i=n+1}^{n+u}$ being unknown. Naturally, the use of SSL only makes sense when the distributions of labeled data and unlabeled are approximately the same. Otherwise, we would be building a biased classification model which often leads to a poor prediction performance [6]. Note that randomly selecting an SSL method will not necessarily lead to an improved performance compared to supervised learning. Moreover, it can even lead to a worsened performance. Indeed, the effectiveness of SSL is largely determined by how well underlying assumptions for SSL are met [43]. While each SSL method has its own unique assumptions, some general assumptions can be identified [38]. The first one is the *smoothness assumption*, which states that for two nearby datapoints in the predictor space, the corresponding labels are also nearby. The second assumption is the *low-density assumption*, implying that the decision-boundary of a classifier lies in a low-density region of the predictor space. Note that here, the true distribution of the data is considered. The third assumption is called the *manifold assumption* and it is based on the fact that predictor space can be decomposed into multiple lower-dimensional manifolds that con-

[2]Transduction refers to inference from observed examples to specific examples [14].

[3]Induction refers to inference from observed examples to general rules [14].

tain all the datapoints. The assumption states that datapoints on the same manifolds share the same label. The final assumption is the *cluster assumption* and is argued to be a generalization of the previous assumptions. It states that datapoints from the same cluster tend to belong to the same class as well.

Some well-known SSL methods include self-training, co-training, Gaussian mixture models (GMM) and semi-supervised support vector machines (S3VM). In the next section, we elaborate on self-training as it serves as the principal method in our proposed SSL framework.

2.3 Self-training

Self-training[4] is one of the oldest existing SSL methods and it was first introduced by Scudder [36]. This method uses prediction results from a specific supervised learning method on an unlabeled set of data as additional training data. By applying this concept in an iterative way the learning method teaches itself based on the supervised learning results [43]. First, the supervised learning method is trained with labeled training data. Next, unlabeled data points are labeled as pseudo-labels and added to the original training data. This procedure is then repeated iteratively. A formal representation of the above mentioned process is given by Algorithm 2.

Algorithm 2: Self-training

Input: Supervised learning method f

Data: Training data $\{(\mathrm{x}_i, y_i)\}_i^n$ as TR

Unlabeled data $\{x_i\}_{i=n+1}^{n+u}$ *as* U

1 **Repeat**

2 Train f with TR

3 Apply classifier f to U and obtain pseudo-labels P

4 Take subset S from U and add $UP_S = \{(x, f(x)) \mid x \in S)\}$ to TR

Apart from the general SSL assumptions, a specific underlying assumption for self-training is that the initial predictions, originating from the supervised learning method that is used, are correct. From this assumption, a drawback of the method becomes immediately clear: a bad performing supervised learning method may result in a bad performing self-training algorithm, since the wrong predictions from the supervised learning method are reinforced within the self-training algorithm. The phenomenon in which

[4]In the literature also referred to as self-learning or pseudo-labeling.

the data is overfitted with incorrect pseudo-labels is referred to as *confirmation bias* by [1]. Second, a potentially problematic aspect of self-training is convergence. From now on, with convergence in the context of self-training, we mean a sequence of performance values that approaches an improved and fixed performance value. To our best knowledge, the literature does not provide any mathematical proof for this type of convergence to occur within self-training. In fact, it is mathematically not yet proven when and why self-training algorithms work in general [28].
A self-training algorithm can be implemented in various ways. In this study, the algorithm developed in [34] is chosen as a conventional implementation and can be observed in Algorithm 3.

Algorithm 3: Self-training

Input: Classifier f

Data: Training data $\{(\mathrm{x}_i, y_i)\}_{i=1}^{n}$ as TR
Testing data$\{(x_i, y_i)\}_{i=n+1}^{n+m}$ as TE
Unlabeled data$\{(x_i, y_i)\}_{i=n+m+1}^{n+m+u}$ as U

Let $C = False$
while $C =$ *False* **do**
    Train classifier f with TR
    Apply classifier f to U and obtain pseudo-labels P
    Take subset S from U
    Add $UP_S = \{(x, f(x)) \mid x \in S)\}$ to TR
    Check convergence:
    if convergence is $True$ **then**
        $C = True$

Since we are dealing with a classification problem, the supervised learning method that is used within the algorithm has to be some classifier *f*. Then, classifier f is trained with the training set TR, after +which the unlabeled data U is labeled and a set of pseudo-labels P is obtained. This set P in combination with the corresponding subset U can be considered as the semi-supervised data set UP. Finally, a subset S is taken from U and this set and the corresponding pseudo-labels which together form UP_S are then added to TR. This process can be repeated until convergence is attained.

While all the steps are coherently formulated, two important elements in the algorithm remain ambiguous and must be clarified before implementing our self-training algorithm based on Algorithm 3. First, it does not become clear in what way a subset S is selected

from the unlabeled set U during an iteration. Therefore, we select subsets based on a predefined sample size, where the samples with the most reliable pseudo-labels are then added the training set. This procedure is based on the algorithm introduced in [29]. Second, it is assumed that the algorithm is convergent. As mentioned earlier, assuming self-training to be convergent may not always be valid. Therefore, we implemented a maximum iteration criterion and a maximizer that selects the classifier that scores highest on a prespecified performance metric, which we refer to as the *principal performance metric*. By applying the above mentioned modifications on Algorithm 3, we obtain Algorithm 4.

Algorithm 4: Self-training

Input: Classifier f
Principal performance metric m
Sample size z
Confidence threshold ϵ
Gradual learning option w

Data: Training data $\{(x_i, y_i)\}_{i=1}^{n}$ as TR
Testing data $\{(x_i, y_i)\}_{i=n+1}^{n+m}$ as TE
Unlabeled data $\{x_i\}_{i=n+m+1}^{n+m+u}$ as U

Output: classifier f^* corresponding to principal performance metric score b_m^*

Let $i = 0$, $C = False$ and $I = \frac{|U|}{z|U|}$.
$B_m[]$: array for saving metric scores in each iteration
while $i < I$ **do**
    $i = i + 1$
    Train classifier f with TR
    Apply classifier f to TE and save metric score b_m in B_m
    Apply classifier f to U and obtain probabilities PR_U
    Sort probabilities PR_U and obtain PR_U^{sorted}
    Select $z|U|$ highest probabilities above ϵ; obtain PR_U^{high}
    Select partition S from U corresponding to PR_U^{high}
    Add $UP_S = \{(x, f(x)) \mid x \in S)\}$ to TR
Determine maximum metric score b_m^* from B_m

Initially, sample size z and confidence threshold ϵ and gradual learning option w are chosen as input parameters and f is selected as input classifier. Next, The cardinality of U is divided by the product of sample size z and the same cardinality of U in order to obtain the maximum number of iterations I. Then an iterative process starts. First classifier f is trained with the training set TR and subsequently applied on the testset TE

and on the unlabeled data U. The application on TE results in a performance score b_m based on the principal performance metric that is chosen. This score can be deduced using the predicted output and real output of test set TE. As mentioned in the introduction, average precision is selected as the principal performance metric for this study and is further explained in Chapter 3. Now, the application from f on U results in a set of pseudo-label probabilities PR_U, which are the probabilities that a classifier assigns to the unlabeled observations in U, representing the likelihood of an observation belonging to a certain class. These pseudo-label probabilities are then sorted and the $z|U|$ highest probabilities are selected resulting in PR_U^{high}. Now, a partition S is selected from the unlabeled data U (without replacement), such that the datapoints in S correspond to the probabilities in PR_U^{high}. Then this set S is labeled with classifier f accordance with PR_U^{high} and added to the training set. If the maximum number of iterations I is not yet reached, a new iteration is started by training classifier f with the new training set. Eventually, when the maximum number of iterations I is attained, the maximum principal performance metric score b_m^* is determined and the corresponding classifier f^* is returned. The maximizer is constructed in such a way that it neglects the score in the first iteration, where the model is trained without pseudo-labels. Note that the first iteration of the algorithm is in fact the second stage in the SSL framework (see Figure 2.1) which serves as an initialization providing supervised baseline scores SL-base.

Every iteration within the self-training algorithm must have an associated loss function to minimize a specified fitting error corresponding to the selected input classifier f. Lee [22] defines the general loss function for self-training as

$$\text{Loss}(i) = \frac{1}{n}\sum_{j=1}^{n} \text{Loss}(y_j, f_j) + S(i)\frac{1}{u}\sum_{j=n+1}^{n+u} \text{Loss}(y'_j, f'_j) \tag{2.1}$$

The first part of the function represents the original loss function for fitting a labeled batch of data, while the second part represents the loss function for fitting the unlabeled batch of data with pseudo-labels (with j denoting an observation). The weight parameter $S(i)$ can be adjusted in every iteration to define the impact of the pseudo-labels on the total loss function. It is suggested in [22] that slowly increasing the weight parameter $S(i)$ in the loss function might help to avoid ending up in local minima during optimization process of an iteration's loss function. Therefore, an additional simple gradual learning option is included in the algorithm which gradually increases $S(i)$ in every iteration such

that the pseudo-labels slowly gain importance during the algorithm:

$$S(i) = \frac{i}{I} \tag{2.2}$$

where i denotes the current iteration and I the maximum number of iterations in the self-training algorithm. This will only be used as a parameter in combination with input classifiers logistic regression, linear SVM and random forests, due to implementation limits.

SelfTrain can be optimized using three hyperparameters. The first parameter is sample size z, which determines the portion of the unlabeled dataset U selected for semi-supervision in each iteration. The second parameter is the confidence threshold ϵ above which samples are selected from the unlabeled dataset to be potentially added to the training data. Finally, the gradual learning option from (2.2) is implemented as a boolean parameter w.

2.4 Supervised Learning

Supervised learning is a statistical learning paradigm characterized by learning through example with a training set in which each datapoint consists of prediction measurement and a corresponding output measurement [18]. The goal of supervised learning is to fit a model that captures the relationship between observed prediction measurements and output measurements, such that future output measurements can be predicted based on new observations. Within the domain of supervised learning, classification is the method for predicting qualitative outputs, also referred to as classes. These classes are labeled with specific numerical codes, depending on the classification method that is used. As this study focuses on the application of financial fraud detection where we want to correctly classify fraudulent and genuine transactions, we are dealing with a binary classification problem. Hence, we have two classes of interest which are the fraudulent and genuine classes, labeled by the numerical codes 1 and 0, respectively. We denote the class variable by Y, where $Y \in L$ with $L = \{0,1\}$. Furthermore, each financial transaction has certain characteristics, e.g. customer ID, transaction amount or country. These characteristics are the features of an observed transaction and serve as prediction measurements. The set of feature variables is denoted by X, where $X \in \mathbb{R}^{\mathrm{K}}$ and $K \geq 1$. Taking this all together, a training set that consists of observed financial transactions can be represented by $\{(x_i, y_i)\}_{i=1}^{n}$, where x_i and y_i represent the i-th observed transaction with correspond-

ing feature values (transaction characteristics) and class label (fraudulent or genuine) and where n represents the total number of observations (transactions). By applying a specific classification method on the training set, we can fit a model that relates the transaction features to the classes fraudulent and genuine. Note that a distinction can be made between probabilistic and non-probabilistic classification methods. The first assigns classes to observations based on probabilities and it depends on a so called discrimination threshold, which is a value that dichotomizes the result of a classifier to a binary decision. A non-probabilistic classification rather separates the feature space and assigns a class to an observation based on its location in this space. In the next subsections, the classification methods which are used as input for SelfTrain are discussed.

2.4.1 Logistic regression

Consider feature variable $X \in \mathbb{R}^{\mathrm{K}}$ and the label variable $Y \in \{0, 1\}$. In logistic regression, the goal is to predict Y based on conditional probability $p_1(x) = p(Y = 1|X = x)$, with an output between 0 and 1 [18]. Logistic regression models this probability based on odds. Odds can be interpreted as a substitute for probabilities and are calculated as the ratio of the number of expected events resulting in $Y = 1$ and $Y \neq 1$. For example, the odds of $p_1(x) = 0.1$ is 1/9, indicating a 1 out of 10 expected occurrence of $Y = 1$ under the condition of $X = x$. Logistic regression defines the odds as

$$\frac{p_1(x)}{1 - p_1(x)} = e^{\beta_0 + \beta^T x}. \tag{2.18}$$

where scalar β_0 and vector $\beta \in R^K$ are the model coefficients to be estimated using training data. Now, writing (2.18) in logarithmic form gives

$$\log(\frac{p_1(x)}{1 - p_1(x)}) = \beta_0 + \beta^T x. \tag{2.19}$$

Clearly, the log odds are linear in x, so one unit change in x_k results in a log odds change of β_k. By taking the exponent on both sides of equation (2.19) and doing some manipulation, we get the logistic function

$$p_1(x) = \frac{e^{\beta_0 + \beta^T x}}{1 + e^{\beta_0 + \beta^T x}}. \tag{2.20}$$

Since the rate of change of $p_1(x)$ now depends on the current value of x, the relationship between $p_1(x)$ and x is now non-linear. However, with $\beta > \vec{0}$, an increasing x still results

in an increasing $p(x)$. In order to turn the logistic function into a classifier, the coefficients β_0 and $\beta \in R^K$ are estimated by fitting the logistic function to the training data using the method of maximum likelihood. This results in $\hat{\beta}_0$ and $\hat{\beta} \in R^K$ with a corresponding predicted probability function $\hat{p}_1(x)$. After applying logistic regression on a training set, a new set of test observations $\{x_i\}_{i=n+1}^{n+m}$ can be plugged into the model. The model then returns a set of probabilities and allocates observation x_i to class 1 if $\hat{p}_1(x_i)$ is larger than the discrimination threshold and to class 0 otherwise.

2.4.2 Linear discriminant analysis

Linear discriminant analysis (LDA) is a supervised learning method based on a generative model in which Bayes' theorem is employed to find label probabilities. Bayes' theorem states that the probability of event A to occur given that event B has occurred can be calculated as follows:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \tag{2.4}$$

The posterior probability $P(A|B)$ is considered as the element of interest, which can be calculated using prior knowledge in the form of the so-called likelihood $P(B|A)$, denoting the probability that B occurs given that A has occured, and prior probability $P(A)$ [3]. Furthermore, note that one can rewrite $P(B)$ as the sum of prior probabilities times its likelihood, that is

$$P(B) = \sum_{i \in I} P(B|A = i)P(A = i). \tag{2.5}$$

LDA models the distribution of the features X, given that it belongs to a certain class y [18]. This distribution is denoted by $f_l(x) = P(X = x|Y = l)$ and can be considered as the likelihood function. The prior probabilities are denoted by $\theta_l = P(Y = l)$. Bayes' theorem is then used in combination with the likelihood function and prior probability to estimate the posterior probability

$$P(Y = l|X = x) = \frac{P(X = x|Y = l)P(y = l)}{\sum_{j \in L} P(Y = j)P(X = x|Y = j)}. \tag{2.6}$$

In order to do so, the prior probabilities θ_l and the likelihood functions $f_l(x)$ are estimated. For the prior probabilities this is done by taking the training set $\{(x_i, y_i)\}_{i=1}^{n}$ and dividing the number of observations belonging to class $y = l$ by the total amount of observations

in the training set

$$\theta_l = \frac{|\{Y|Y = l\}|}{n}. \tag{2.7}$$

For the estimation of the likelihood functions, it is first assumed that X has a multivariate Guassian distribution, $X \sim N(\mu, \Sigma)$. An important remark is that mean μ is assumed to be class specific, while the covariance matrix Σ is assumed to be equal among the classes in L. This results in the following notation of the likelihood function:

$$f_l(x) = \frac{1}{2\pi|\Sigma|^{\frac{1}{2}}} exp\Big(-\frac{1}{2}(x-\mu_l)^T\Sigma^{-1}(X-\mu_l)\Big). \tag{2.8}$$

Since μ_l and Σ are not known beforehand, these parameters are estimated from the training data giving $\hat{\mu}_l$ and $\hat{\Sigma}$. The estimated posterior probability $P(y = l|X = x)$ is then denoted by $\hat{p}_l(x)$, such that we get

$$\hat{p}_l(x) = \frac{\theta_l \frac{1}{2\pi|\hat{\Sigma}|^{\frac{1}{2}}} exp\Big(-\frac{1}{2}(x-\hat{\mu}_l)^T\Sigma^{-1}(x-\hat{\mu}_l)\Big)}{\sum_{j\in L}\theta_l \frac{1}{2\pi|\hat{\Sigma}|^{\frac{1}{2}}} exp\Big(-\frac{1}{2}(x-\hat{\mu}_l)^T\hat{\Sigma}^{-1}(x-\hat{\mu}_l)\Big)}. \tag{2.9}$$

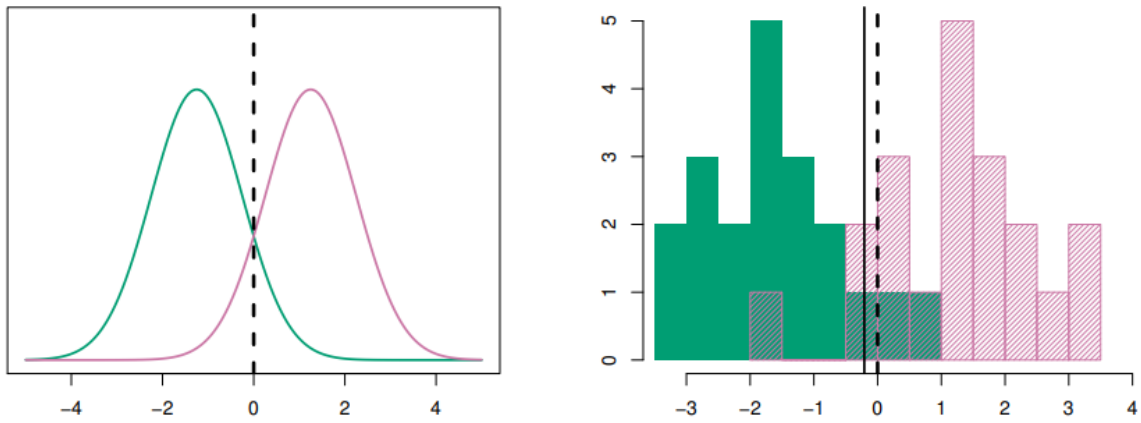
Taking the log and rewriting equation (2.9) results in a so called linear discriminant function

$$\hat{\alpha}_l(x) = x^T\hat{\Sigma}^{-1}\hat{\mu}_l - \frac{1}{2}\mu_l^T\hat{\Sigma}^{-1}\hat{\mu}_l + log\ \theta_l. \tag{2.10}$$

In fact, by equating (2.10) with both classes, we can obtain a linear decision boundary due to the function being linear in x. Figure 2.4 shows an example of such a linear decision boundary derived from 20 training observations (for the 1-dimensional case) which is compared with the true Bayes decision boundary which is determined using the true distribution of the data[5].

[5]One-dimensional Guassian distribution and 20 training observations with dashed line representing Bayes decision boundary and solid line representing LDA decision boundary. Reprinted from *An Introduction to Statistical Learning: with Applications in R* (p. 140), G. James et al., 2013, Springer.

Figure 2.4 One-dimensional Guassian distribution and 20 training observations with dashed line representing Bayes decision boundary and solid line representing LDA decision boundary.



Finally, when the LDA classifier is fitted using a training set, a new set of test observations $\{x_i)\}_{i=n+1}^{n+m}$ can be plugged into the model. The LDA classifier then returns a set of probabilities and allocates observation x_i to class for which probability $\hat{p}_l(x_i)$ is the largest. That is

$$l(x) = \underset{l \in L}{\operatorname{argmax}}\, \hat{p}_l(x_i). \tag{2.11}$$

2.4.3 Quadratic discriminant analysis

quadratic discriminant analysis (QDA) is built upon the same foundation as linear discriminant analysis. More specifically, QDA employs Bayes' theorem, prior probabilities $\theta_l = P(Y = l)$ and likelihood functions $f_l(X = x) = P(X = x|Y = l)$ in the same way as LDA in order to estimate posterior probabilities $p_l(x) = P(Y = l|X = x)$ [18]. However, as opposed to LDA, the coveriance matrix Σ is now assumed to be class specific. Replacing Σ by Σ_l in equation (2.9) gives

$$\hat{p}_l(x) = \frac{\theta_l \frac{1}{2\pi|\hat{\Sigma}_l|^{\frac{1}{2}}} exp\Big(-\frac{1}{2}(x-\hat{\mu}_l)^T \hat{\Sigma}_l^{-1}(x-\hat{\mu}_l)\Big)}{\sum_{j \in L} \theta_l \frac{1}{2\pi|\hat{\Sigma}_j|^{\frac{1}{2}}} exp\Big(-\frac{1}{2}(x-\hat{\mu}_l)^T \hat{\Sigma}_j^{-1}(x-\hat{\mu}_l)\Big)}. \tag{2.12}$$

Taking the log and rewriting equation (2.12) results in the following equivalent quadratic discriminant function

$$\hat{\alpha}_l(x) = -\frac{1}{2}x^T\hat{\Sigma}_l^{-1}x + x^T\hat{\Sigma}_l^{-1}\hat{\mu}_l - \frac{1}{2}x^T\hat{\Sigma}_l^{-1}\hat{\mu}_l - \frac{1}{2}log|\hat{\Sigma}_l| + log\ \theta_l. \tag{2.13}$$

After fitting the QDA classifier using a training set, a new set of test observations $\{x_i\}_{i=n+1}^{n+m}$ can be plugged into the model. The QDA classifier then returns a set of probabilities and allocates observation x_i to class for which probability $\hat{p}_l(x_i)$ is largest. That is

$$l(x) = \underset{l \in L}{\operatorname{argmax}} \hat{p}_l(x_i). \tag{2.14}$$

Due to the fact that QDA estimates seperate coveriance matrices for each class, it tends to be more flexible and therefore has a higher variance compared to LDA.

2.4.4 Random forests

Random forests is a non-parametric tree-based ensemble method [18]. In short, tree-based methods break up the feature space $\mathbb{R}^{\mathrm{K}}$ in set of (distinct) K-dimensional regions $\{R_1, ..., R_j\}$, which are based on splitting-rules for each feature k. Using training data $\{(x_i, y_i)\}_{i=1}^{n}$, a model can be fitted that forms the regions in the feature space, while minimizing a specific error measure $E(\cdot)$. In case of classification, a future observation will eventually be assigned to the class with the highest share of training observations in the region to which the future observation belongs.

For the fitting procedure it is practically infeasible to evaluate every possible partition of the feature space, so a greedy algorithm is applied [16]. Consider observation variable X, feature k and split point s so that we can define the following two half-planes representing the regions of the feature space where X_k is larger or smaller than s:

$$R_1(k, s) = \{X | X_k < s\} \text{ and } R_2(k, s) = \{X | X_k > s\}. \tag{2.21}$$

Each step of the algorithm occurs at a so-called leaf in the tree, which is denoted by v. To each v, a region R_v is connected that can be separated into two new regions $R_{v,1}(k_v, k_v)$ and $R_{v,2}(k_v, s_v)$. At each leaf v, the objective is to split region R_v while minimizing the value of the error measure $E(\cdot)$ over all the possible splitting features k and splitting points s using the training data $\{(x_i, y_i)\}_{i=1}^{n_v}$ (training observations left at leaf v). The optimal combination (k_v^*, s_v^*) can be found by solving the following minimization problem

$$(k_v^*, s_v^*) = \underset{k_v, s_v}{\operatorname{argmin}} E(R_{v,1}(k_v, s_v), R_{v,2}(k_v, s_v)). \tag{2.22}$$

Note that there are multiple options for choosing the error measure in case of classification. One of the most frequently used measures is the Gini index and is characterized by its ability to measure class purity in a region. The Gini index will also be used in this study

and is defined as

$$G = \sum_{l=1}^{L} \hat{p}_{m,l}(1 - \hat{p}_{m,l}). \tag{2.23}$$

Within this definition, $\hat{p}_{m,l}$ represents the proportion of class $l \in L$ training observations $\{(x_i, y_i)\}_{i=1}^{n_{v,m}}$ in region $R_{v,m}(k, s)$ ($m \in \{1, 2\}$) and can be derived by

$$\hat{p}_{m,l} = \frac{1}{n_{v,m}} \sum_{x_i \in R_{v,m}(k,s)} I(y_i = l). \tag{2.24}$$

As the Gini index only needs one region as input, (2.22) can in this case be interpreted as a minimization problem of a weighted Gini index of two regions. These weights depend on the number of training observations $n_{v,m}$ in each of the two regions.

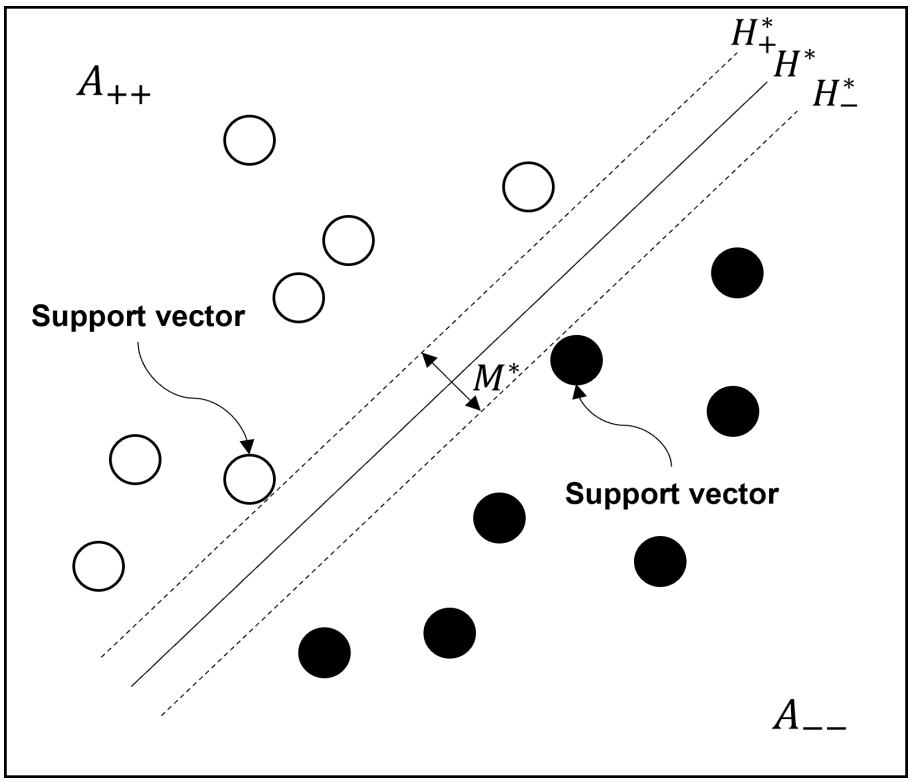
While the above is just an explanation of creating one decision tree, random forests employs an ensemble method called bagging by building multiple decision trees on bootstrapped training observations, after which the set of predictions obtained from these trees are averaged in order to obtain a single low-variance classifier. However, random forests differs from bagging with regard to the way that the features are selected at a splitting step. Instead of considering the full set of features, which is the case for bagging, only a (small) random sample of the full set is chosen as the set of candidates. The rationale behind this sampling step is reducing the correlation between decision trees.

2.4.5 Linear support vector machines

The final method, linear support vector machines (linear SVM), is a non-probabilistic classifier and is based on finding an optimal hyperplane to separate the data. A support vector can be considered as a datapoint that lies closest to a separating hyperplane $H = \{x \mid w^T x + b = 0\}$ that splits up the training data $\{(x_i, y_i)\}_{i=1}^{n}$ ($y \in \{-1, 1\}$) into two areas, A_+ and A_- [39]. Since there exits two areas, one can define two hyperplanes: $H_+ = \{x \mid w^T x + b = 1\}$ in A_+ and $H_- = \{x \mid w^T x + b = -1\}$ and A_-. These hyperplanes are both parallel to the median hyperplane H. Now, let the shortest distance between H_+ and H_- be denoted by margin M. The theoretical goal of linear SVM is then to find an optimal separating hyperplane $H^* = \{x \mid w^{*T} x + b^* = 0\}$ such that margin M is maximized, while ensuring that all positive labeled datapoints in $A_{++} = \{x \mid w^{*T} x + b^* \geq 1\}$ and all negative labeled datapoints are in $A_{--} = \{x \mid w^{*T} x + b^* \leq -1\}$. Hence, the support vectors in both A_{++} and A_{--} are exactly located on the hyperplanes H_+^* and H_-^* and one obtains a perfect maximum margin classifier, which is illustrated in Figure

2.5[6]. Note that the hyperplane functions corresponding to H_+ and H_- are equated to 1 and -1 in order to simplify the problem and to be in line with the values that y can take. However, any combination $\{-\delta, \delta\}$ would have been valid.

Figure 2.5 Linear SVM in an optimal setting on a two-dimensional dataset.



Assume that x_+^H denotes any datapoint on a hyperplane H_+ and x_-^H is the datapoint on hyperplane H_+ closest to x_+^H. Then, it is known that (1) $x_+^H = x_-^H + \delta w$ and (2) $M = ||x_+^H - x_-^H||$. Rewriting (2) using (1) results in $M = \frac{2}{||w||}$. Now, the goal of linear SVM can be captured by the following optimization problem:

$$\max_{w,b} M = \frac{2}{||w||} \quad \text{s.t.} \quad w^T(y_i x_i) + y_i b \geq 1 \quad \forall i \in \{1, ..., n\}. \tag{2.15}$$

Solving this optimization problem is very complex due to the nature of the objective function. This issue can be mitigated by converting it into an equivalent quadratic

[6]Note that A_+ and A_- still apply even though these are not included in Figure 2.5 due to visual limitations.

minimization problem that can be efficiently solved:

$$\begin{aligned} &\min_{w,b} \frac{1}{2}||w||^2 \\ &\text{s.t.} \quad w^T(y_i x_i) + y_i b \geq 1 \quad \forall i \in \{1, ..., n\}. \end{aligned} \tag{2.16}$$

An issue that may arise is that the data is only non-linearly separable. This problem can be mitigated by introducing a trade-off parameter P and a slack variable α_i representing the distance between a misclassified datapoint and the hyperplane H_+ or H_- (depending on the value of y_i). The objective function now contains a trade-off between maximizing the margin and minimizing the sum of all slack variables α_i which can be tuned with P. By allowing misclassifications to occur in the constraints, we get the following minimization problem:

$$\begin{aligned} &\min_{w,b} ||w||^2 + P \sum_{i=1}^{n} \varepsilon_i \\ &\text{s.t.} \quad w^T(y_i x_i) + y_i b \geq 1 - \varepsilon_i \quad \forall i \in \{1, ..., n\} \\ &\varepsilon_i \geq 0 \quad \forall i \in \{1, ..., n\}. \end{aligned} \tag{2.17}$$

Note that by employing the concept of Lagrangian duality, it becomes possible to apply kernel functions which are used to map data into high-dimensional spaces in which it becomes linearly separable. However, since only linear SVM is applied in this study, kernel functions are out of scope.

The linear SVM classifier is fitted using a training set resulting in a decision function $f(x)$. A new set of test observations $\{x_i\}_{i=n+1}^{n+m}$ can be plugged into the model. Then for each observation x_i, $f(x_i)$ returns a decision value. Now, for performance generalization purposes, these decision values are transformed into probabilities using a classifier calibration tool based on the Platt-scaling method introduced by [30]. The procedure of this method is as follows. First a linear SVM classifier is trained on a subset of the training set resulting in a specific decision function $f(x)$. Subsequently, a transformation is applied to $f(x)$ such that a logistic function is obtained which is fitted to the counterpart of the training set with logistic regression. This is repeated for multiple splits of the training set, such that multiple calibrated classifiers are obtained. Finally, probabilistic output is generated by averaging the probability predictions from the separate calibrated classifiers on the test set.

3. Performance Metrics

For the evaluation of the performance of SelfTrain, it is important to choose appropriate performance metrics. More specifically, we want to find a suitable principal performance metric, along with corresponding metrics complementary to the principal metric. In this chapter, an overview of and justification for the selected performance metrics is given. we say that a fraudulent transaction belongs to the positive class and a genuine transaction to the negative class. Furthermore, it is assumed that the testing data contains the same degree of class imbalance as the training data and that the positive class is of significantly higher importance than the negative class.

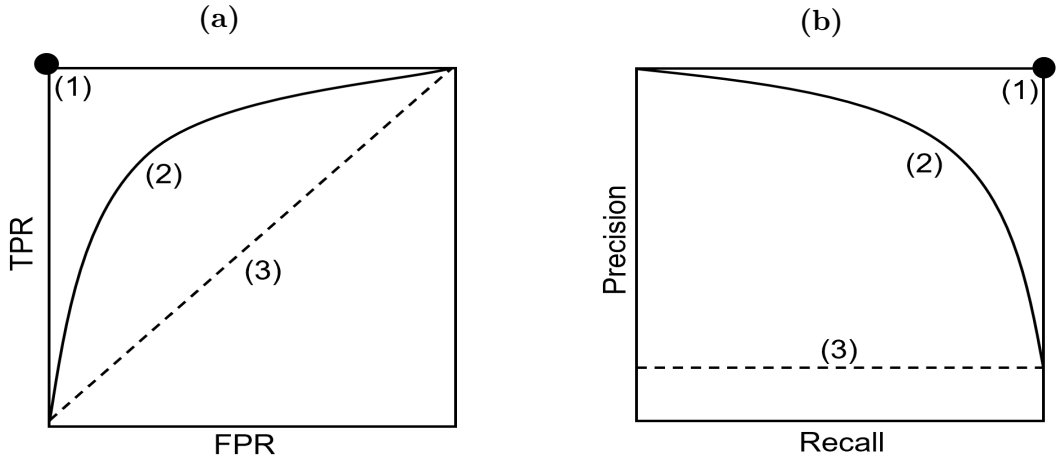
A performance metric that is widely used in detection problems to visualize the performance of a binary classifier is the *receiver-operating characteristics curve* (ROC curve) and it is shown to be insensitive to class imbalance in the test set. This was done by running experiments in which the ROC curves of the same predictor were compared on different test sets with varying degrees of class imbalance [13]. The ROC curve plots the *true positive rate* (TPR) against the *false positive rate* (FPR) under varying discrimination thresholds using linear interpolation. TPR represents the fraction of total positives in a test set which are correctly classified as such, whereas FPR is the fraction of total negatives in a test set which are incorrectly classified as positive. TPR and FPR can be calculated as follows:

$$\text{TPR} = \frac{\text{True Positives}}{\text{Total Positives}}, \tag{3.1}$$

$$\text{FPR} = \frac{\text{False Positives}}{\text{Total Negatives}}. \tag{3.2}$$

The *area under the ROC curve* (ROC AUC) can be calculated to deduce a single value from the ROC curve that represents the performance. It has a range between 0 and 1 (since it is a portion of a square) and a higher area implies a better performing classifier.

Figure 3.1 Illustrations of the ROC curve (a) and the precision recall curve (b).



The ROC AUC can be interpreted as the probability that the corresponding classifier will rank a random positive datapoint higher than a random negative datapoint in terms of prediction probability. In Figure 3.1a, examples of ROC curves are illustrated, including a perfect classifier with a TPR of 1 and a FPR of 0 (1), an intermediate classifier with skill (2) and a random classifier with an AUC of 0.5 (3).

According to [19] it is possible that the ROC curve is overoptimistic and masks a low classifier performance in the case of class imbalance. More specifically, when a negative class dominates a positive class, the FPR may not be able to capture the impact that false positives have on the performance, due to the relatively high number of negative instances in the denominator. To address this issue, the *precision recall curve* (PR curve) can be employed. *Recall* is equal to TPR and denotes the fraction of true positives which are retrieved by the classifier, which is consistent with (3.1). Furthermore, *precision* represents the fraction of total classified positives that are truly positive and it is calculated by

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}. \tag{3.3}$$

Due to the exclusion of true negatives in (3.3), precision is expected to be a more suitable metric for capturing the impact of false positives compared to FPR in case of class imbalance with a dominant negative class. The PR curve plots precision against recall under different discrimination thresholds and like ROC, it is shown by [33] to be robust against class imbalance in the test set. Figure 3.1b shows examples of PR curves, including a perfect classifier with a precision and recall of 1 (1), an intermediate classifier with skill

(2) and a random classifier (3). In contrast with the ROC curve, the PR curve cannot be constructed using linear interpolation, since a linear relation between precision and recall does not necessarily exist [8]. Therefore, linear interpolation could provide an overly optimistic picture of the performance. As an alternative, the construction of an empirical PR curve can be avoided by using *average precision* (AP) to derive a value that summarizes the PR curve [12]. AP is a weighted mean of precision values corresponding to different discrimination thresholds, where each threshold weight reflects the increase in recall compared to the recall from the previous threshold. AP can be defined by

$$AP = \sum_{n} (R_n - R_{n-1}) * P_n, \tag{3.4}$$

where R_n and P_n denote recall and precision for the nth threshold.

Taking the above into account, we have chosen to select AP as the principal performance metric since we argue that it provides the most relevant performance information with respect to the models overall ability of identifying true positive cases while limiting the amount of false positives. Furthermore, the ROC AUC is included as the second performance metric, primarily in order to assess how FPR influences the performance compared to precision and to evaluate how ROC AUC behaves in comparison with AP. Although AP and ROC AUC provide a comprehensive indication of the performance in terms of recall and precision, the values on itself are not very intuitive. However, in order to present operational recall and precision values, we have to decide upon a specific discrimination threshold, which is rather cumbersome since we are not representing a decision-makers perspective in any sense.

Finally, because all the input classifiers are constructed such that they provide probabilistic prediction values, we also want to measure the accuracy of those probabilistic values. For this purpose, the Brier score is included as the third performance metric. Considering a test set $\{(x_i, y_i)\}_{i=n+1}^{n+m}$, it computes the average squared deviation between the predicted probability of the positive class $\hat{p}_{i1}(x_i)$ and the observed class $y_i \in L$ [9].

$$\text{Brier score} = \frac{1}{n} \sum_{i=1}^{m} (\hat{p}_{i1}(x_i) - y_i)^2. \tag{3.5}$$

The Brier score is always a number between 0 and 1, where a lower score indicates a higher accuracy.

Summarizing, the three performance metrics that are selected include AP, ROC AUC and Brier score of which AP is the principal performance metric that is to be maximized

in SelfTrain. The other metrics serve as additional performance indicators in order to put the performance in a broader perspective. In the next chapter, an analysis of the dataset that is used in this study is presented.

4. Results

In the first part of this chapter, a basic analysis of the utilized financial fraud dataset is presented, with a focus on structure, features and separability of the data. Moreover, a clear overview of the data composition for generating results is provided, in which data splitting procedures are elaborated for the sake of appropriately training and testing SelfTrain with the selected dataset. In the second part, the results of applying SelfTrain on the chosen dataset are presented and evaluated. First, the results on the real-data set are set out based on the label scarcity scenarios low, medium and high. Subsequently, the effect of SMOTE is assessed by analyzing the results of SelfTrain applied on SMOTE-data within a medium label scarcity scenario. Finally, a sensitivity analysis is conducted to assess the robustness of SelfTrain with regard to its corresponding hyperparameters.

4.1 Data

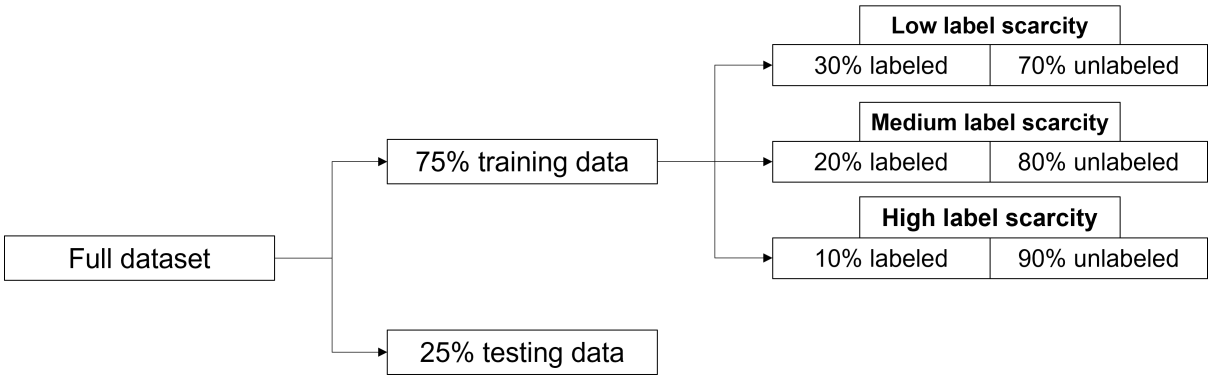
The dataset that is used in this study is derived from Kaggle[1] and consists of anonymized credit card transactions labeled as fraudulent or genuine. It consists of 284,807 transactions made by credit cards in september 2013 by European cardholders. Out of the 284,807 transactions, 492 are labeled as fraudulent, implying an imbalance ratio of (0.17%). Furthermore, the classes fraudelent and genuine are indicated with a 1 and 0, respectively. The raw feature data is a result of dimensionality reduction applied in the form of a principal component analysis (PCA) such that the feature space has been shrunk to 30 numerical features. 28 features are represented as principal components, whereas the 2 features 'Time' and 'Amount' were not included in the PCA and kept their original label and values. The values of the principal components are already normalized, whereas the features 'Time' and 'Amount' still contain their original values as stated before. Although 'Time' is included as a feature, it is still assumed that the data is static. Treating the data as a dynamic time-series would bring up additional factors that have to be taken into account, such as trend, seasonality, cyclicality and random

[1] https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

variability [7]. Moreover, the SelfTrain would have to be radically changed with respect to the sampling strategy of the unlabeled data, since the current strategy would impair the temporal order of the data. For the sake of completeness, the most important descriptive statistics of the data are included in the Appendix A.1 Table A.1. Finally, a 'SVM-trick' is applied to assess the linear separability of the data. This trick involves instantiating a linear SVM classifier with a sufficiently high penalty parameter, making misclassifications sufficiently expensive to guarantee the data to be linearly separated if possible. Then, this classifier is fitted and subsequently tested on the full data set, resulting in an accuracy of 99.86%. Note that this accuracy only gives an indication about the total linear separability, however it does not provide us with information about the distribution of the separated classes following from this separation.

In order to train and validate our models, the data is split up into a training and testing set. Dobbin and Simon [10] argue that in general, the fraction of training data should be within the range of 40% - 80%. Since the labeled fraction of the training set is rather low, we have chosen a training/testing split of 75%/25% to compensate for this. Since all datapoints in the dataset are labeled, there is no initial label scarcity in the training set, nor is there an unlabeled data set available. However, since we want to address the problem of label scarcity with SelfTrain which is built upon the use of unlabeled data, it becomes necessary to artificially mimic label scarcity. This is done by removing labels from a major part of the training set such that this part becomes the unlabeled training set and the counterpart becomes the labeled training set. We distinguish three different scenarios of label scarcity: low label scarcity representing 10% labeled and 90% unlabeled data, medium label scarcity representing 20% labeled and 80% unlabeled data and high label scarcity representing 30% labeled and 70% unlabeled data. For the sake of comparability, we have decided that for each scenario of label scarcity, the imbalance ratio of the corresponding labeled training set must lie within a range of 0.0002 around the original imbalance ratio of 0.0017. See Figure 4.1 for a structured overview of the data composition.

Figure 4.1 Structured overview of the data composition for the generation of results.



To sum up, we are dealing with an imbalanced static dataset which is linearly separable for 99.86% with mostly non-interpretable features. To mimic label scarcity and simultaneously create unlabeled data, three splits of unlabeled and labeled training data are used corresponding to low, medium and high label scarcity scenarios. In the following section, the performance results of applying the SSL framework on our selected dataset are analyzed.

4.2 Empirical analysis

In order to analyze the permance of SelfTrain, each of the selected input classifiers were evaluated by means of three performance metrics: AP, ROC AUC and Brier score, where AP functions as the principal performance metric. SelfTrain is constructed in such a way that it selects the maximum value of the principal performance metric considering all iterations within the algorithm. Hence, for each input classifier, the maximized AP (barring exceptions) corresponds to a different set of pseudo-labeled data that is added to the original training set. This implies that each input classifier is (possibly) fitted on a different training set including its unique set of pseudo-labeled datapoints. The optimal maximized AP scores and the corresponding SelfTrain hyperparameter settings are determined using a manual grid search, which is a parameter tuning method that performs an exhaustive search over a specified set of parameter values to find the best model (with respect to a given performance metric). Our grid search is conducted over the parameter ranges denoted in Table 4.1. Gradual learning parameter w is only applicable for logistic regression, random forests and linear SVM.

Table 4.1 Manual grid search ranges for SelfTrain parameters sample size z, confidence threshold ϵ, and gradual learning option w.

hyperparameter	Range
Sample size z	{0.05, 0.01, 0.005}
Confidence threshold ϵ	{0.6, 0.7, 0.8, 0.9, 0.99}
Gradual learning option w	{True, False}

All input classifiers are implemented with library Scikit-learn 1.0.2. As this study specifically focuses on SelfTrain paramaters, the hyperparameters of the input classifiers are set on default (See Appendix A.2 for an overview of these settings per input classifier). implementing and running SelfTrain is executed on a Windows computer in Python 3.8.8. The CPU times for running SelfTrain with the input classifiers and optimal parameter settings and label scarcity scenarios are shown in Table 4.2. Clearly, random forests and linear SVM require the most computational power for the application of SelfTrain. On the contrary, LDA and QDA clearly require the least amount of computational power. Generally, the CPU time appears to decrease with increasing label scarcity, which can be attributed to a shrinking training set. The ROC AUC and Brier score are determined on the basis of the optimized SelfTrain algorithm and correspond to the maximized AP, meaning that these score metrics do not necessarily represent maximized results. For this reason, AP represents the performance, while ROC AUC and Brier score provide additional performance information. Hence, in the performance analysis, AP is leading, unless another metric is specifically mentioned. SelfTrain was first applied on the real-data for the three different label scarcity scenarios. As a baseline, initial supervised learning results are included, which are in fact the results of solely applying an input classifier on the labeled training set. We denote these supervised learning baselines by SL-base.

Table 4.2 CPU time for running SelfTrain with optimized parameter settings for input classifiers logistic regression, LDA, QDA, random forests and linear SVM considering all three label scarcity scenarios.

	Scenario		
Model	Low	Medium	High
Logistic regression	12m 48.8s	9m 5.6s	7m 2.2s
LDA	4m 1.2s	3m 16.9s	3m 19.7s
QDA	6m 3.7s	4m 39.5s	4m 45.7s
Random Forests	243m 32s	201m 55s	123m 15.7s
Linear SVM	223m 12.3s	164m 9.6s	92m 50.4s

4.2.1 Performance analysis: real-data

In this subsection, the performance results on real-data for all label scarcity scenarios are presented and evaluated. Table 4.3 shows the results of SL-base and SelfTrain on real-data in the low (a), medium (b) and high (c) label scarcity scenario for the input classifiers logistic regression, LDA, QDA, random forests and linear SVM. The imbalance ratios corresponding to the labeled training sets with low, medium and high label scarcity scenarios are 0.1516%, 0.1614% and 0.1735% respectively, meaning that they fall within the imbalance ratio boundaries mentioned in Section 4.1. First, the initial SL-base results are analysed. Thereafter, the mutations in performance by SelfTrain are evaluated.

Based on the initial SL-base performance, it can be stated that random forests is the most favourable input classifier, outperforming the other classifiers in the low and medium label scarcity cases, and being a close second in the high label scarcity case. Moreover, these performances are in all cases accompanied by the lowest Brier score, meaning that random forests has the most reliable predictions. In contrast, the corresponding ROC AUC scores are relatively on the low end. For the high label scarcity scenario, it is linear SVM that outperforms the other input classifiers. Lastly, QDA stands out in a negative manner, due to its low performance accompanied by a high corresponding Brier score. A reason for this low performance could be that QDA is overfitting the data due to the degree of linear seperability of the data (99.86%) in combination with the relatively high flexibility of QDA. Another possible factor could be that the estimated covariance matrices for both classes are equal, whereas QDA assumes the covariance of classes to be non-equal.

Considering performance improvement, it can be seen that for each label scarcity scenario, SelfTrain is able to (at least marginally) improve upon SL-base with input classifiers LDA, random forests and linear SVM, of which SelfTrain with random forests has the most robust improvement. For these input classifiers, the corresponding ROC AUC and Brier scores remain approximately the same, whereas for ROC AUC mainly inconsistent mutations can be observed. Furthermore, SelfTrain with random forests attains bigger improvements in scenarios with higher label scarcity, suggesting that random forest as input classifier has a leveling effect on the scores regarding the different label scarcity scenarios. To a lesser extent, this effect is also visible for LDA and Linear SVM. Focusing on the specific label scarcity scenarios, it can be observed that in low label scarcity scenario, the biggest improvement is achieved by SelfTrain with linear SVM, accompanied by a minor decrease in ROC AUC. The development of AP and class imbalance ratio (CIR) during SelfTrain with linear SVM can be observed in Figure 4.2a.

Table 4.3 Optimized maximum AP scores with corresponding ROC AUC and Brier scores for all input classifiers on real-data for SL-base and SelfTrain.

(a) Low Label Scarcity

SelfTrain			
Model	AP*	ROC AUC	Brier score
Lgistic regression	0.7595	0.9701	0.0008
LDA	0.7349	0.9781	0.0008
QDA	0.1106	0.9634	0.0282
Random forests	0.8243	0.9426	0.0005
Linear SVM	0.8071	0.9665	0.0007
SL-base			
Model	AP*	ROC AUC	Brier score
Logistic regression	0.7595	0.9701	0.0008
LDA	0.7263	0.9825	0.0007
QDA	0.1577	0.9625	0.0213
Random forests	0.8153	0.9418	0.0005
Linear SVM	0.7741	0.9736	0.0007

(b) Medium Label Scarcity

SelfTrain			
Model	AP*	ROC AUC	Brier score
Logistic regression	0.7259	0.9661	0.0008
LDA	0.7369	0.9824	0.0006
QDA	0.1701	0.9530	0.0199
Random forests	0.8099	0.9387	0.0005
Linear SVM	0.7983	0.9700	0.0007
SL-base			
Model	AP*	ROC AUC	Brier score
Logistic regression	0.7259	0.9661	0.0008
LDA	0.7308	0.9823	0.0006
QDA	0.2020	0.9555	0.0158
Random forests	0.7812	0.9295	0.0006
Linear SVM	0.7727	0.9724	0.0006

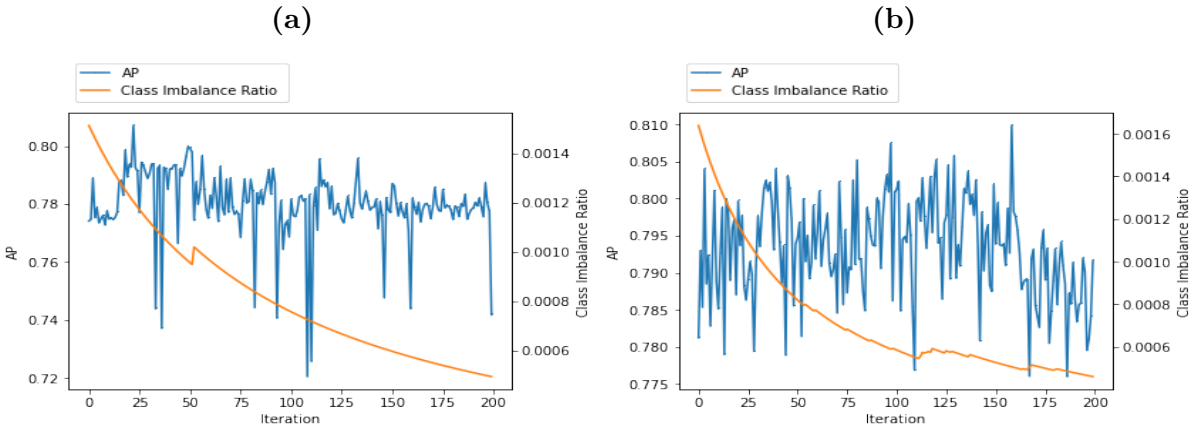
(c) High Label Scarcity

SelfTrain			
Model	AP*	ROC AUC	Brier score
Logistic regression	0.6801	0.9500	0.0009
LDA	0.7313	0.9820	0.0007
QDA	0.4781	0.9050	0.0017
Random forests	0.7708	0.9106	0.0006
Linear SVM	0.7775	0.9562	0.001
SL-base			
Model	AP*	ROC AUC	Brier score
Logistic regression	0.6801	0.9500	0.0009
LDA	0.7174	0.9820	0.0007
QDA	0.4780	0.9005	0.0018
Random forests	0.7471	0.9296	0.0006
Linear SVM	0.7752	0.9562	0.0008

* denotes the principal performance metric

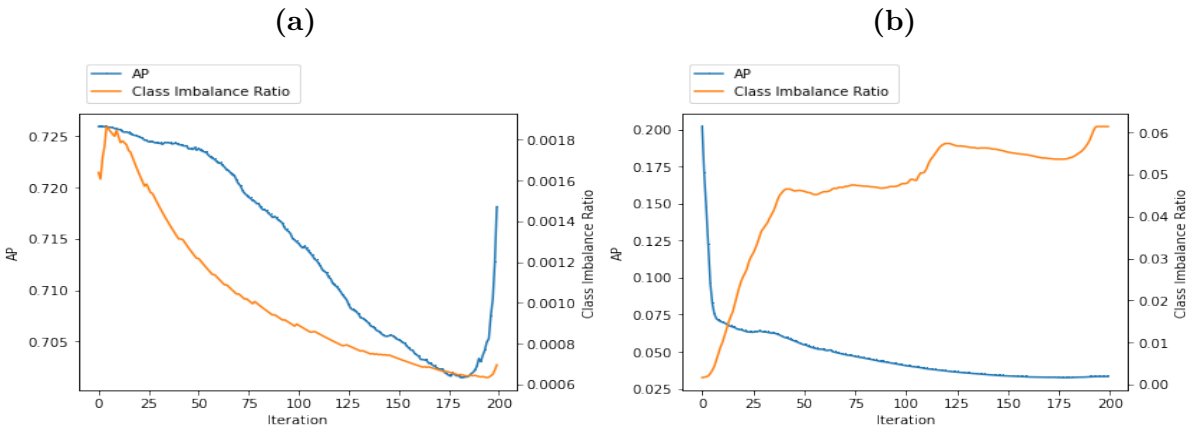
The beginning of the course is increasing, after which the graph shows an oscillating pattern with a downward trend. With regard to class imbalance ratio, an exponentially decreasing pattern is observable with a total decrease of roughly 70% in ratio. In the medium and high label scarcity case, SelfTrain with random forests achieves the biggest improvement. Figure 4.2b shows the AP and CIR pattern for SelfTrain with random forests in the medium label scarcity case. For roughly the first 160 iterations, an oscillatory pattern with an increasing trend is observable. As for SelfTrain with linear SVM, the class imbalance ratio is exponentially decreasing with again approximately 70%. Lastly, an illustration of the development of AP and CIR for SelfTrain with LDA can be found in Appendix A.3 Figure A.3.

Figure 4.2 Development of AP and Class Imbalance Ratio over 200 iterations in the SelfTrain for input classifiers linear SVM (a) and random forests (b).



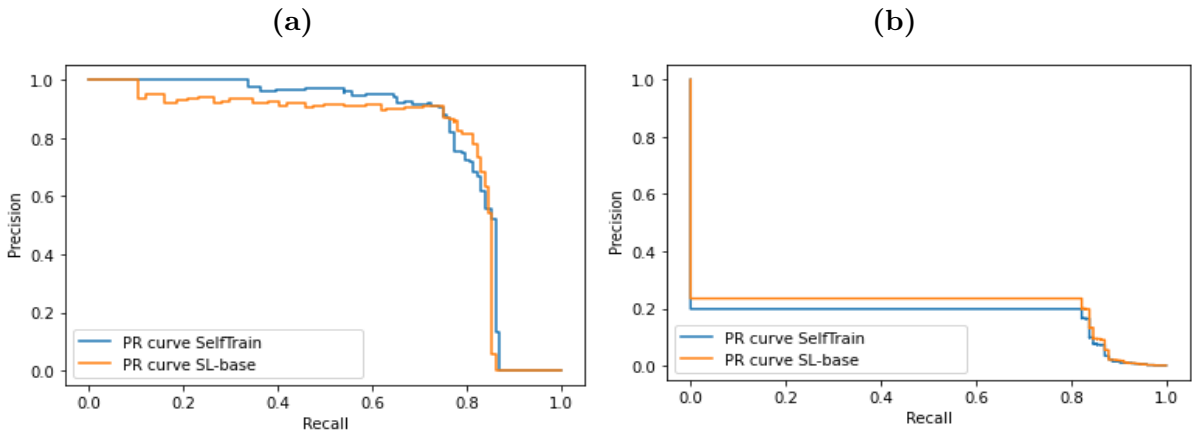
SelfTrain with logistic regression or QDA is never able to (significantly) improve upon SL-base performances. For SelfTrain with QDA, there is even a degraded performance visible in the low and medium label scarcity case. In Figure 4.4a it can be seen that for SelfTrain with logistic regression, AP is strictly decreasing until approximately 175 iterations, after which the it starts to increase until the end. However, the maximal AP remains at the first iteration. The same holds for SelfTrain with QDA in Figure 4.4b, although a small increase in AP can be observed between the 25th and the 50th iteration. Furthermore, note that the class imbalance ratio tremendously increases with QDA and decreases with logistic regression. This implies that within SelfTrain, QDA assigns a relatively high portion of unlabeled data to the positive class, while the reverse is true for logistic regression, even though it is to a much lesser extent.

Figure 4.3 Development of AP and Class Imbalance Ratio over 200 iterations in the SelfTrain for input classifiers logistic regression (a) and QDA (b).



Summarizing, we can state that random forests is the most interesting input classifier for SelfTrain regarding the performance improvement it entails. Moreover, the corresponding probability predictions remain reliable based on the corresponding Brier score mutations. However, it should be noted that the corresponding ROC AUC scores for both SL-base and SelfTrain are consistently in the low end relative to the other input classifiers. QDA clearly is the least interesting classifier in terms of performance improvement. Based on the corresponding Brier scores, it also has the least reliable probability predictions, which is even enforced by SelfTrain. In Figure 4.4, the PR curves for SL-base and SelfTrain are shown for both random forests and QDA in the medium label scarcity scenario. For random forests, the increase in AP by SelfTrain can be mainly attributed to the elevated precision values at recall values between 0.1 and 0.7, approximately (Figure 4.4a). Moreover, in case that a recall above 0.7 is required (e.g. from a business perspective), no significant advantage can be gained from the improvement in AP. For QDA on the other hand, a drop of the (already low) precision values can be observed after SelfTrain at recall values 0 to roughly 0.8 (Figure 4.4b). Outside this range, the precision values for SL-base and SelfTrain do not differ significantly. Note that the PR curves in Figure 4.4 are approximated based on a trapezoidal method as an alternative for linear interpolation between points which is not valid for PR curves which is explained in Chapter 3. This explains the angular structure of the curves.

Figure 4.4 PR curves corresponding to SL-base and SelfTrain for random forests (a) and QDA (b) in the medium label scarcity scenario.



4.2.2 Performance analysis: SMOTE

In this subsection, the effect of re-balancing the data with data augmentation method SMOTE is evaluated. SMOTE is solely applied on the training set, due to the fact that this set would be the only data available in practice. The amount of K nearest neighbours to visit at each minority class datapoint is fixed at 5. Furthermore, for each input classifier, the amount of synthetic minority class datapoints is determined by optimizing the AP score of SelfTrain over the following ratios: 0.2, 0.4, 0.6, 0.8 and 1, where the ratios represent the number of datapoints in the minority class over the number of datapoints in the majority class after resampling. For comparability reasons, we have chosen to apply the optimized SelfTrain parameter settings from the previous section on SelfTrain with SMOTE, which we denote by SelfTrain-S. Table 4.4 shows the supervised baseline results denoted by SL-base-S and the optimized SelfTrain-S for the input classifiers logistic regression, LDA, QDA, random forests and linear SVM in the medium scarcity scenario.

It can be seen that for input classifiers logistic regression, random forests and QDA, SL-base-S and SelfTrain-S outperform SL-base and SelfTrain, where SelfTrain-S with random forests again has the best performance. Moreover, for logistic regression and QDA, SL-base-S even already outperforms SelfTrain. Note that for logistic regression, the corresponding Brier scores are significantly higher after applying SMOTE, implying an impaired accuracy of the probability predictions. Furthermore, for LDA and Linear SVM, a degraded performance from SL-base to SL-base-S and from SelfTrain to SelfTrain-S can be observed. Regarding performance improvement by SelfTrain-S compared to SL-base-S, input classifier LDA achieves an increase of 0.1279 in AP, which is by far the biggest

Table 4.4 Optimized maximum AP scores with corresponding ROC AUC and Brier scores for all input classifiers on SMOTE-data for SL-base-S and SelfTrain-S.

SelfTrain-S			
Model	AP*	ROC AUC	Brier score
Logistic regression	0.7528	0.9506	0.0163
LDA	0.7144	0.9815	0.0008
QDA	0.2459	0.9483	0.0184
Random forests	0.8152	0.9708	0.0006
Linear SVM	0.7512	0.9519	0.0089
SL-base-S			
Model	AP*	ROC AUC	Brier score
Logistic regression	0.7513	0.9511	0.0163
LDA	0.5865	0.9744	0.0015
QDA	0.2457	0.9483	0.0181
Random forests	0.8044	0.9621	0.0006
Linear SVM	0.7401	0.9512	0.0049
* denotes the principal performance metric			

improvement attained by self-training in both the real-data and SMOTE-data context. The other input classifiers are only able to slightly improve SL-base-S with SelfTrain-S. Hence, the performance degradation from Selftrain to Selftrain-S can therefore only be fully attributed to the performance degradation from SL-base to SL-base-S in the LDA case.

4.2.3 Sensitivity analysis

This subsection covers a sensitivity analysis of the SelfTrain parameters z, ϵ and w in order to assess how robust SelfTrain is against changes in the values of these parameters. Naturally, within the analysis, the maximized AP generated by SelfTrain is chosen as the dependent variable, whereas the parameters are considered as the independent variables. As a beginning point, we have selected the optimized SelfTrain parameter values for each input classifier. Then for these parameters, a five-point-interval is formed around these values.

In Figure 4.5 the sensitivity of SelfTrain to sample size z in the medium label scarcity scenario is graphically shown. Around optimized sample size $z* = 0.005$ we constructed the interval $[0.004, 0.0045, 0.005, 0.0055, 0.006]$. For the sake of interpretability, the values in the interval are converted into their corresponding number of iterations by dividing the amount of unlabeled samples by the product of z and the amount of unlabeled samples. For all input classifiers except QDA, AP remains fairly constant as the number

of iterations increases (i.e. z decreases), proving that SelfTrain is considerably robust against sample size z in the specified interval. However, for SelfTrain with QDA, AP appears to be slightly increasing with an increasing number of iterations (i.e. decreasing z). This might indicate that for QDA, the AP can be considerably increased when running SelfTrain with a greater amount of iterations. Note that when we apply the analysis in the low and high label scarcity scenario, we obtain a comparable picture of the robustness. Therefore, these figures are left out.

Figure 4.5 Sensitivity analysis of SelfTrain showing the impact of the number of iterations on AP in the medium label scarcity scenario for all input classifiers.

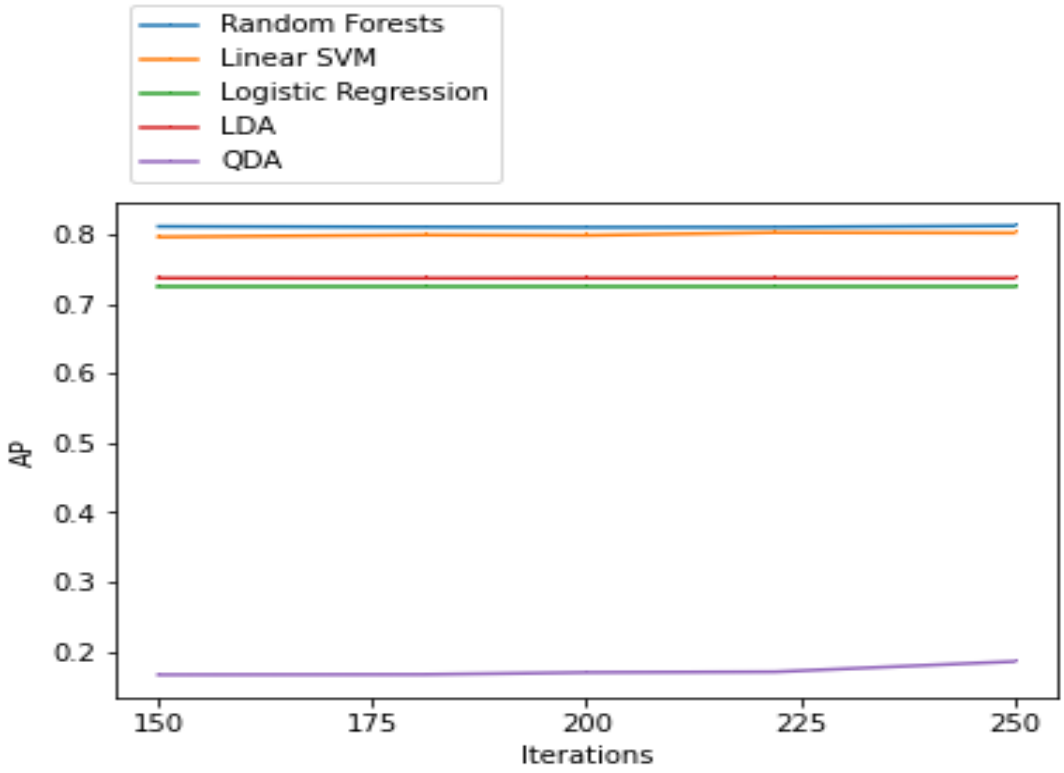
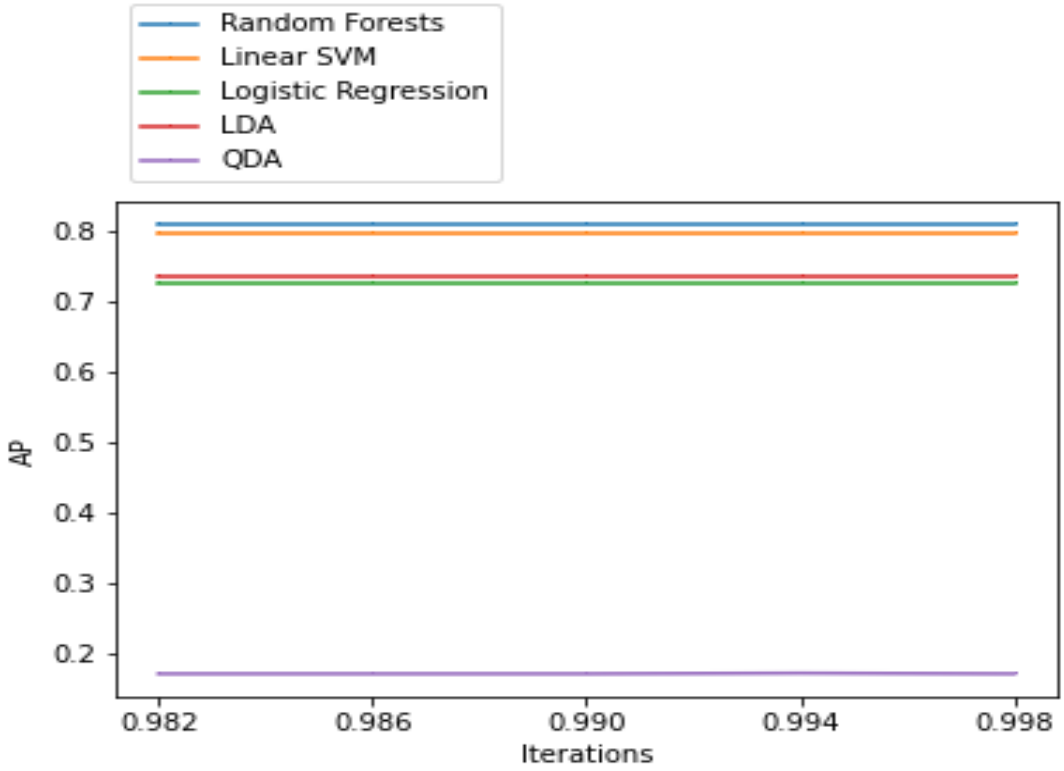


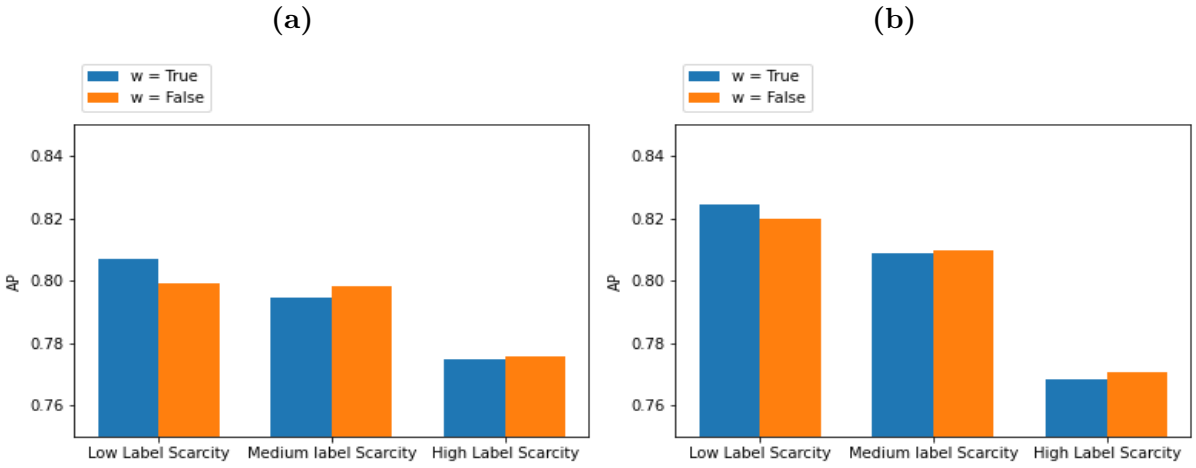
Figure 4.6 graphically shows the sensitivity of SelfTrain to confidence threshold ϵ in the medium label scarcity scenario. Around optimized confidence threshold $\epsilon^* = 0.99$ we constructed the interval $[0.982, 0.986, 0.990, 0.994, 0.998]$. It can be easily seen that for all input classifiers, SelfTrain is completely robust against the varying confidence thresholds in the specified interval. This insensitivity indicates that the probability outputs corresponding to the predictions are generally very close to 1, suggesting that the classifiers are very confident about the predictions for the unlabeled data. The same robustness results are observed for the low and high label scarcity case and are therefore left out.

Figure 4.6 Sensitivity analysis of SelfTrain showing the impact of the confidence threshold on AP in the medium label scarcity scenario for all input classifiers.



In Figure 4.7, the sensitivity of SelfTrain to gradual learning (parameter w) is shown by means of a bar chart. For each of the label scarcity scenario, a comparison is made between the AP generated with gradual learning ($w = True$) and the AP generated without gradual learning ($w = False$). It can be seen that for both linear SVM (Figure 4.7a) and random forests (Figure 4.7b), SelfTrain only benefits from gradual learning in the low label scarcity scenario. In the other scenarios, SelfTrain without gradual learning performs slighlty better. SelfTrain with logistic regression does not show any sensitivity with respect to gradual learning and the bar chart is therefore not included in the figure.

Figure 4.7 Sensitivity analysis of SelfTrain showing the impact of applying gradual learning on AP in each of the label scarcity scenarios for input classifiers linear SVM (a) and random forests (b).



5. Conclusion

In this study, we proposed to investigate the performance of self-training on real-data and SMOTE-data within the context of financial fraud detection by means of integrating self-training algorithm SelfTrain in a structured three stage SSL framework and applying it on input classifiers logistic regression, LDA, QDA, random forests and linear SVM. The justification for this study is based on identified data related problems in financial fraud detection. The roots of these problems lie in the imbalanced structure and limited (label) availability of financial fraud data, which, according to previous studies, can be mitigated with the use of SSL and additional data re-balancing techniques. The main goal of this study was to draw a comparison between the prediction performance of SelfTrain and SL-base for each of the input classifiers. An additional goal was to assess the effect that re-balancing the data with SMOTE has on both the SL-base and SelfTrain results. The performance is measured by the principal performance metric AP and additional information is provided by performance metrics ROC AUC and Brier score. For the evaluation of the results, low, medium and high label scarcity scenarios are considered, representing labeled/unlabeled ratios of 30%/70%, 20%/80% and 10%/90%, where in each scenario, the class imbalance ratio fluctuates around 0.17%.

Findings on real-data show that SelfTrain is always able to improve the prediction performance in terms of AP when applied with LDA, random forests and linear SVM, where the improvement with random forests is most robust and accompanied by the most accurate prediction probabilities in terms of Brier score. In contrast, SelfTrain with logistic regression or QDA never achieves an improved performance, where more detailed inspection shows decreasing AP scores within the algorithm. A possible explanation for these decreasing AP scores is that logistic regression and QDA suffer from confirmation bias, such that inaccurate pseudo-labels are being over-fitted. Especially for QDA this phenomenon is likely to occur, due to its poor SL-base performance. In general, the results on real-data suggest that SelfTrain is an effective method for improving upon SL-base performance for classifiers LDA, random forests and linear SVM in financial fraud detection. Moreover, SelfTrain appears to be the most effective when applied with random Forests, due to the robust performance improvements and sustained accurate prediction probabilities

observed in all label scarcity scenarios. In the medium label scarcity scenario, SMOTE improves the performance of both SL-base and SelfTrain for logistic regression, QDA and random forests. However, the foundation of the improvement on SelfTrain appears to be mainly rooted in the supervised baseline results (SL-base-S), which can be attributed to the direct effect that SMOTE has on the initial data. Moreover, for the mentioned input classifiers SelfTrain is only able to slightly improve upon the SL baselines, indicating that applying SMOTE has a damping effect on the performance improvement of SelfTrain for these classifiers. This suggests that there might exist a ceiling for the performance that can maximally be achieved with SelfTrain.

This study has several limitations regarding study design, data and proposed methodology that should be taken into account while interpreting the study results. First of all, for testing the classification performances of SL-base and SelfTrain only one initial training/testing split is utilized. It is generally perceived that with a large dataset, the training and testing datasets sufficiently approach the true distribution of the data such that significant variations in testing results are not expected to occur. Although it is assumed that our dataset is large enough, it is important to note that we transform labeled training data into unlabeled training data for the sake of creating different label scarcity scenarios. This substantially decreases the amount of initial training data, which imposes the risk of overfitting such that different training/testing splits may give different results. Next, just one set of financial fraud data is considered for evaluating the SL-base and SelfTrain performance. Naturally, drawing conclusions about the effectiveness of SelfTrain in financial fraud detection based on one dataset is premature. Therefore study results must be interpreted as suggestive rather than conclusive. Moreover, due to the fact that the selected dataset ensues from a PCA transformation, (the majority of) features are principal components and are not interpretable. This implies that we are actually dealing with 'black box' classification, since we do not know how raw features interact with the output of generated by any of the input classifiers in both SL-base and SelfTrain. This lack of interpretation may be problematic in real-world applications of our proposed framework. Finally, A drawback of using random rorests and Linear SVM as input classifiers for SelfTrain is that it requires a lot of computing power, which results in high CPU running times. Therefore, the degree of performance improvement should be carefully weighed against the running time of the input classifier in real-world applications.

Taking both the results of this study and its limitations into account, we conclude that in some cases, SelfTrain is suggestive of being an effective method for improving upon supervised baseline performances when applied within the context of financial fraud de-

tection. Even though the same can be stated for SMOTE, this method does not show to be particularly complementary to SelfTrain. From the limitations of this study, it becomes clear that further research is necessary to turn the above mentioned suggestions into conclusions.

The first suggestion for further research builds on the above mentioned limitations and it involves applying the proposed SSL framework with SelfTrain on multiple other financial fraud datasets to assess the robustness of this study's results. In addition, performing multiple training/testing splits and averaging out the corresponding performance scores might give more robust and thus more reliable results. However, note that this takes significantly more CPU time, which can be especially problematic in combination with time-intensive input classifiers like random forests and linear SVM. Next, as this study utilizes data augmentation method SMOTE as a data re-balancing method to address class imbalance, a next step of research could be to compare the use of SMOTE with other data re-balancing methods like oversampling, undersampling or cost-sensitive learning in combination with SelfTrain. Finally, it could be interesting to select other metrics as principal performance metric (e.g. ROC AUC or Brier score) in order to investigate the consistency of classifier rankings with respect to the principal performance metric scores.

BIBLIOGRAPHY

[1] Arazo, E., Ortego, D., Albert, P., O'Connor, N. E., and McGuiness, K. (2019). Pseudo-labeling and confirmation bias in deep semi-supervised learning. *CoRR*, abs/1908.02983.

[2] Berry, M., Mohamed, A., and Yap, B. (2020). *Supervised and Unsupervised Learning for Data Science*. Unsupervised and semi-supervised learning. Springer International Publishing.

[3] Bolstad, W. M. and Curran, J. M. (2016). *Introduction to Bayesian Statistics*. ProQuest Ebook Central.

[4] Casalino, G., Castellano, G., and Mencar, C. (2019). Credit card fraud detection by dynamic incremental semi-supervised fuzzy clustering. In *Proceedings of the 2019 Conference of the International Fuzzy Systems Association and the European Society for Fuzzy Logic and Technology*.

[5] Chawla, N. V., Bowyer, K. W., hall, L. O., and Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357.

[6] Chawla, N. V. and Karakoulas, G. (2005). Learning from labeled and unlabeled data: An empirical study across techniques and domains. *Journal of Artificial Intelligence Research*, 23:331–366.

[7] Davey, A. and Flores, B. (1993). Identification of seasonality in time series: A note. *Mathematical and Computer Modelling*, 18(6):73–81.

[8] Davis, J. and Goadrich, M. (2006). The relationship between precision-recall and roc curves. *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, 148:233–240.

[9] Dharmarathne, G., Hanea, A., and Robinson, A. P. (2021). Improving the computation of brier scores for evaluating expert-elicited judgements. *Frontiers in Applied Mathematics and Statistics*, 7.

[10] Dobbin, K. and Simon, R. (2011). Optimally splitting cases for training and testing high dimensional classifiers. *BMC medical genomics*, 4:31.

[11] Dzakiyullah, N., Pramuntadi, A., and Fauziyyah, A. (2021). Semi-supervised classification on credit card fraud detection using autoencoders. *Journal of Applied Data Sciences*, 2(1).

[12] Fan, G. and Zhu, M. (2011). Detection of rare items with target. *Statistics and Its Interface Volume*, 4:11–17.

[13] Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27:861–874.

[14] Gammerman, A., Azoury, K., and Vapnik, V. (1998). Learning by transduction. pages 148–155.

[15] George, R. and Roy, B. (2022). Handling class imbalance in fraud detection using machine learning techniques. In Kumar, A., Senatore, S., and Gunjan, V. K., editors, *ICDSMLA 2020*, pages 803–813, Singapore. Springer Singapore.

[16] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning (2nd edition)*. Springer-Verlag.

[17] He, H. and Ma, Y. (2013). *Imbalanced Learning: Foundations, Algorithms, and Applications*. Wiley-IEEE Press.

[18] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning: with Applications in R*. Springer.

[19] Jeni, L. A., Cohn, J. F., and Torre, F. D. L. (2013). Facing imbalanced data recommendations for the use of performance metrics. *International Conference on Affective Computing and Intelligent Interaction and workshops*, page 245–251.

[20] Kotsiantis, S., Kanellopoulos, D., and Pintelas, P. (2006). Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering*, 30:25–36.

[21] Kumari, P. and Mishra, S. P. (2019). Analysis of credit card fraud detection using fusion classifiers. In Behera, H. S., Nayak, J., Naik, B., and Abraham, A., editors, *Computational Intelligence in Data Mining*, pages 111–122, Singapore. Springer Singapore.

[22] Lee, D.-H. (2013). Pseudo-label : The simple and efficient semi-supervised learning method for deep neural networks. *ICML 2013 Workshop : Challenges in Representation Learning*.

[23] Ling, C. X. and Sheng, V. S. (2017). Class imbalance problem. In Sammut, C. and Webb, G. I., editors, *Encyclopedia of Machine Learning and Data Mining*, pages 204–205. Springer US, Boston, MA.

[24] Masud, M. M., Woolam, C., and Gao, J. (2012). Facing the reality of data stream classification: coping with scarcity of labeled data. *Knowledge and Information Systems*, 33:213–244.

[25] Melo-Acosta, G. E., Duitama-Muñoz, F., and Arias-Londoño, J. D. (2017). Fraud detection in big data using supervised and semi-supervised learning techniques. In *2017 IEEE Colombian Conference on Communications and Computing*, pages 1–6.

[26] Mojtahed, V. (2019). Big data for fraud detection. In Cecconi, F. and Campennì, M., editors, *Information and Communication Technologies (ICT) in Economic Modeling*, pages 177–192. Springer International Publishing, Cham.

[27] Mqadi, N., Naicker, N., and Adeliyi, T. (2021). A SMOTE based oversampling datapoint approach to solving the credit card data imbalance problem in financial fraud detection. *International Journal of Computing and Digital Systems*, 10:277–286.

[28] Oymak, S. and Gulcu, T. C. (2021). A theoretical characterization of semi-supervised learning with self-training for gaussian mixture models. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 3601–3609. PMLR.

[29] Pavlinek, M. and Podgorelec, V. (2017). Text classification method based on self-training and LDA topic models. *Expert Systems with Applications*, 80:83–93.

[30] Platt, J. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in Large Margin Classifiers*, 10(3):61–74.

[31] Provost, F. (2000). Machine learning from imbalanced data sets 101. In *Proceedings of the AAAI'2000 workshop on imbalanced data sets*, volume 68, pages 1–3. AAAI Press.

[32] Quah, J. T. S. and Sriganesh, M. (2008). Real time credit card fraud detection using computational intelligence. *Expert Systems with Applications*, 35:863–868.

[33] Saito, T. and Rehmsmeier, M. (2015). The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PLoS One*, 10.

[34] Salazar, A., Safont, G., and Vergara, L. (2018). Semi-supervised learning for imbalanced classification of credit card transaction. *2018 International Joint Conference on Neural Networks*, pages 1–7.

[35] Sasaki, Y. (2007). The truth of the F-measure. *Teach Tutor Mater*.

[36] Scudder, H. (1965). Probability of error of some adaptive pattern-recognition machines. *IEEE Transactions on Information Theory*, 11(3):363–371.

[37] Taneja, S., Suri, B., and Kothari, C. (2019). Application of balancing techniques with ensemble approach for credit card fraud detection. In *2019 International Conference on Computing, Power and Communication Technologies*, pages 753–758.

[38] van Engelen, J. E. and Hoos, H. H. (2020). A survey on semi-supervised learning. *Machine Learning*, 109:373–440.

[39] Vapnik, V. N. (2000). *The Nature of Statistical Learning Theory*. Springer.

[40] Weiss, G., McCarthy, K., and Zabar, B. (2007). Cost-sensitive learning vs. sampling: Which is best for handling unbalanced classes with unequal error costs? In *Proceedings of the 2007 International Conference on Data Mining*, pages 35–41. DBLP.

[41] West, J. and Bhattacharya, M. (2016). Intelligent financial fraud detection: A comprehensive review. *Computers Security*, 57:47–66.

[42] Yang, Y. and Xu, Z. (2020). Rethinking the value of labels for improving class-imbalanced learning. *CoRR*, abs/2006.07529.

[43] Zhu, X. and Goldberg, A. B. (2009). *Introduction to Semi-Supervised Learning*. Morgan Claypool.

Appendices

A. Appendix

A.1 Data: descriptive statistics

Table A.1 Basic descriptive statistics of the feature variables.

	mean	std	min	max
Time	9.481386e+04	47488.145955	0.000000	172792.000000
V1	3.918649e-15	1.958696	-56.407510	2.454930
V2	5.682686e-16	1.651309	-72.715728	22.057729
V3	-8.761736e-15	1.516255	-48.325589	9.382558
V4	2.811118e-15	1.415869	-5.683171	16.875344
V5	-1.552103e-15	1.380247	-113.743307	34.801666
V6	2.040130e-15	1.332271	-26.160506	73.301626
V7	-1.698953e-15	1.237094	-43.557242	120.589494
V8	-1.893285e-16	1.194353	-73.216718	20.007208
V9	-3.147640e-15	1.098632	-13.434066	15.594995
V10	1.772925e-15	1.088850	-24.588262	23.745136
V11	9.289524e-16	1.020713	-4.797473	12.018913
V12	-1.803266e-15	0.999201	-18.683715	7.848392
V13	1.674888e-15	0.995274	-5.791881	7.126883
V14	1.475621e-15	0.958596	-19.214325	10.526766
V15	3.501098e-15	0.915316	-4.498945	8.877742
V16	1.392460e-15	0.876253	-14.129855	17.315112
V17	-7.466538e-16	0.849337	-25.162799	9.253526
V18	4.258754e-16	0.838176	-9.498746	5.041069
V19	9.019919e-16	0.814041	-7.213527	5.591971
V20	5.126845e-16	0.770925	-54.497720	39.420904
V21	1.473120e-16	0.734524	-34.830382	27.202839
V22	8.042109e-16	0.725702	-10.933144	10.503090
V23	5.282512e-16	0.624460	-44.807735	22.528412
V24	4.456271e-15	0.605647	-2.836627	4.584549
V25	1.426896e-15	0.521278	-10.295397	7.519589
V26	1.701640e-15	0.482227	-2.604551	3.517346
V27	-3.662252e-16	0.403632	-22.565679	31.612198
V28	-1.217809e-16	0.330083	-15.430084	33.847808
Amount	8.834962e+01	250.120109	0.000000	25691.160000

A.2 Hyperparameter settings

A.2.1 Logistic Regression

Table A.2 Relevant hyperparameter settings for application of logistic regression in Scitkit-learn package.

Parameter	Setting
penalty	'l2'
dual	False
tol	1e-4
C	1.0
fit_intercept	True
intercept_scaling	1
class_weight	None
random_state	None
solver	'lbfgs'
max_iter	100
multi_class	'auto'
verbose	0
warm_start	False
n_jobs	None

A.2.2 Linear Discriminant Analysis

Table A.3 Relevant hyperparameter settings for application of LDA in Scitkit-learn package.

Parameter	Setting
solver	'svd'
shrinkage	None
priors	None
n_components	None
store_covariance	False
tol	1e-4
covariance_estimator	None

A.2.3 Quadratic Discriminant Analysis

Table A.4 Relevant hyperparameter settings for application of QDA in Scitkit-learn package.

Parameter	Setting
priors	None
reg_param	0.0
store_covariance	False
tol	1e-4

A.2.4 Random Forests

Table A.5 Relevant hyperparameter settings for application of random forests in Scitkit-learn package.

Parameter	Setting
n_estimators	100
criterion	'gini'
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	'sqrt'
max_leaf_nodes	None
min_impurity_decrease	0.0
bootstrap	True
oob_score	False
n_jobs	None
random_state	None
verbose	0
warm_start	False
class_weight	None
max_samples	None

A.2.5 Linear SVM

Table A.6 Relevant hyperparameter settings for application of linear SVM in Scitkit-learn package.

Parameter	Setting
penalty	'l2'
loss	'squared_hinge'
dual	True
tol	1e-4
C	1.0
fit_intercept	True
intercept_scaling	1
class_weight	None
verbose	0
random_state	None
max_iter	1000

A.3 Performance analysis

Figure A.1 Development of AP and Class Imbalance Ratio over 200 iterations in the SelfTrain for LDA.