Learned lessons in credit card fraud detection from a practitioner perspective

Andrea DAL POZZOLO[a], Olivier CAELEN[b], Yann-Aël LE BORGNE[a], Serge WATERSCHOOT[b], Gianluca BONTEMPI[a]

[a] *Machine Learning Group, Computer Science Department, Faculty of Sciences ULB, Université Libre de Bruxelles, Brussels, Belgium*
[b] *Fraud Risk Management Analytics, Worldline, Brussels, Belgium*

Abstract

Billions of dollars of loss are caused every year due to fraudulent credit card transactions. The design of efficient fraud detection algorithms is key for reducing these losses, and more and more algorithms rely on advanced machine learning techniques to assist fraud investigators. The design of fraud detection algorithms is however particularly challenging due to non stationary distribution of the data, highly imbalanced classes distributions and continuous streams of transactions.

At the same time public data are scarcely available for confidentiality issues, leaving unanswered many questions about which is the best strategy to deal with them.

In this paper we provide some answers from the practitioner's perspective by focusing on three crucial issues: unbalancedness, non-stationarity and assessment. The analysis is made possible by a real credit card dataset provided by our industrial partner.

Keywords: Incremental Learning, Unbalanced data, Fraud detection

Email addresses: adalpozz@ulb.ac.be (Andrea DAL POZZOLO), olivier.caelen@worldline.com (Olivier CAELEN), yleborgn@ulb.ac.be (Yann-Aël LE BORGNE), serge.waterschoot@worldline.com (Serge WATERSCHOOT), gbonte@ulb.ac.be (Gianluca BONTEMPI)

1. Introduction

Nowadays, enterprises and public institutions have to face a growing presence of fraud initiatives and need automatic systems to implement fraud detection [1]. Automatic systems are essential since it is not always possible or easy for a human analyst to detect fraudulent patterns in transaction datasets, often characterized by a large number of samples, many dimensions and online updates. Also, the cardholder is not reliable in reporting the theft, loss or fraudulent use of a card [2]. Since the number of fraudulent transactions is much smaller than the legitimate ones, the data distribution is unbalanced, i.e. skewed towards non-fraudulent observations. It is well known that many learning algorithms underperform when used for unbalanced dataset [3] and methods (e.g. resampling) have been proposed to improve their performances. Unbalancedness is not the only factor that determines the difficulty of a classification/detection task. Another influential factor is the amount of overlapping of the classes of interest due to limited information that transaction records provide about the nature of the process [4].

Detection problems are typically addressed in two different ways. In the static learning setting, a detection model is periodically relearnt from scratch (e.g. once a year or month). In the online learning setting, the detection model is updated as soon as new data arrives. Though this strategy is the most adequate to deal with issues of non stationarity (e.g. due to the evolution of the spending behavior of the regular card holder or the fraudster), little attention has been devoted in the literature to the unbalanced problem in changing environment.

Another problematic issue in credit card detection is the scarcity of available data due to confidentiality issues that give little chance to the community to share real datasets and assess existing techniques.

2. Contributions

This paper aims at making an experimental comparison of several state of the art algorithms and modeling techniques on one real dataset, focusing in particular on some open questions like: Which machine learning algorithm should be used? Is it enough to learn a model once a month or it is necessary to update the model everyday? How many transactions are sufficient to train the model? Should the data be analyzed in their original unbalanced form?

If not, which is the best way to rebalance them? Which performance measure is the most adequate to asses results?

In this paper we address these questions with the aim of assessing their importance on real data and from a practitioner perspective. These are just some of potential questions that could raise during the design of a detection system. We do not claim to be able to give a definite answer to the problem, but we hope to that our work serves as guideline for other people in the field. Our goal is to show what worked and what did not in a real case study. In this paper we give a formalisation of the learning problem in the context of credit card fraud detection. We present a way to create new features in the datasets that can trace the card holder spending habits. By doing this it is possible to present the transactions to the learning algorithm without providing the card holder identifier. We then argue that traditional classification metrics are not suited for a detection task and present existing alternative measures.

We propose and compare three approaches for online learning in order to identify what is important to retain or to forget in a changing and non-stationary environment. We show the impact of the rebalancing technique on the final performance when the class distribution is skewed. In doing this we merge techniques developed for unbalanced static datasets with online learning strategies. The resulting frameworks are able to deal with unbalanced and evolving data streams. All the results are obtained by experimentation on a dataset of real credit card transactions provided by our industrial partner.

3. State of the art in credit card fraud detection

Credit card fraud detection is one of the most explored domains of fraud detection [5, 6, 7] and relies on the automatic analysis of recorded transactions to detect fraudulent behavior. Every time a credit card is used, transaction data, composed of a number of attributes (e.g. credit card identifier, transaction date, recipient, amount of the transaction), are stored in the databases of the service provider.

However a single transaction information is typically not sufficient to detect a fraud occurrence [5] and the analysis has to consider aggregate measures like total spent per day, transaction number per week or average amount of a transaction [8].

3.1. Supervised versus Unsupervised detection

In the fraud detection literature we encounter both supervised techniques that make use of the class of the transaction (e.g. genuine or fraudulent) and unsupervised techniques. Supervised methods assume that labels of past transactions are available and reliable but are often limited to recognize fraud patterns that have already occurred [9]. On the other hand, unsupervised methods don't use the class of transactions and are capable of detecting new fraudulent behaviours [5]. Clustering based methods [10, 11] form customer profiles to identify new hidden fraud patterns.

The focus of this paper will be on supervised methods. In the literature several supervised methods have been applied to fraud detection such as Neural networks [12], Rule-based methods (BAYES [13], RIPPER [14]) and tree-based algorithms (C4.5 [15] and CART [16]). It is well known however that an open issue is how to manage unbalanced class sizes since the legitimate transactions generally far outnumber the fraudulent ones.

3.2. Unbalanced problem

Learning from unbalanced datasets is a difficult task since most learning systems are not designed to cope with a large difference between the number of cases belonging to each class [17]. In the literature, traditional methods for classification with unbalanced datasets rely on sampling techniques to balance the dataset [3].

In particular we can distinguish between methods that operates at the data and algorithmic levels [18]. At the data level, balancing techniques are used as a pre-processing step to rebalance the dataset or to remove the noise between the two classes, before any algorithm is applied. At the algorithmic level, the classification algorithms themselves are adapted to deal with the minority class detection. In this article we focus on data level techniques as they have the advantage of leaving the algorithms unchanged.

Sampling techniques do not take into consideration any specific information in removing or adding observations from one class, yet they are easy to implement and to understand. *Undersampling* [19] consists in down-sizing the majority class by removing observations at random until the dataset is balanced.

SMOTE [20] over-samples the minority class by generating synthetic minority examples in the neighborhood of observed ones. The idea is to form new minority examples by interpolating between examples of the same class. This has the effect of creating clusters around each minority observation.

Ensemble methods combine balancing techniques with a classifier to explore the majority and minority class distribution. *EasyEnsemble* is claimed in [21] to be better alternative to undersampling. This method learns different aspects of the original majority class in an unsupervised manner. This is done by creating different balanced training sets by *Undersampling*, learning a model for each dataset and then combining all predictions.

3.3. Incremental learning

Static learning is the classical learning setting where the data are processed all at once in a single learning batch. *Incremental learning* instead interprets data as a continuous stream and processes each new instance "on arrival" [22]. In this context it is important to preserve the previously acquired knowledge as well as to update it properly in front of new observations. In incremental learning data arrives in chunks where the underlying data generation function may change, while static learning deals with a single dataset. The problem of learning in the case of unbalanced data has been widely explored in the static learning setting [3, 19, 20, 21]. Learning from non-stationary data stream with skewed class distribution is however a relatively recent domain.

In the incremental setting, when the data distribution changes, it is important to learn from new observations while retaining existing knowledge form past observations. Concepts learnt in the past may re-occur in the future as new concepts may appear in the data stream. This is known as the stability-plasticity dilemma [23]. A classifier is required to be able to respond to changes in the data distribution, while ensuring that it still retains relevant past knowledge. Many of the techniques proposed [24, 25, 26] use ensemble classifiers in order to combine what is learnt from new observations and the knowledge acquired before. As fraud evolves over time, the learning framework has to adapt to the new distribution. The classifier should be able to learn from a new fraud distributions and "forget" outdated knowledge. It becomes critical then to set the rate of forgetting in order to match the rate of change in the distribution [27]. The simplest strategy uses a constant forgetting rate, which boils down to consider a fix window of recent observations to retrain the model. FLORA approach [28] uses a variable forgetting rate where the window is shrunk if a change is detected and expanded otherwise. The evolution of a class concept is called in literature *concept drift*.

Gao et all [29] proposes to store all previous minority class examples into the current training data set to make it less unbalanced and then to combine

the models into an ensemble of classifiers. SERA [30] and REA [31] selectively accumulate old minority class observations to rebalance the training chunk. They propose two different methods (Mahalanobis distance and K nearest neighbours) in order to select the most relevant minority instances to include in the current chunk from the set of old minority instances.

These methods consist in *oversampling* the minority class of the current chunk by retaining old positive observations. Accumulation of previous minority class examples is of limited volume due to skewed class distribution, therefore oversampling does not increase a lot the chunk size.

4. Formalization of the learning problem

In this section, we formalize the credit card fraud detection task as a statistical learning problem. Let X_{ij} be the transaction number j of a card number i. We assume that the transactions are ordered in time such that if X_{iv} occurs before X_{iw} then $v < w$. For each transaction some basic information is available such as amount of the expenditure, the shop where it was performed, the currency, etc. However these variables do not provide any information about the normal card usage. The normal behaviour of a card can be measured by using a set of historical transactions from the same card. For example, we can get an idea of the card holder spending habits by looking at the average amount spent in different merchant categories (e.g. restaurant, online shopping, gas station, etc.) in the last 3 months preceding the transaction. Let $X_{i\lambda}$ be a new transaction and let $dt\,(X_{i\lambda})$ be the corresponding transaction date-time in the dataset. Let T denote the time-frame of a set of historical transactions for the same card. $X_{i\lambda}^H$ is then the set of the historical transactions occurring in the time-frame T before $X_{i\lambda}$ such that $X_{i\lambda}^H = \{X_{ij}\}$, where $dt\,(X_{ij}) \neq dt\,(X_{i\lambda})$ and $dt(X_{i\lambda}) > dt\,(X_{ij}) \geq dt\,(X_{i\lambda}) - T$. For instance, with $T = 90$ days, $X_{i\lambda}^H$ is the set of transactions for the same card occurring in the 3 months preceding $dt\,(X_{i\lambda})$. The card behaviour can be summarised using classical aggregation methods (e.g. *mean*, *max*, *min* or *count*) on the set $X_{i\lambda}^H$. This means that it is possible to create new *aggregated* variables that can be added to the original transaction variables to include information of the card. In this way we have included information about the user behaviour at the transaction level and we can now no longer consider the card ID. Transactions from card holders with similar spending habits will share similar aggregate variables. Let $\{\mathcal{X}\}$ be the new set of transactions with aggregated variables. Each transaction $\mathcal{X}_j$ is assigned a binary

status $\mathcal{Y}_j$ where $\mathcal{Y}_j = 1$ when the transaction j is fraudulent and $\mathcal{Y}_j = 0$ otherwise. The goal of a detection system is to learn $P(\mathcal{Y}|\mathcal{X})$ and predict the class of a new transaction $\widehat{\mathcal{Y}}_N \in (0, 1)$. Note that here the focus is not on classifying the card holder, but the transaction as fraudulent or legitimate.

Credit card fraud detection has some specificities compared to classical machine learning problems. For instance, the continuous availability of new products in the market (like purchase of music on the Internet) changes the behaviour of the cardholders and consequently the distributions $P(\mathcal{X})$. At the same time the evolution of the types of frauds affects the class conditional probability distribution $P(\mathcal{Y}|\mathcal{X})$. As a result the joint distribution $P(\mathcal{X}, \mathcal{Y})$ is not stationary: this is is known as *concept-drift* [32]. Note that Gao [29] suggests that even when the concept drift is not detected, there is still a benefit in updating the models.

5. Performance measure

Fraud detection must deal with the following challenges: i) *timeliness* of decision (a card should be blocked as soon as it is found victim of fraud, quick reaction to the appearance of the first can prevent other frauds), ii) unbalanced class sizes (the number of frauds are relatively small compare to genuine transactions) and iii) cost structure of the problem (the cost of a fraud is not easy to define). The cost of a fraud is often assumed to be equal to the transaction amount [33]. However, frauds of small and big amounts must be treated with equal importance. A fraudulent activity is usually tested with a small amount and then, if successful, replicated with bigger amount. The cost should also include the time taken by the detection system to react. The shorter is the reaction time, the larger is the number of frauds that it is possible to prevent. Depending on the fraud risk assigned by the detection system to the transaction, the following can happens: i) transaction accepted, ii) transaction refused iii) card blocked. Usually the card is blocked only in few cases where there is a high risk of fraud (well known fraudulent patterns with high accuracy, e.g. 99% correct). When a transaction is refused, the investigators make a phone call to the card holder to verify if it is the case of a false alert or a real fraud. The cost of a false alert can then be considered equivalent to the cost of the phone call, which is negligible compared to the loss that occurs in case of a fraud. However, when the number of false alerts is too big or the card is blocked by error, the impossibility to make transactions can translate into big losses for the

customer. For all these reasons, defining a cost measure is a challenging problem in credit card detection.

The fraud problem can be seen as a binary classification and detection problem.

5.1. Classification

In a classification problem an algorithm is assessed on its accuracy to predict the correct classes of new unseen observations. Let $\{\mathcal{Y}_0\}$ be the set of genuine transactions, $\{\mathcal{Y}_1\}$ the set of fraudulent transactions, $\{\widehat{\mathcal{Y}}_0\}$ the set of transactions predicted as genuine and $\{\widehat{\mathcal{Y}}_1\}$ the set of transactions predicted as fraudulent. For a binary classification problem it is conventional to define a confusion matrix (Table 1).

	True Fraud ($\mathcal{Y}_1$)	True Genuine ($\mathcal{Y}_0$)
Predicted Fraud ($\widehat{\mathcal{Y}}_1$)	TP	FP
Predicted Genuine ($\widehat{\mathcal{Y}}_0$)	FN	TN

Table 1: Confusion Matrix

In an unbalanced class problem, it is well-known that quantities like TPR ($\frac{TP}{TP+FN}$), TNR ($\frac{TN}{FP+TN}$) and Accuracy ($\frac{TP+TN}{TP+FN+FP+TN}$) are misleading assessment measures [34]. Balanced Error Rate ($0.5 \times \frac{FP}{TN+FP} + 0.5 \times \frac{FN}{FN+TP}$) may be inappropriate too because of different costs of misclassification false negatives and false positives.

A well accepted measure for unbalanced dataset is AUC (area under the ROC curve) [35]. This metric gives an measure of how much the ROC curve is close to the point of perfect classification. Hand [36] considers calculation of the area under del ROC curve as inappropriate, since this translate into making an average of the misclassification cost of the two classes. An alternative way of estimating AUC is based on the use of the MannWhitney statistic and consists in ranking the observations by the fraud probability and measuring the probability that a random minority class example ranks higher than a random majority class example [37]. By using the rank-based formulation of AUC we can avoid setting different probability thresholds to generate the ROC curve and avoid the problem raised by Hand. Let $n_0 = |\mathcal{Y}_0|$ be the number of genuine transactions, and $n_1 = |\mathcal{Y}_1|$ be the number of fraudulent transactions. Let $g_i = \hat{p}_0(x_{i0})$ be the estimated probability of belonging to

the genuine class for the i^{th} transaction in $\{\mathcal{Y}_0\}$, for $i = 1, ..., n_0$. Define $f_i = \hat{p}_0(x_{i1})$ similarly for the n_1 fraudulent transactions. Then $\{g_1, ..., g_{n_0}\}$ and $\{f_1, ..., f_{n_1}\}$ are samples from the g and f distributions. Rank the combined set of values $g_1, ..., g_{n_0}, f_1, ..., f_{n_1}$ in increasing order and let ρ_i be the rank of the i^{th} genuine transaction. There are $(\rho_i - i)$ fraudulent transactions with estimated probabilities of belonging to class 0 which are smaller than that of the i^{th} genuine transaction [38]. Summing over the 0 class, we see that the total number of pairs of transactions, one from class 0 and one from class 1, in which the fraudulent transaction has smaller estimated probability of belonging to class 0 than does the fraudulent transaction, is

$$\sum_{i=0}^{n_0} (\rho_i - i) = \sum_{i=0}^{n_0} \rho_i - \sum_{i=0}^{n_0} i = \Sigma_0 - n_0(n_0 + 1)/2$$

where $\Sigma_0 = \sum_{i=0}^{n_0} \rho_i$. Since there are $n_0 n_1$ such pairs of transactions altogether, our estimate of the probability that a randomly chosen fraudulent transaction has a lower estimated probability of belonging to class 0 than a randomly chosen genuine transaction is

$$\hat{A} = \frac{\Sigma_0 - n_0(n_0 + 1)/2}{n_0 n_1}$$

$\hat{A}$ gives an estimation of AUC that avoids errors introduced by smoothing procedures in the ROC curve and that is threshold-free [38].

5.2. Detection

The performance of a detection task (like fraud detection) is not necessarily well described in terms of classification [39]. In a detection problem what matters most is whether the algorithm can rank the few useful items (e.g. frauds) ahead of the rest. In a scenario with limited resources, fraud investigators cannot revise all transactions marked as fraudulent from a classification algorithm. They have to put their effort into investigating the transactions with the highest risk of fraud, which means that the detection system is asked to return the transactions ranked by their posteriori fraud probability. The goal then is not only to predict accurately each class, but to return a correct rank of the minority classes.

In this context a good detection algorithm should be able to give a high rank to relevant items (frauds) and low score to non-relevant. Fan et all [39]

consider the average precision (AP) as the correct measure for a detection task. Let π be the number of positive (fraud) case in the original dataset. Out of the $t\%$ top-ranked candidates, suppose $h(t)$ are truly positive ($h(t) <= t$). We can then define recall as $R(t) = h(t)/\pi$ and precision as $P(t) = h(t)/t$. Then $P(t_r)$ and $R(t_r)$ is the precision and recall of the r^{th} ranked observation. The formula for calculating the average precision is:

$$\text{AP} = \sum_{r=1}^{N} P(t_r)\Delta R(t_r)$$

where $\Delta R(t_r) = R(t_r) - R(t_{r-1})$ and N is the total number of observation in the dataset. From the definition of $R(t_r)$ we have:

$$\Delta R(t_r) = \frac{h(t_r) - h(t_{r-1})}{\pi} = \begin{cases} \frac{1}{\pi} & \text{if the the } r^{th} \text{ is fraudulent} \\ 0 & \text{if the the } r^{th} \text{ is genuine} \end{cases}$$

An algorithm "A" is superior to an algorithm "B" only if it detects the frauds before algorithm "B". The better the rank, the greater the AP. The optimal algorithm that ranks all the frauds ahead of the legitimates has average precision of 1.

In detection teams like the one of our industrial partner, each time a fraud alert is generated by the detection system, it has to be checked by investigators before proceeding with actions (e.g. customer contact or card stop). Given the limited number of investigators it is possible to verify only a limited number of alerts. Therefore it is crucial to have the best ranking within the maximum number α of alerts that they can investigate. In this setting it is important to have the highest Precision within the first α alerts.

In the following we will denote as *PrecisionRank* the Precision within the α observations with the highest rank.

6. Strategies for incremental learning with unbalanced fraud data

The most conventional way to deal with sequential fraud data is to adopt a *static approach* (Figure 1) which creates once in a while a classification model and uses it as a predictor during a long horizon. Though this approach reduces the learning effort, its main problem resides in the lack of adaptivity

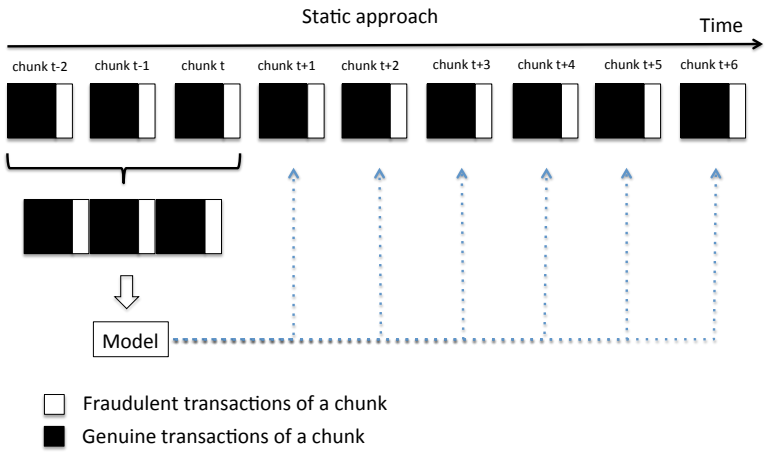

Figure 1: Static approach: a model is trained on $K = 3$ chunks and used to predict future chunks.

which makes it insensitive to any change of distribution in the upcoming chunks.

On the basis of the state-of-the-work described in Section 3.3, it is possible to conceive two alternative strategies to address both the incremental and the unbalanced nature of the fraud detection problem.

The first approach, denoted as the *updating* approach and illustrated in Figure 2, is inspired to Wang et all [40]. It uses a set of M models and a number K of chunks to train each model. Note that for $M > 1$ and $K > 1$ the training sets of the M models are overlapping. This approach adapts to changing environment by forgetting chunks at a constant rate. The last M models are stored and used in a weighted ensemble of models E_M. Let $PrecisionRank_m$ denote the predictive accuracy measured in terms of PrecisionRank on the last (testing) chunk of the m^{th} model. The ensemble E_M is defined as the linear combination of all the M models h_m:

$$E_M = \sum_{m=1}^{M} w_m h_m$$

where

$$w_m = \frac{PrecisionRank_m - PrecisionRank_{min}}{PrecisionRank_{max} - PrecisionRank_{min}}$$

$$PrecisionRank_{min} = \min_{m \in M}(PrecisionRank_m)$$
$$PrecisionRank_{max} = \max_{m \in M}(PrecisionRank_m)$$

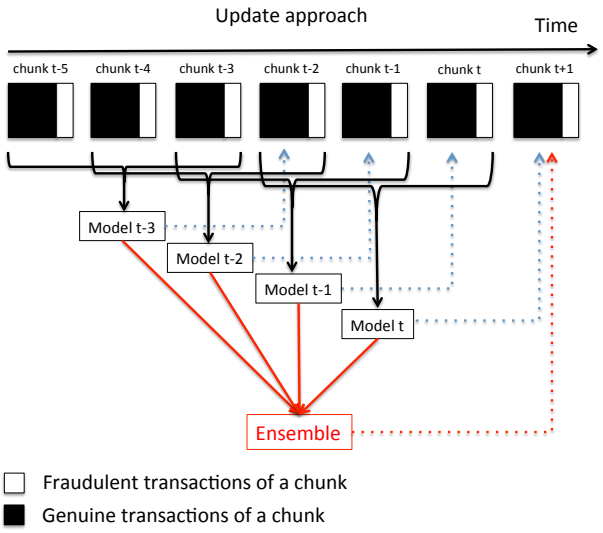

Figure 2: Updating approach for $K = 3$ and $M = 4$. For each new chunk a model is trained on the K latest chunks. Single models are used to predict the following chunk or can be combined into an ensemble.

The second approach denoted as the *forgetting genuine* approach and illustrated in Figure 3 is inspired to Gao et all's work [29]. In order to mitigate the unbalanced effects, each time a new chunk is available, a model is learned on the genuine transactions of the previous K_{gen} chunks and all past fraudulent transactions. Since this approach leads to training sets which grow in size over the time, a maximum training size is set to avoid overloading. Once this size is reached older observations are removed in favor of the more recent ones. An ensemble of models is obtained by combining the last M models as in the update approach.

Note that in all these approaches (including the static one), a balancing technique (Section 3.2) can be used to reduce the skewness of the training set (Figure 4).

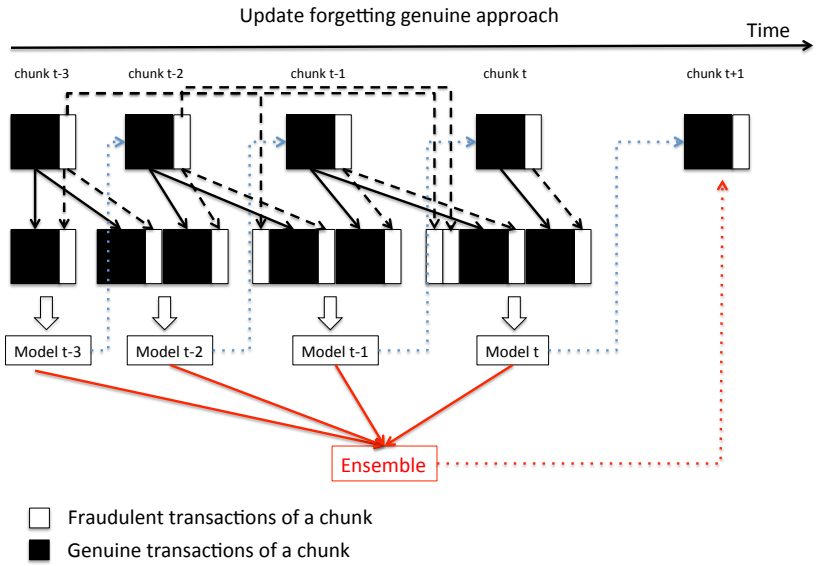

Figure 3: Forgetting genuine approach: for each new chunk a model is created by keeping all previous fraudulent transactions and a small set of genuine transactions from the last 2 chunks ($K_{gen} = 2$). Single models are used to predict the following chunk or can be combined into an ensemble ($M = 4$).

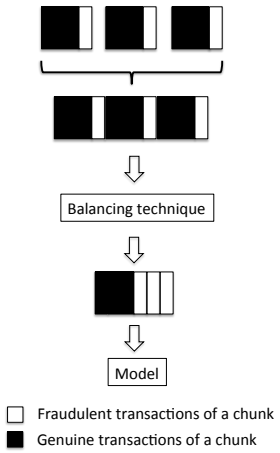

Figure 4: A balancing technique is used to reduce the skewness of the training set before learning a model.

In Table 2 we have summarised the strengths and weaknesses of the incremental approaches presented. The *Static* strategy has the advantage of being fast as the training of the model is done only once, but this does not return a model that follows the changes in the distribution of the data. The other two approaches on the contrary can adapt to concept drift. They differ essentially in the way the minority class is accumulated in the training chunks. The *Forget* strategy propagates instances between chunks leading to bigger training sets and computational burden.

Table 2: Strengths and weaknesses of the incremental approaches

Approach	**Strengths**	**Weaknesses**
Static	– Speed	– No adaptation to changing distributions.
Update	– No instances propagation – Adapts to changing distribution	– Need several chunks for the minority class
Forget	– Accumulates minority instances faster – Adapts to changing distribution	– Instances propagation

7. Experimental assessment

In this section we perform an extensive experimental assessment on the basis of real data (Section 7.1) in order to address common issues that the practitioner has to solve when facing large credit card fraud datasets (Section 7.2).

7.1. Dataset

The credit card fraud dataset was provided by a payment service provider in Belgium. It contains the logs of a subset of transactions from the first of February 2012 to the twentieth of May 2013 (details in Table 3). The dataset was divided in daily chunks and contained e-commerce fraudulent transactions.

The original variables included the transaction amount, point of sale, currency, country of the transaction, merchant type and many others. However

the original variables do not explain card holder behaviour. Aggregated variables are added to the original ones (see section 4) in order to profile the user behaviour. For example the transaction amount and the card ID is used to compute the average expenditure per week and per month of one card, the difference between the current and previous transaction and many others. For each transaction and card we took 3 months ($H = 90$ days) of previous transactions to compute the aggregated variable. Therefore the weekly average expenditure for one card is the weekly average of the last 3 months.

Table 3: Fraudulent dataset

Ndays	Nvar	Ntrx	Period
422	45	2202228	1Feb12 - 20May13

This dataset is strongly unbalanced (the percentage of fraudulent transactions is lower than 0.4%) and contains both categorical and continuous variables. In what follow we will consider that chunks contain sets of daily transactions, where the average transactions per chunk is 5218.

7.2. Learned lessons

Our experimental analysis allows to provide some answers to the most common questions of credit card fraud detection. The questions and the answers based on our experimental findings are detailed below.

7.2.1. Which algorithm and which training size is recommended in case of a static approach ?

The static approach (described in Section 6) is one of the most commonly used by practitioners because of its simplicity and rapidity. However, open questions remain about which learning algorithm should be used and the consequent sensitivity of the accuracy to the training size. We tested three different supervised algorithms: Random Forests (RF), Neural Network (NNET) and Support Vector Machine (SVM) provided by the R software [41]. We used R version 3.0.1 with packages *randomForest* [42], *e1071* [43], *unbalanced* [44] and *MASS* [45].

In order to assess the impact of the training set size (in terms of days/chunks) we carried out the predictions with different windows (K = 30, 60 and 90). All training sets were rebalanced using undersampling at first (50% fraudulent, 50% genuine).

All experiments are replicated five times to reduce the variance caused by the sampling implicit in unbalanced techniques. Figure 5 shows the sum of the ranks from the Friedman test [46] for each strategy in terms of AP, AUC and PrecisionRank. For each chunk, we rank the strategies from the least to the best performing. Then we sum the ranks over all chunks. More formally, let $r_{s,k} \in \{1, ..., S\}$ be the rank of strategy s on chunk k and S be the number of strategies to compare. The strategy with the highest accuracy in k has $r_{s,k} = S$ and the one with the lowest has $r_{s,k} = 1$. Then the sum of ranks for the strategy s is defined as $\sum_{k=1}^{K} r_{k,s}$, where K is the total number of chunks. The higher the sum, the higher is the number of times that one strategy is superior to the others. The white bars denote models which are significantly worse than the best (paired t-test based on the ranks of each chunk).

The strategy names follow a structure built on the following options:

- Algorithm used (RF, SVM, NNET)
- Sampling method (Under, SMOTE, EasyEnsemble)
- Model update frequency (One, Daily, 15days, Weekly)
- Number of models in the ensemble (M)
- Incremental approach (Static, Update, Forget)
- Incremental parameter (K, K_{gen})

Then the strategy options are concatenated using the dot as separation point (e.g. RF.Under.Daily.10M.Update.60K).

In both datasets, Random Forests clearly outperforms its competitors and, as expected, accuracy is improved by increasing the training size (Figure 5). Because of the significative superiority of Random Forests with respect to the other algorithms, in what follows we will limit to consider only this learning algorithm.

7.2.2. Is there an advantage in updating models?

Here we assess the advantage of adopting the *update* approach described in Section 6. Figure 6 reports the results for different values of K and M.

The strategies are called *daily* if a model is built every day, *weekly* if once a week or *15days* if every 15 days. We compare ensemble strategies ($M > 1$)

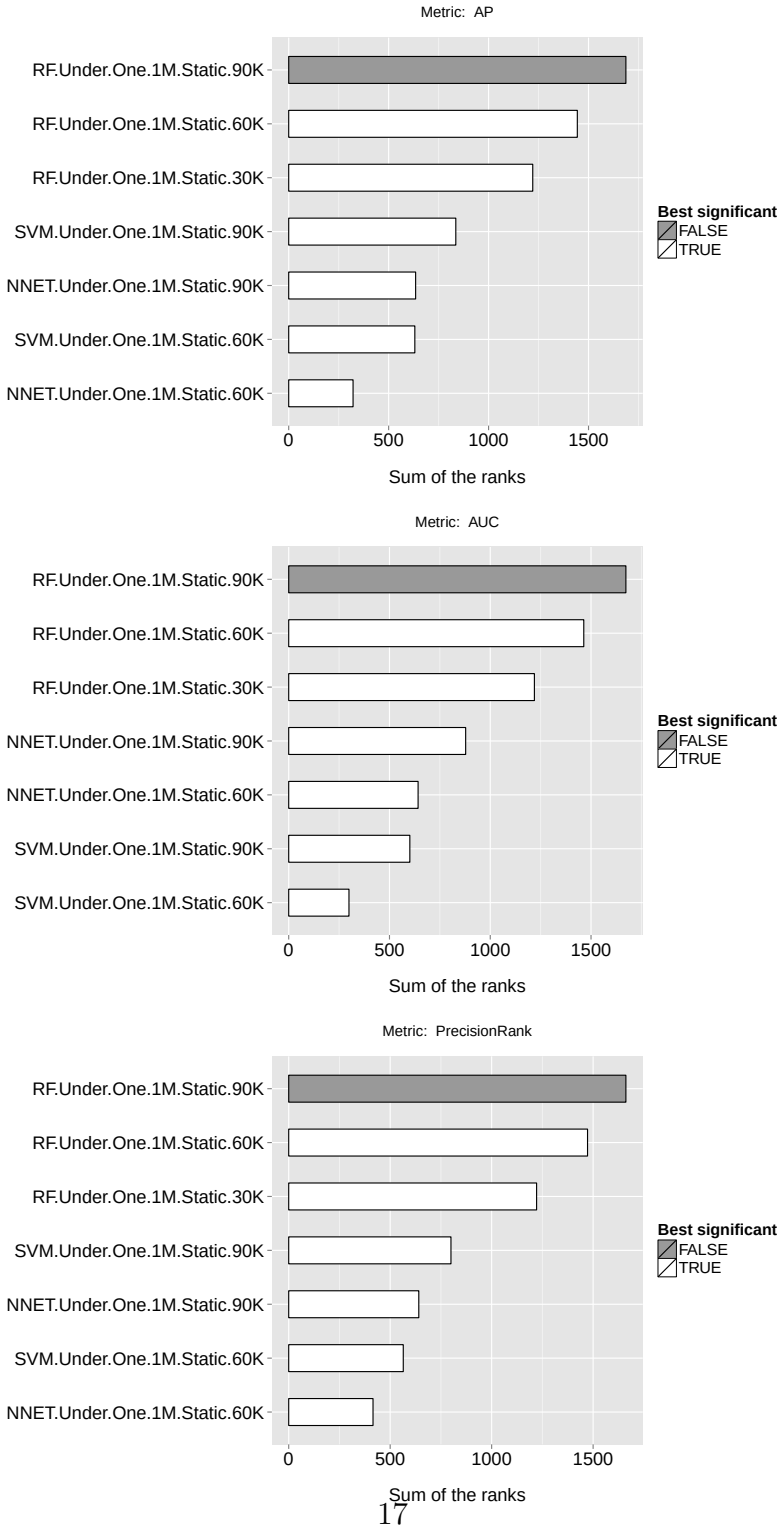

Figure 5: Comparison of static strategies using sum of ranks in all chunks.

with models built on single chunks ($K = 1$) against single models strategies ($M = 1$) using several chunks in the training ($K > 1$).

For all metrics, the best strategy is *RF.Under.Daily.5M.Update.90K*. It creates a new model at each chunk using previous 90 days ($K = 90$) for training and keeps the last 5 models created ($M = 5$) for predictions. In the case of AP however this strategy is not statistically better than the ensemble approaches ranked as second.

For all metrics, the strategies that use only the current chunk to build a model ($K = 1$) are coherently the worst. This confirms the result of previous analysis showing that a too short window of data (and consequently a very small fraction of frauds) is insufficient to learn a reliable model.

When comparing the update frequency of the models using the same number of chunks for training ($K = 90$), *daily* update is ranking always better than *weekly* and *15days*. This confirms the intuition that fraud distribution is always evolving and therefore it is better to update the models as soon as possible.

7.2.3. Retaining old genuine transactions together with old frauds is beneficial?

This section assesses the accuracy of the *forgetting* approach described in Section 6 whose rationale is to avoid the discard of old fraudulent observations.

Accumulating old frauds leads to less unbalanced chunks. In order to avoid having chunks where the accumulated frauds outnumber the genuine transactions, two options are available: i) forgetting some of the old frauds ii) accumulating old genuine transactions as well. In the first case when the accumulated frauds represent 40% of the transaction, new frauds replace old frauds as in Gao [47]. In the second case we accumulate genuine transactions from previous K_{gen} chunks, where K_{gen} defines the number of chunks used (see Figure 3).

Figure 7 shows the sum of ranks for different strategies where the genuine transactions are taken from a different number of days (K_{gen}). The best strategy for AP and PrecisionRank uses an ensemble of 5 models for each chunk ($M = 5$) and 30 days for genuine transactions ($K_{gen} = 30$). The same strategy ranks third in terms of AUC and is significantly worse than the best. To create ensembles we use a time-based array of models of fixed size M, which means that when the number of models available is greater than M, the most recent in time model replaces the M^{th} model in the array

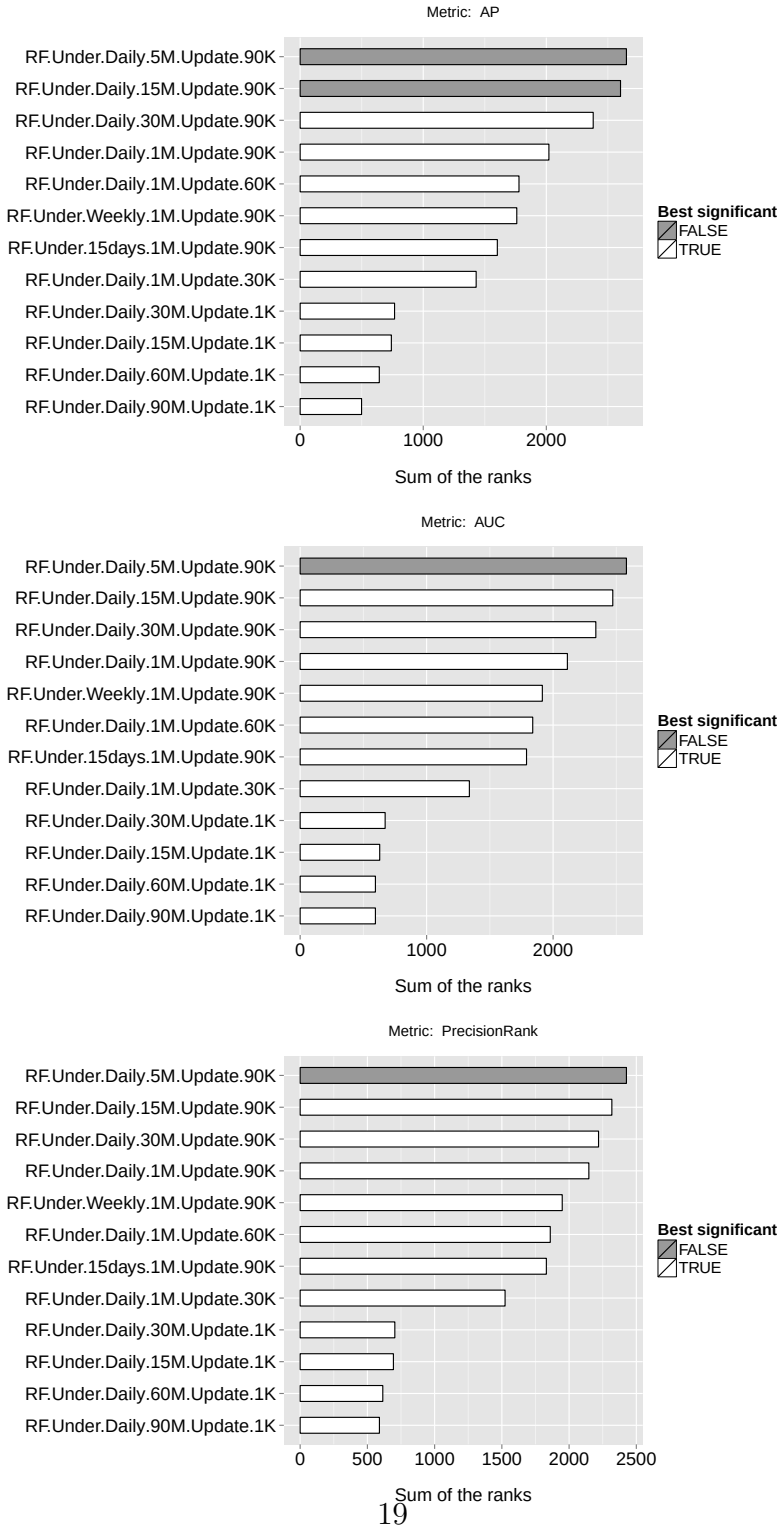

Figure 6: Comparison of update strategies using sum of ranks in all chunks.

removing the oldest model in the ensemble.

In general we see better performances when K_{gen} increase from 0 to 30 and only in few cases $K_{gen} > 30$ leads to significantly better accuracy. Note that in all our strategies after selecting the observations to include in the training sets we use undersampling to make sure we have the two classes equally represented.

7.2.4. Do balancing techniques have an impact on accuracy?

So far we considered exclusively undersampling as balancing technique in our experiments. In this section we assess the impact of using alternative methods like SMOTE and EasyEnsemble. Experimental results (Figure 8) show that they both over-perform undersampling.

In our datasets, the number of frauds is on average 0.4% of all transactions in the chunk. Undersampling randomly selects a number of genuine transactions equal to the number of frauds, which means removing about 99.6% of the genuine transactions in the chunk. EasyEnsemble is able to reduce the variance of undersampling by using several sub-models for each chunk, while SMOTE creates new artificial fraudulent transactions. In our experiments we used 5 sub-models in EasyEnsemble. For all balancing techniques, between the three approaches presented in section 6, the *static* approach is consistently the worse.

In Figure 9 we compare the previous strategies in terms of average prediction time over all chunks. SMOTE is computationally heavy since it consists in oversampling, leading to bigger chunk sizes. EasyEnsemble replicates undersampling and learns from several sub-chunks. This gives higher computational time than undersampling. Between the different incremental approaches, static has the lowest time as the model is learnt once and no retrained. Forget strategy has the highest prediction time over all balancing methods. This is expected since it retains old transactions to deal with unbalanced chunks.

7.2.5. Overall, which is the best strategy?

The large number of possible alternatives (in terms of learning classifier, balancing technique and incremental learning strategy) require a joint assessment of several combination in order to come up with a recommended approach. Figure 10 summaries the best strategies in terms of different metrics. The combinations of EasyEnsemble with *forgetting* emerge as best for all metrics. SMOTE with update is not significantly worse of the best for

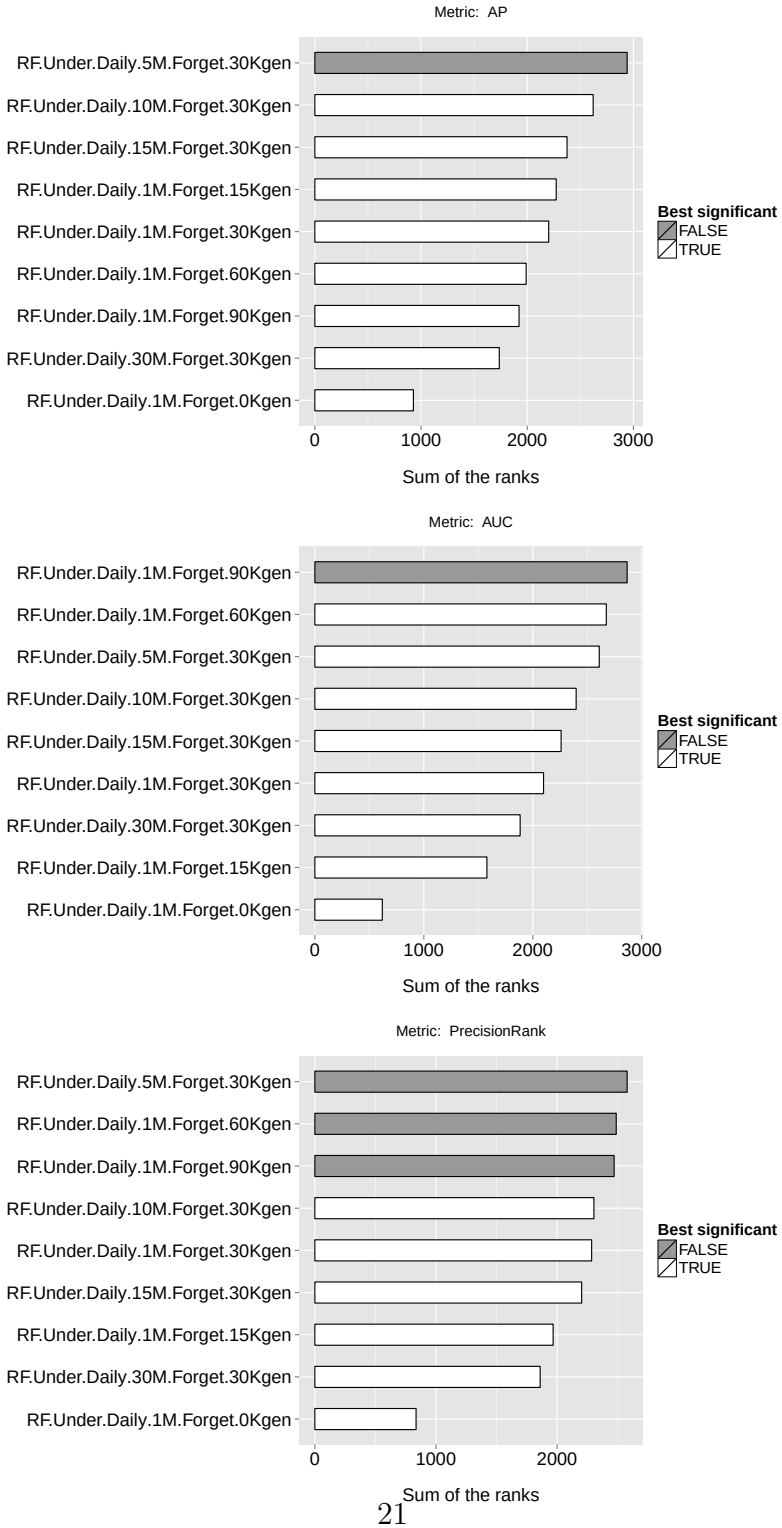

Figure 7: Comparison of forgetting strategies using sum of ranks in all chunks.

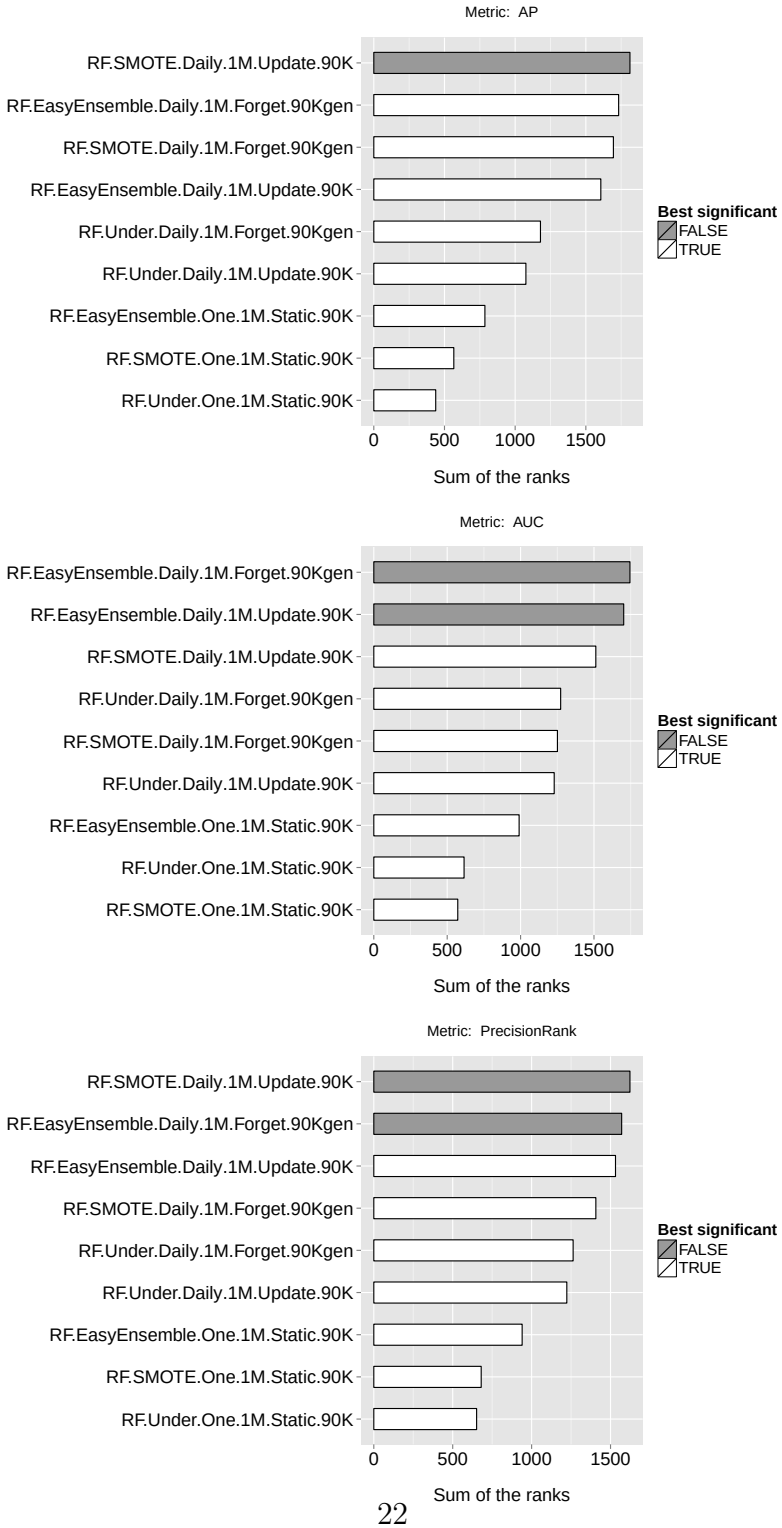

Figure 8: Comparison of different balancing techniques and strategies using sum of ranks in all chunks.

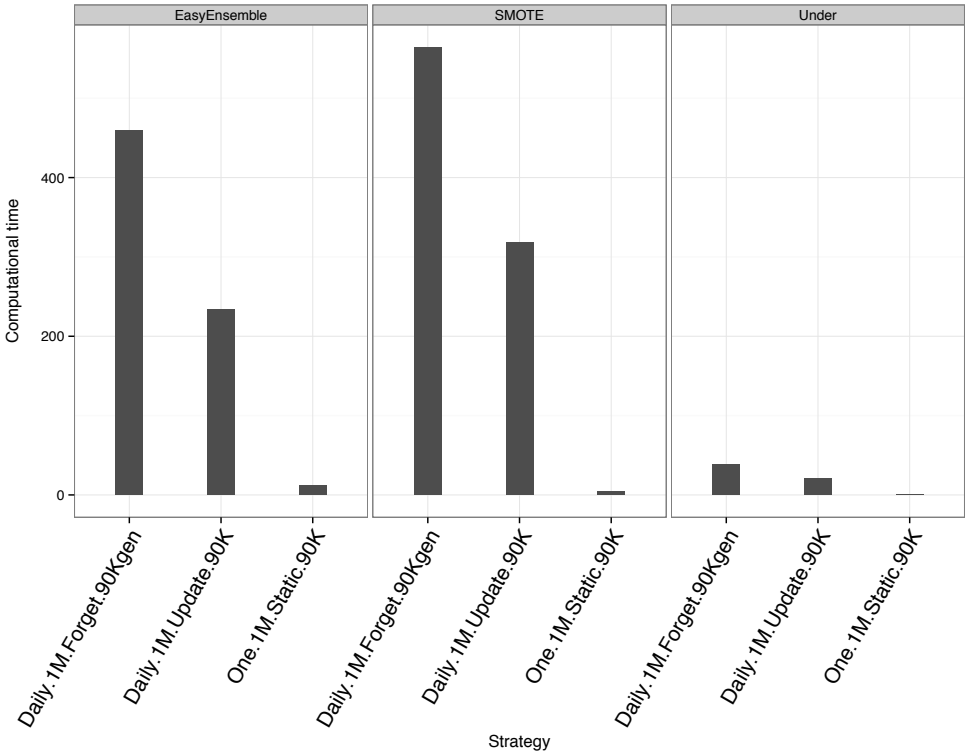

Figure 9: Comparison of different balancing techniques and strategies in terms of average prediction time (in seconds) over all chunks. Experiments run on a HP ProLiant BL465c G5 blades with 2x AMD Opteron 2.4 GHz, 4 cores each and 32 GB DDR3 RAM.

AP and PrecisionRank, but it is not ranking well in terms of AUC. The fact that within the best strategies we see different balancing techniques confirms that in online learning when the data is unbalanced, the adopted balancing strategy may play a major role. As expected the static approach ranks low in Figures 10 as it is not able to adapt to the changing distribution. The *forgetting* approach is significantly better than *update* for EasyEnsemble, while SMOTE gives better ranking with update.

It is worth notice that strategies which combines more than one model ($M > 1$) together with undersampling are not superior to the predictions with a single model and EasyEnsemble. EasyEnsemble learns from different samples of the majority class, which means that for each chunk different concepts of the majority class are learnt.

8. Future work

Future work will focus on the automatic selection of the best unbalanced technique in the case of online learning. Dal Pozzolo et all [48] recently proposed to use a F-race [49] algorithm to automatically select the correct unbalanced strategy for a given dataset. In their work a cross validation is used to feed the data into the race. A natural extension of this work could be the use of racing in incremental data where the data fed into the race comes from new chunks in the stream.

Throughout our paper we used only data driven techniques to deal with the unbalanced problem. HDDT [50] is a decision tree that uses Hellinger distance [51] as splitting criteria that is able to deal with skewed distribution. With HDDT, balancing method are not longer needed before training. The use of such algorithm could remove the need of positive instances propagation between chunks to fight the unbalanced problem.

In our work the combination of models in an ensemble is based on the performance of each model in the testing chunk. Several other methods [52, 53, 54] have been proposed to combine models in presence of concept drift. In future work it would be interesting to test some of these methods and compare it to our framework.

In this manuscript we assumed that there is a single concept to learn for the minority class. However, as frauds are different from each other we could distinguish several sub-concept within the positive class. Hoes et all [52] suggest to use Naive Bayes to retain old positive instances that come from the same sub-concept. REA [31] and SERA [30] proposed by Chen and He

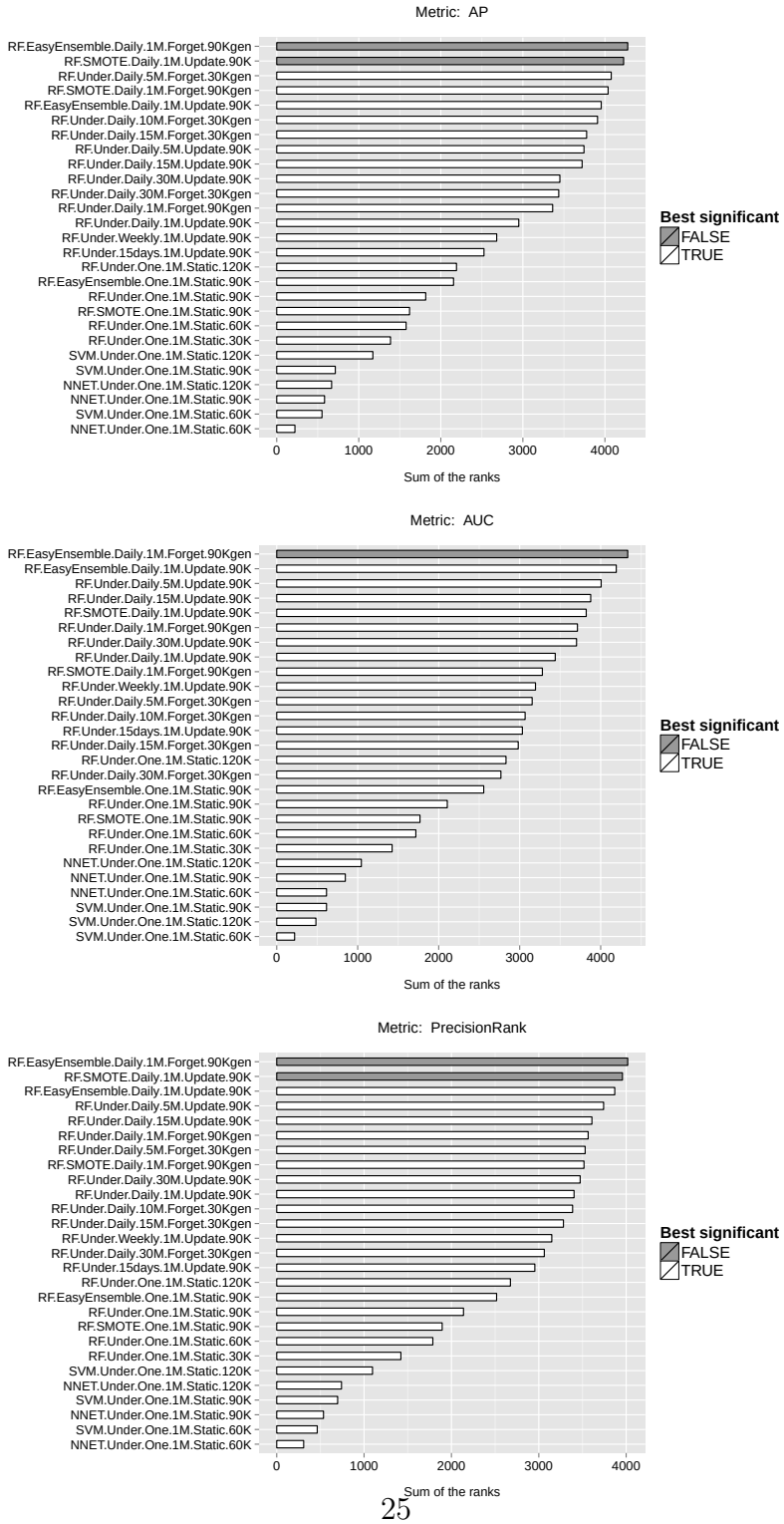

Figure 10: Comparison of all strategies using sum of ranks in all chunks.

propagate to the last chunk only minority class that belong to the same concept using Mahalanobis distance and a k-nearest neighbors algorithm. Future work should take into consideration the possibility of having several minority concepts.

9. Conclusion

The need to detect fraudulent patterns in huge amount of data demands the adoption of automatic methods. The scarcity of public available dataset in credit card transactions gives little chance to the community to test and asses the impact of existing techniques on real data. The goal of our work it to give some guidelines to practitioners on how to tackle the detection problem.

The paper presents the fraud detection problem and proposes AP, AUC and PrecisonRank as correct performance measures for a fraud detection task. Frauds occur rarely compared to the total amount of transactions. As explained in section 5.1, standard classification metrics such as *Accuracy* are not suitable for unbalanced problems. Moreover, the goal of detection is giving the investigators the transactions with the highest fraud risk. For this reason we argue that having a good ranking of the transactions by their fraud probability is more important than having transactions correctly classified.

Credit card fraud detection relies on the analysis of recorded transactions. However a single transaction information is not considered sufficient to detect a fraud occurrence [5] and the analysis has to take into consideration the cardholder behaviour. In this paper we have proposed a way to include cardholder information into the transaction by computing aggregate variables on historical transaction of the same card.

As new credit-card transactions keep arriving, the detection system has to process them as soon as they arrive incrementally and avoid retaining in memory too many old transactions. Fraud types are in continuous evolution and detection has to adapt to fraudsters. Once a fraud is well detected, the fraudster could change his habits and find another way to fraud. Adaptive schemes are therefore required to deal with non-stationary fraud dynamics and discover potentially new fraud mechanisms by itself. We compare three alternative approaches (*static*, *update* and *forgetting*) to learn from unbalanced and non-stationary credit card data streams.

Fraud detection is a highly unbalanced problem where the number of genuine transactions far outnumbers the fraudulent ones. In the static learning

setting a wide range of techniques have been proposed to deal with unbalanced dataset. In incremental learning however few attempts have tried to deal with unbalanced data streams [32, 29, 55]. In these works, the most common balancing technique consists in undersampling the majority class in order to reduce the skewness of the of the chunks.

The best technique for unbalanced data may depend on several factors such as i) data distribution ii) classifier used iii) performance measure adopted, etc. [48]. In our work we adopted two alternatives to undersampling: SMOTE and EasyEnsemble. In particular we show that they are both able to return higher accuracies. Our framework can be easily extended to include other data-level balancing techniques.
The experimental part has shown that in online learning, when the data is skewed towards one class it is important maintaining previous minority examples in order to learn a better separation of two classes. Instance propagation from previous chunks has the effect of increasing the minority class in the current chunk, but it is of limited impact given the small number of frauds. The *update* and *forgetting* approaches presented in section 6 differ essentially in the way the minority class is oversampled in the current chunk. We tested several ensemble and single models strategies using different number of chunks for training. In general we see that models trained on several chunks have better accuracy than single chunk models. Multi-chunks models learn on overlapping training sets, when this happens single models strategies can beat ensembles.

Our framework addresses the problem of non-stationary in data streams by creating a new model every time a new chunk is available. This approach has showed better results than updating the models at a lower frequency (*weekly* or every *15days*). Updating the models is crucial in a non-stationary environments, this intuition is confirmed by the bad results of the static approach. In our dataset, overall we saw Random Forest beating Neural Network and Support Vector Machine. The final best strategy implemented the *forgetting* approach together with EasyEnsemble and daily update.

References

[1] L. Delamaire, H. Abdou, J. Pointon, Credit card fraud and detection techniques: a review, Banks and Bank Systems 4 (2009) 57–68.

[2] J. M. Pavía, E. J. Veres-Ferrer, G. Foix-Escura, Credit card incidents

and control systems, International Journal of Information Management 32 (2012) 501–503.

[3] N. Japkowicz, S. Stephen, The class imbalance problem: A systematic study, Intelligent data analysis 6 (2002) 429–449.

[4] R. C. Holte, L. E. Acker, B. W. Porter, et al., Concept learning and the problem of small disjuncts, in: Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, volume 1, Citeseer, 1989.

[5] R. J. Bolton, D. J. Hand, et al., Unsupervised profiling methods for fraud detection, Credit Scoring and Credit Control VII (2001) 235–255.

[6] R. Brause, T. Langsdorf, M. Hepp, Neural data mining for credit card fraud detection, in: Tools with Artificial Intelligence, 1999. Proceedings., IEEE, 1999, pp. 103–106.

[7] P. Chan, W. Fan, A. Prodromidis, S. Stolfo, Distributed data mining in credit card fraud detection, Intelligent Systems and their Applications 14 (1999) 67–74.

[8] C. Whitrow, D. J. Hand, P. Juszczak, D. Weston, N. M. Adams, Transaction aggregation as a strategy for credit card fraud detection, Data Mining and Knowledge Discovery 18 (2009) 30–55.

[9] R. Bolton, D. Hand, Statistical fraud detection: A review, Statistical Science (2002) 235–249.

[10] J. T. Quah, M. Sriganesh, Real-time credit card fraud detection using computational intelligence, Expert Systems with Applications 35 (2008) 1721–1732.

[11] D. Weston, D. Hand, N. Adams, C. Whitrow, P. Juszczak, Plastic card fraud detection using peer group analysis, Advances in Data Analysis and Classification 2 (2008) 45–62.

[12] J. Dorronsoro, F. Ginel, C. Sgnchez, C. Cruz, Neural fraud detection in credit card operations, Neural Networks 8 (1997) 827–834.

[13] P. Clark, T. Niblett, The cn2 induction algorithm, Machine learning 3 (1989) 261–283.

[14] W. W. Cohen, Fast effective rule induction, in: Machine Learning-International Workshop then Conference, MORGAN KAUFMANN PUBLISHERS, INC., 1995, pp. 115–123.

[15] J. Quinlan, C4. 5: programs for machine learning, volume 1, Morgan kaufmann, 1993.

[16] L. Olshen, C. Stone, Classification and regression trees, Wadsworth International Group (1984).

[17] G. Batista, A. Carvalho, M. Monard, Applying one-sided selection to unbalanced datasets, MICAI 2000: Advances in Artificial Intelligence (2000) 315–325.

[18] N. V. Chawla, N. Japkowicz, A. Kotcz, Editorial: special issue on learning from imbalanced data sets, ACM SIGKDD Explorations Newsletter 6 (2004) 1–6.

[19] C. Drummond, R. Holte, et al., C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling, in: Workshop on Learning from Imbalanced Datasets II, Citeseer, 2003.

[20] N. Chawla, K. Bowyer, L. Hall, W. Kegelmeyer, Smote: synthetic minority over-sampling technique, Arxiv preprint arXiv:1106.1813 (2011).

[21] X. Liu, J. Wu, Z. Zhou, Exploratory undersampling for class-imbalance learning, Systems, Man, and Cybernetics, Part B: Cybernetics 39 (2009) 539–550.

[22] N. C. Oza, Online bagging and boosting, in: Systems, man and cybernetics, volume 3, IEEE, 2005, pp. 2340–2345.

[23] S. Grossberg, Nonlinear neural networks: Principles, mechanisms, and architectures, Neural networks 1 (1988) 17–61.

[24] R. Polikar, L. Upda, S. Upda, V. Honavar, Learn++: An incremental learning algorithm for supervised neural networks, Systems, Man, and Cybernetics, Part C: Applications and Reviews 31 (2001) 497–508.

[25] W. N. Street, Y. Kim, A streaming ensemble algorithm (sea) for large-scale classification, in: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2001, pp. 377–382.

[26] S. Chen, H. He, K. Li, S. Desai, Musera: multiple selectively recursive approach towards imbalanced stream data mining, in: Neural Networks (IJCNN), The 2010 International Joint Conference on, IEEE, 2010, pp. 1–8.

[27] L. I. Kuncheva, Classifier ensembles for changing environments, in: Multiple classifier systems, Springer, 2004, pp. 1–15.

[28] G. Widmer, M. Kubat, Learning in the presence of concept drift and hidden contexts, Machine learning 23 (1996) 69–101.

[29] J. Gao, W. Fan, J. Han, S. Y. Philip, A general framework for mining concept-drifting data streams with skewed distributions., in: SDM, 2007.

[30] S. Chen, H. He, Sera: selectively recursive approach towards nonstationary imbalanced stream data mining, in: Neural Networks, 2009. IJCNN 2009. International Joint Conference on, IEEE, 2009, pp. 522–529.

[31] S. Chen, H. He, Towards incremental learning of nonstationary imbalanced data stream: a multiple selectively recursive approach, Evolving Systems 2 (2011) 35–50.

[32] T. R. Hoens, R. Polikar, N. V. Chawla, Learning from streaming data with concept drift and imbalance: an overview, Progress in Artificial Intelligence 1 (2012) 89–101.

[33] C. Elkan, The foundations of cost-sensitive learning, in: International joint conference on artificial intelligence, volume 17, Citeseer, 2001, pp. 973–978.

[34] F. Provost, Machine learning from imbalanced data sets 101, in: Proceedings of the AAAI2000 Workshop on Imbalanced Data Sets, 2000.

[35] N. V. Chawla, Data mining for imbalanced datasets: An overview, in: Data mining and knowledge discovery handbook, Springer, 2005, pp. 853–867.

[36] D. Hand, Measuring classifier performance: a coherent alternative to the area under the roc curve, Machine learning 77 (2009) 103–123.

[37] D. Bamber, The area above the ordinal dominance graph and the area below the receiver operating characteristic graph, Journal of mathematical psychology 12 (1975) 387–415.

[38] D. J. Hand, R. J. Till, A simple generalisation of the area under the roc curve for multiple class classification problems, Machine Learning 45 (2001) 171–186.

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

[40] H. Wang, W. Fan, P. S. Yu, J. Han, Mining concept-drifting data streams using ensemble classifiers, in: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2003, pp. 226–235.

[41] R Development Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2011. URL: `http://www.R-project.org/`, ISBN 3-900051-07-0.

[42] A. Liaw, M. Wiener, Classification and regression by randomforest, R News 2 (2002) 18–22.

[43] D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, F. Leisch, e1071: Misc Functions of the Department of Statistics (e1071), TU Wien, 2012. URL: `http://CRAN.R-project.org/package=e1071`, r package version 1.6-1.

[44] A. D. Pozzolo, unbalanced: The package implements different data-driven method for unbalanced datasets, 2014. R package version 1.0.

[45] W. N. Venables, B. D. Ripley, Modern Applied Statistics with S, fourth ed., Springer, New York, 2002. URL: `http://www.stats.ox.ac.uk/pub/MASS4`, iSBN 0-387-95457-0.

[46] M. Friedman, The use of ranks to avoid the assumption of normality implicit in the analysis of variance, Journal of the American Statistical Association 32 (1937) 675–701.

[47] J. Gao, B. Ding, W. Fan, J. Han, P. S. Yu, Classifying data streams with skewed class distributions and concept drifts, Internet Computing 12 (2008) 37–49.

[48] A. Dal Pozzolo, O. Caelen, S. Waterschoot, G. Bontempi, Racing for unbalanced methods selection, in: Proceedings of the 14th International Conference on Intelligent Data Engineering and Automated Learning, IDEAL, 2013.

[49] M. Birattari, T. Stützle, L. Paquete, K. Varrentrapp, A racing algorithm for configuring metaheuristics, in: Proceedings of the genetic and evolutionary computation conference, 2002, pp. 11–18.

[50] D. A. Cieslak, N. V. Chawla, Learning decision trees for unbalanced data, in: Machine Learning and Knowledge Discovery in Databases, Springer, 2008, pp. 241–256.

[51] E. Hellinger, Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen., Journal für die reine und angewandte Mathematik 136 (1909) 210–271.

[52] T. R. Hoens, N. V. Chawla, R. Polikar, Heuristic updatable weighted random subspaces for non-stationary environments, in: Data Mining (ICDM), 2011 IEEE 11th International Conference on, IEEE, 2011, pp. 241–250.

[53] R. N. Lichtenwalter, N. V. Chawla, Adaptive methods for classification in arbitrarily imbalanced and drifting data streams, in: New Frontiers in Applied Data Mining, Springer, 2010, pp. 53–75.

[54] J. Z. Kolter, M. A. Maloof, Dynamic weighted majority: A new ensemble method for tracking concept drift, in: Data Mining, 2003. ICDM 2003. Third IEEE International Conference on, IEEE, 2003, pp. 123–130.

[55] G. Ditzler, R. Polikar, N. Chawla, An incremental learning algorithm for non-stationary environments and class imbalance, in: Pattern Recognition (ICPR), 2010 20th International Conference on, IEEE, 2010, pp. 2997–3000.