Pattern Recognition Letters 105 (2018) 175–181

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

End-to-end neural network architecture for fraud scoring in card payments

Jon Ander Gómez[a], Juan Arévalo[b], Roberto Paredes[a], Jordi Nin[b,*]

[a] Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València, València, Spain
[b] BBVA Data & Analytics, Madrid, Spain

ARTICLE INFO

Article history:
Available online 23 August 2017

MSC:
00-01
99-00

Keywords:
Fraud detection
Credit card payments
Deep learning
Neural networks

ABSTRACT

Millions of euros are lost every year due to fraudulent card transactions. The design and implementation of efficient fraud detection methods is mandatory to minimize such losses. In this paper, we present a neural network based system for fraud detection in banking systems. We use a real world dataset, and describe an end-to-end solution from the practitioner's perspective, by focusing on the following crucial aspects: unbalancedness, data processing and cost metric evaluation. Our analysis shows that the proposed solution achieves comparable performance values with state-of-the-art proprietary and costly solutions.

© 2017 Elsevier B.V. All rights reserved.

1. Introduction

The advent of new payment solutions, most of them based on cloud solutions [1], such as person-to-person mobile payments and mobile card-based contactless proximity payments, has significantly increased the use of credit and debit cards. Additionally, the Single Euro Payments Area (SEPA) implemented in 2014 allowed consumers and businesses to use a single payment account for all euro credit card payments [2], hence easing card payment. These and other factors have established credit and debit card as the most popular payment mode for both online as well as daily purchases. Unfortunately, with the arrival of these new technologies, card fraud has also been increased.

The security of card payments and the trust of the general public in making card payments is a matter of concern for any bank in the world. Fraud with cards issued within SEPA stood at 1.3 billion of euros in 2012, *i.e.*, 0.038% of the value of card transactions [3]. Technological advances have made card transactions safer, *e.g.* with the use of a chip instead of a magnetic stripe for authenticating the card. However, fraudsters constantly change their strategies to avoid being detected, making traditional fraud detection tools – such as expert rules or machine learning static models– inadequate [4,5]. In this regard, one of the main challenges is to counteract the increasing fraud for "card-not-present" payments, especially in e-commerce activities.

Different techniques have been proposed to deal with the automatic credit card fraud detection [6–11]. However, several aspects must be considered when building an engine for card fraud detection. For instance, the skewness of the data makes model training challenging; typically, only a very small portion of card transactions are fraudulent (in our dataset, only 1 out of 5000 transactions is fraudulent). On the other hand, the search space has a high dimensionality, so performing a good feature pre-processing step is mandatory to obtain decent classification results. Another critical aspect is that the system has to response in very short times to become useful in real scenarios. In addition, due to the varying nature of fraud strategies, online training or at least frequent re-training is a must in any credit card fraud detection model [12].

However, the most difficult issue to tackle down is that credit card fraud detection is by definition a cost-sensitive problem, in the sense that the cost produced by a false positive is different than the cost of a false negative [13]. Notice that, when the system predicts a transaction as fraudulent when in fact it is not (false positive), the financial institution has an administrative cost, as well as a decrease in customer satisfaction. On the contrary, when the system does not detect a fraudulent transaction (false negative), the amount of that transaction is lost [14]. Moreover, it is not enough to assume a constant cost difference between false positives and false negatives, since the amount of the transactions varies significantly; therefore, its financial impact is not constant because it depends on the amount of each transaction.

* Corresponding author.
E-mail addresses: jon@dsic.upv.es (J.A. Gómez), juanmaria.arevalo@bbvadata.com (J. Arévalo), rparedes@dsic.upv.es (R. Paredes), jordi.nin@bbvadata.com (J. Nin).

http://dx.doi.org/10.1016/j.patrec.2017.08.024
0167-8655/© 2017 Elsevier B.V. All rights reserved.

In this paper we propose a new approach for online credit card fraud detection based on a set of artificial neural networks (ANN) [15]. Firstly, we apply a cascade of two consecutive filters to each transaction. The goal of this cascade is not to definitively classify a transaction as fraudulent or genuine, but to reduce the data unbalance as much as possible, keeping the largest amount of fraudulent transactions whilst removing as many genuine transactions as possible. Each filter is implemented as an ensemble of ANNs. Each single ANN is trained as a binary classifier *(genuine vs fraudulent transactions)*, whose output layer has two neurons with the softmax activation function. Using this combination of ensembles and cascading methods, the ratio of genuine to fraudulent transactions is typically reduced from 5000 : 1 to around 100: 1. Once the data skewness has been mitigated as much as possible, another ANN is trained to finally score and classify each transaction.

As aforementioned, in order to construct a credit card fraud detection model, it is very important to use those features that allow an accurate classification. Traditional systems only exploit raw transactional features –such as time, amount or place of the transaction– and do not consider the spending behavior of the customer or the merchant, which is expected to help the model to discover complex fraud patterns [16]. In ref. [17] a transaction aggregation strategy is applied to take into account the customer spending behavior. The computation of the aggregated features consists of grouping the transactions made during the last given number of hours. The aggregation is made first by card or account number; then by transaction type, merchant group, country or other; finally, the number of transactions or the total amount spent on those transactions is calculated. Instead of these approaches, in this paper we propose to mix raw transactional features with aggregated spending behavior at customer, merchant, currency and country level.

Our dataset consists of more than 900 millions of anonymized real card transactions from January 2014 to June 2015 provided by BBVA, together with another anonymized data set corresponding to all fraud claims of BBVA's customers. In order to label transactions as fraudulent or genuine, we have created an approximate record linkage system to relate fraud claims with real transactions. With such datasets, we have been able to reproduce all the issues of a real environment to test our approach. Furthermore, we have trained our system and tested it obtaining similar results to the ones provided by actual commercial solutions.

The rest of this paper is organized as follows. Firstly, in Section 2, we describe the credit card fraud detection scenario. Then, in Section 3 we introduce some required concepts about artificial neural networks. Later, Section 4 gives all the details of our proposal. Experiments are detailed in Section 5. Finally, the paper ends with some conclusions and future works.

2. Credit card fraud detection scenario

Although it is important to understand that there are multiple payment systems, all of them function in a similar fashion. Here, we introduce a generic overview of the card transaction process without describing the particularities of each card system.

From a general point of view, every credit card transaction is the result of a series of interactions among several participants, see Fig. 1 for a representation of the full transaction cycle. Such cycle is typically characterized by the following steps:

1. The cardholder presents a card to pay for purchases, or type the card information in online payments.
2. The merchant processes the card and transaction information, and request an authorization from the bank.
3. The bank submits the authorization request to the card issuer.

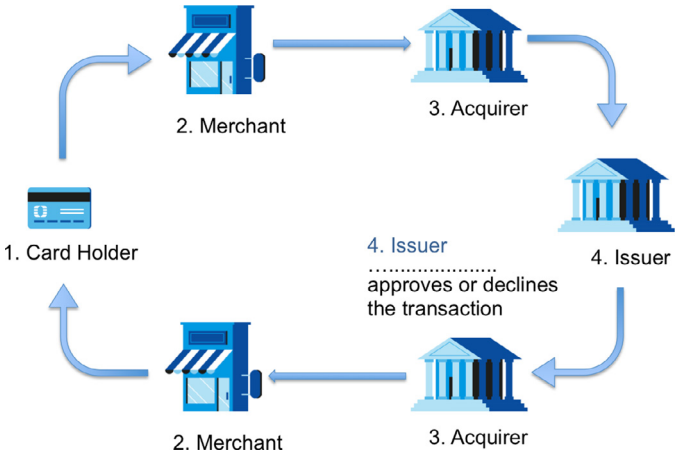

Fig. 1. Card transaction validation flow.

4. The card issuer approves or declines the transaction.
5. The bank forwards the response to the merchant.
6. The merchant receives the authorization response and completes the transaction accordingly.

Authorization (step 4) involves assessing card's transaction risk and, if approved, reserving the sales amount on the cardholder's account. Here is where fraud scoring algorithms are executed and, depending on the obtained score, the authorization response can be approved, declined, called for referrals, picked up or not match. Then, the merchant must proceed accordingly.

Once the authorization process ends, card transactions are still in process as they must be settled. Depending on the settling process this may require some time, from minutes to days. For this reason, when an approved payment is a fraudulent operation, cardholders cannot see the movement in the bank account immediately. Moreover, it may happen that a cardholder does not check card balance quite often; therefore, the reporting of fraudulent payments to the card issuer may have an important delay, or even present involuntary mistakes. Besides, cardholders may report non-fraudulent payments among fraudulent ones for their own benefit.

In order to mitigate this problem, reported fraudulent payments are only linked with payments in the card transactions database when a one to one relation between two operations is found. To do this, we have applied an approximate record linkage process with the import and timestamp fields. Specifically, we match two operations when imports differ less than a 10% and timestamps differ in less than ± 36 hours. Using this process, we link around 90% of reported fraudulent payments. Note that, this linkage is completely unsupervised and it may contain incorrect links (*i.e.* wrong labeled operations).

3. Artificial Neural Networks

In the last years, artificial neural networks, under the new branding of Deep Learning (DL), have improved the state of the art in machine learning domains in which the data is related to high level concepts, hence requiring many levels of abstraction. Deep architectures are able to be self-trained in order to catch these different abstraction levels in different layers [18]. Due to this, DL has been successfully applied in different domains such as speech recognition or image processing.

DL approaches consist of avoiding most of the preprocessing steps on the input data by training (usually by backpropagation) a many layers Neural Network. Deep Neural Networks (DNN) are able to learn different levels of abstraction in each of its layers, even when a minimum preprocessing is applied on the input data.

Specifically Neural Networks cascades (NNC) have been successfully trained for face detection in images [19], among others. NNCs take advantage of their architecture requiring less parameters in each layer than other types of networks [20] making its training easier.

In the rest of this section we introduce the main ANN details used in our architecture.

3.1. Artificial Neural Networks for classification

An artificial neural network is a set of units –called neurons– interconnected. The units are essentially simple mathematical models defining a non-linear projection Φ: $R^D \rightarrow R$, where D is the dimension of the input space. It is customary to represent this transformation in matrix notation as

$$\Phi(\mathbf{x}) = s(b + \mathbf{w}^T \cdot \mathbf{x}). \tag{1}$$

Here, $\mathbf{x} \in R^D$ is the input vector, while the bias $b \in R$ and the weight vector $\mathbf{w} \in R^D$ are learnable parameters. The function s is a vanishing function known as the activation function. Typical choices for s include hyperbolic tangent, $\tanh(a) = (1 - e^{-2a})/(1 + e^{-2a})$, or the logistic sigmoid function, $\sigma(a) = 1/(1 + e^{-a})$. A neuron is said to be activated when Φ is greater than its minimum value.

Neural units are combined together forming a layer, so that the layer projects the input vector into a L-dimension space, $\mathbf{\Phi}$: $R^D \rightarrow R^L$:

$$\mathbf{\Phi}(\mathbf{x}) = s(\mathbf{b} + \mathbf{W} \cdot \mathbf{x}), \tag{2}$$

where now the bias is a vector $\mathbf{b} \in R^L$ and the weights $\mathbf{W} \in R^{L \times D}$ a matrix. The activation function is applied point-wise.

An artificial neural network consists of several of such layers interconnected. Connections among different units of the same layer are not allowed here –as opposed to, *e.g.*, Boltzmann machines or probabilistic graphical models–. The network is said to be a feedforward network if connections between units of different layers do not form a cycle –in contrast to recurrent neural networks. The simplest kind of feedforward ANN is a single-unit perceptron, i.e. a binary linear classifier like that of Eq. (1). Notice that this processing unit can only classify linearly separable inputs.

A Multi Layer Perceptron (MLP) [21] is a set of feedforward layers stacked one after the other. The last layer is a single unit classifier, see Eq. (1), making the MLP a non-linear classifier. The intermediate layers are referred to as hidden layers. It is proven that a single hidden layer MLP is sufficient to approximate continuous functions on compact subsets of R^D with a finite number of neurons. However there are substantial benefits when using more than one hidden layer.

Formally, a single hidden layer MLP is a function f: $R^D \rightarrow R$, such that:

$$\begin{aligned} f(\mathbf{x}) &= \Phi^{(2)}(\mathbf{\Phi}^{(1)}(\mathbf{x})) = s^{(2)}(b^{(2)} + \mathbf{w}^{(2)}\mathbf{\Phi}^{(1)}(\mathbf{x})) \\ &= s^{(2)}(b^{(2)} + \mathbf{w}^{(2)} \cdot (s^{(1)}(\mathbf{b}^{(1)} + \mathbf{W}^{(1)} \cdot \mathbf{x}))). \end{aligned} \tag{3}$$

Here, $\mathbf{\Phi}^{(1)}(\mathbf{x}) = s^{(1)}(\mathbf{b}^{(1)} + \mathbf{W}^{(1)} \cdot \mathbf{x})$ constitutes the hidden layer, also denoted by $\mathbf{h}(\mathbf{x})$. On the other hand, $\Phi^{(2)}(\mathbf{h})$ is a single unit classifier.

In the general case of having C output classes, a Softmax layer can be used at the last layer of a ANN classifier, where the activation is defined as

$$s_j := \frac{e^{y_j}}{\sum_{i=1}^{C} e^{y_i}}. \tag{4}$$

Here, j indicates a given neuron, C is the total number of neurons and y_i stands for the ith-component of the linear projection of the input vector $\mathbf{x}$ to the layer, $\mathbf{y} = \mathbf{b} + \mathbf{W} \cdot \mathbf{x}$. The SoftMax layer thus represents a non-linear variant of the multinomial logistic regression setting. Notice that softmax activation is computationally expensive, as it requires knowledge of all the values of the neurons in a layer.

The idea behind this type of layer is that output classes are mutually exclusive; indeed, the Softmax layer tries to take advantage of this fact within the network architecture. Ideally, the most suitable architecture for this requirement would be to directly apply a maximum layer output, which will provide a probability equal to one for the maximum output of previous layer and a probability equal to zero for rest of the output neurons. However, such output layer will not be differentiable, and thus, impossible to train with back-propagation.

The optimization of ANN classifiers is expressed in this work in terms of the cross-entropy between the true value of a label and the distribution of predicted values by the model [22], although other losses such as the sum of the squared differences are very extended [23]. With the cross-entropy loss, the minimization process implies making the probability of the predicted class as close as possible to the true label.

3.2. Learning the parameters

In a feedforward ANN the information flows from the input to the output layer. The initial information is propagated through the network in training time until it produces a scalar cost $J(\boldsymbol{\theta})$, parameterized by a set of parameters $\boldsymbol{\theta}$ – in the single hidden layer MLP, Eq. (3), $\boldsymbol{\theta} \equiv \{\mathbf{b}^{(1)}, \mathbf{W}^{(1)}, b^{(2)}, \mathbf{w}^{(2)}\}$. This process is known as forward propagation. In order to minimize the cost J, the information must flow backwards through the network, so that the gradient of the cost respect to the parameters can be computed. This is called back-propagation, see e.g. [24]. The gradient can be expressed in terms of the operations defined in each layer by applying the chain rule of calculus.

The updates of the learning parameters $\boldsymbol{\theta}$ can be calculated using gradient descent as

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \tag{5}$$

where η stands for the learning rate which determines the size of the steps we take to reach a (local) minimum. As observed, gradient descent updates the parameters in the opposite direction of the gradient of the objective function $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. Roughly, it follows the direction of the slope of the surface created by the objective function downhill until a valley is reached. As we need to calculate the gradients for the whole dataset to perform just one update, gradient descent can be very slow and it is intractable for datasets that do not fit in memory.

In contrast, stochastic gradient descent (SGD) performs a parameter update for each training example $x(i)$ and label $y(i)$ [25]:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; x(i), y(i)). \tag{6}$$

Notice that this expression approximates the true value of the gradient by its value at a single training point. This allows one to parallelize the code, but induces at the same time large fluctuations in the parameters $\boldsymbol{\theta}$. Although such fluctuations may ultimately complicate the convergence to the exact minimum, this same fact enables the iteration process to jump to new and potentially better local minima, in contrast to normal gradient descent operation, which converges to the minimum of the basin where the initial parameters are placed in.

There are several ways of improving SGD that we use in our training. In particular, the use of mini-batches to compute the gra-

dients at more than one training point in each iteration, which smooths the fluctuations provoked by SGD; and the use of adaptive learning rates, such as momentum techniques.

There are many non-linear functions that can be used to activate a neuron in Eq. (1). A Rectified Linear Unit (ReLu) [26] is a neuron with an activation function defined as $s(x) := \max(0, x)$. In contrast to the vanishing property of hyperbolic tangent or the sigmoid activations, the ReLu does not go to a finite value for large values of the input. This helps to prevent zero gradients when back-propagating the error through large networks. For this reason ReLu units have become one of the most popular activation functions for Deep Neural Networks [27].

A smoother and differentiable variant to ReLu units is the softplus function [28], $s(x) := \ln(1 + e^x)$. The derivative of this function is the traditional logistic function. ReLu units are widely used in many applications such as computer vision or speech recognition, see e.g.[29,30].

4. Methodology

ANNs are seamless representation and classification models that learn simultaneously an appropriate hidden representation of the input data and a discriminative model that connects these hidden representations with the discriminant information. Recently, ANNs have grabbed attention in many areas, such as biomedicine [31], mainly for two reasons: the very good results obtained in real problems such as image recognition or natural language processing; and the possibility to train them with very large data sets on clusters of graphical processing units (GPUs). In contrast to other classification problems, fraud detection entails a very difficult detection problem: first, it is an extremely unbalanced problem where the ratio between genuine and fraudulent transactions is greater than 5, 000: 1; second, fraudulent transactions use to mimic genuine transactions in order to avoid being detected. These two issues lead to a very complicated scenario where standard techniques fail.

There are different techniques to deal with imbalance data but not all of them scale properly to millions of samples. Similarity based methods like k-nn and oversampling [32] does not scale properly. The k-nn is a direct estimator of the posterior probability thus is not affected by the prior. However k-nn scalability, even when approximate search is used, is problematic. Moreover k-nn and oversampling techniques are quite sensible to outliers, and fraud detection is a problem where noise labelling is present and editing techniques aim to remove outliers does not scale properly to such as large scale problems. On the other hand, cascade of classifiers has demonstrated to be a very feasible way to deal with the unbalanced problems. In fact we find lot of similarities between fraud and face detection where cascade of classifiers is among the state of the art, see for instance [33] a face detector based on a cascade of Adaboost classifiers. In fact, the fraud detection problem resembles the face detector problem where the number of pixels where a face is not present is much larger than the number of pixels where a face is present. In our problem, as the cascade gets deeper, more and more samples of genuine transactions are discarded,making the last classifier in the cascade better suited for distinguishing between the hard examples of fraud and the ones that are genuine.

Therefore, we propose the use of a cascade of two filters to deal with the first problem (unbalanced ratio fraudulent/genuine transactions), and a neural network classifier to deal with the second problem (fraudulent transactions use to mimic genuine transactions). As explained in the Introduction, the two filters are implemented as an ensemble of ANNs, where each single ANN is trained as a binary classifier. Thanks to using ensembles of ANNs, it is not necessary to adjust a threshold for making the decision at the output of each filter. Those transactions rejected by the cascade of filters are the ones the system accepts as genuine ones and will not reach the final neural network classifier.

4.1. Data preprocessing

Raw data consists of the records, known as *transaction messages*, that every bank stores from credit or debit card transactions. We process the stream of records in order to obtain a representation of each transaction message as a feature vector. Feature vectors are composed of both aggregated and non-transformed attributes, a total of 62 features. Appendix A contains a description of the attributes used for composing each feature vector $\in R^{62}$.

4.1.1. Splitting of the dataset

The dataset contains transactions corresponding to card purchases made during 18 months, from January 2014 to June 2015. But June 2015 was not used because there is a lack of fraud information for some days of this month. In order to simulate the regular operation of fraud-detection systems in real scenarios, we trained classifiers with data of previous months for predicting a particular month. Since annotated fraud messages are not available immediately we never considered the previous month for training. As an example, for testing with data of March 2015, we never considered to train using data of February 2015.

ANNs used for the definitive decision were trained with data of six months, then, for testing March 2015 it was used the data from August 2014 to January 2015.

In order to prepare filtered data, first and second level filters were trained in a similar way but using two months (due to the very large scale before filtering) instead of six. Therefore to obtain the filtered data of August 2014 the second level filter was trained with data of May and June of 2014. Again the previous month, July 2014, was not used. Finally, to obtain the filtered data of May 2014, the first level filter was trained with data of February and March of 2014. January 2014 was used for estimating the initial values for the aggregated attributes used for preparing the input to ANNs, in particular $P(fraud|currency)$, $P(fraud|country)$, $P(fraud|trader)$, and $P(fraud|card)$. These aggregated attributes are updated every month. According to this, we can test last three months.

4.2. Cost sensitive metrics

As aforementioned, credit card fraud detection is by definition a cost-sensitive problem, in the sense that the cost produced by a false positive is different than the cost of a false negative. Normally this kind of problem is assessed by means of the area under the ROC; in fact some of the recent state-of-the art works on fraud detection used this measure [34]. However, in this paper we also evaluate two other quantities: the Value Detection Rate (VDR) and the True False Positive Ratio (TFPR). The former, VDR, is defined as the amount of money associated to fraudulent operations that are detected by the system. The later, TFPR, is the ratio between the total number of transaction blocked by the system and the number of fraudulent transactions blocked.

The VDR is formally described as the ratio between the total amount of informed fraud and the fraud amount detected:

$$\text{VDR} = \frac{\text{fraud detected}}{\text{total fraud}}. \tag{7}$$

For the sake of readability of the results, VDR is usually expressed in percentage.

On the other hand, TFPR aims at evaluating the cost of a false positive, and it is much more challenging because it affects to the

Table 1
Topologies used in the first and second level filters of the cascade. Several topologies were tested for each filtering level. In all cases the output layer has two neurons because ANNs are used here as binary classifiers (genuine/fraudulent), and the number of neurons of the input layer is the dimensionality of the samples. The activation function of all hidden layers is the hyperbolic tangent and the activation of the output layer is the softmax. The performance of each filter is presented as VDR and the reduction in average of the ratio of genuine/fraudulent transactions.

Filter	Topology	VDR	Ratio
Original data	–	–	> 5000: 1
First level	$62 \times 500 \times 500 \times 2$	93.7%	420: 1
Second level	$62 \times 500 \times 500 \times 500 \times 2$	66.4%	100: 1

social credibility of the bank security measures. The goal is to minimize the amount of false positive but at the same time do not block many genuine transactions. TFPR is formally defined as:

$$\text{TFPR} = \frac{\text{TP} + \text{FP}}{\text{TP}}, \tag{8}$$

where *TP* is the True Positives and *FP* the False Positives.

4.3. Cascade of classifiers for filtering

The goal of a filter is to remove as many genuine transactions as possible while conserving the maximum number of fraudulent transactions. Instead of using a unique filter, a cascade of filters based on ensembles of ANNs is applied for this purpose. The performance of filters is measured with the VDR defined in Eq. (7) and the ratio of *genuine vs fraudulent* transactions. Here, VDR refers to the percentage of fraud the system preserves after each filter in the cascade. The amount of fraud corresponding to those fraudulent transactions rejected by the filters is the fraud the system will never recover.

Experiments were conducted for testing different ANN topologies and depths of the cascade of filters. We found that two levels of filtering were sufficient to reduce the ratio of genuine to fraudulent transactions to acceptable levels. The chosen topologies for these two filters are presented in Table 1 jointly with their performance. In both levels of filtering 10 ANNs per ensemble were used, no improvements were observed in the accuracy of first and second level filters when increasing the number of ANNs per ensemble. All the ANNs in an ensemble had the same topology. The differences among ANNs in the same ensemble were the weight initialization and the order the samples were processed, because samples are shuffled at each epoch.

Original data as it was provided has an overall ratio of *genuine/fraudulent* transactions greater than 5000 : 1. The first level filter reduces this ratio to around 420:1, a reduction slightly greater than 90% of the total amount of transactions with a 93.7% of preserved VDR. The second level filter reduces the ratio close to 100: 1, but in this case the preserved VDR is lower, around 66.4%. In all cases both quantities, the ratio of *genuine/fraudulent* transactions and the VDR, are reported in average for all months used for testing. It can be the case of days where these values are far from the average. But the goal in this task is to know the behavior of the detector system in the long term, that is why the values of interest are the average of these measures.

Table 1 summarizes the results of the filtering cascade. Two filters have been enough to achieve a ratio affordable by the definitive ANN used to classify transactions. In other words, adding more filters did not improve the results of the whole system in the task of detecting fraudulent transactions.

5. Experiments

In order to asses our approach we run experiments with a database of 18 months of transactions. We have tested our approach in the last three months using the previous months to train the models.

Table 2
Results for each month.

Month	TFPR	VDR	AROC
March 2015	4,25	13,8%	0,884
April 2015	3,95	10,8%	0,849
May 2015	4,94	23,9%	0,871

As explained in Section 4.2, we use the VDR and TFPR metrics to evaluate our model from a business point of view. As each blocked transaction must be further analyzed by a human, TFPR ratio values should be around 4 [35]. This ratio is a good compromise to avoid to saturate the back-end alert system managed by humans within the bank. In order to properly adjust TFPR, both metrics (VDR and TFPR) are computed once the network is trained considering all the possible threshold values (from 1 to 99), then experts select the final threshold score before put the network in production for the next months.

5.1. Last Neural Network and results

After the cascade of filters we use a MultiLayer Perceptron (MLP) with ReLu hidden units and softmax output units for classifying transactions as genuine or fraudulent. In order to improve the stability and convergence of the SGD backpropagation we apply different techniques: drop out, adaptive learning rate, batch normalization or Maxnorm regularization.

We apply Batch Normalization (BN) [36] to all the layers. Usually, ANN convergence tends be slow because the distribution of each layer's inputs changes during training as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization. BN aims at eliminating the internal covariate shift in order to ease the convergence. A key issue is that normalization is a part of the model, therefore backpropagation gradients flow through this process as well. BN allows to use much higher learning rates and be less careful about initialization in all the layers.

On the other hand, we use maxnorm regularization. Maxnorm is a regularization that restricts the norm of the weights to a given value c as maximum, thus preventing from a runaway growth of the weights. Hence, the possible space where the neurons can learn from is set to a hypersphere of c radius. We set the maxnorm parameter c to 3.0.

One important issue is to define the ANN topology. Here we propose to use *simultaneously* several topologies with one and two hidden layers and combine all these topologies in order to get the final score. This approach allows us to avoid to test any possible topology but use all (a weighted combination). Moreover, this ensemble provides an effective regularization of the network. The resulting neural network has 4 hidden layers in the deepest path. To this end we use a publicly available toolkit *Layers* [37] that allows to create any kind of feedforward neural network. In this work we have combined neural networks with 64, 128 and 256 hidden units. The model combination is performed with an extra hidden layer with 16 units, see Fig. 2 for more details.

As mentioned before, the experiments were run over the last three months of transactions using the previous months for learning the network parameters. For these three months the TFPR, VDR and AROC (Area of Receiver Operating Characteristic) are measured and displayed in Table 2. Moreover Fig. 3 shows the ROC plots.

The AROC results are very similar to the results reported in [38]. Unfortunately it is imposible to perform an exact comparison because the dataset used in [38] is not public, therefore the comparison between both approaches should be considered as a qual-

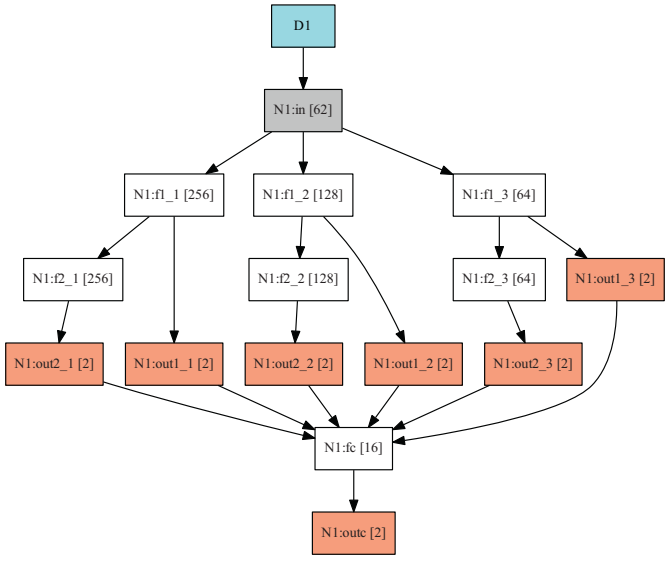

Fig. 2. ANN combination used in the experiments.

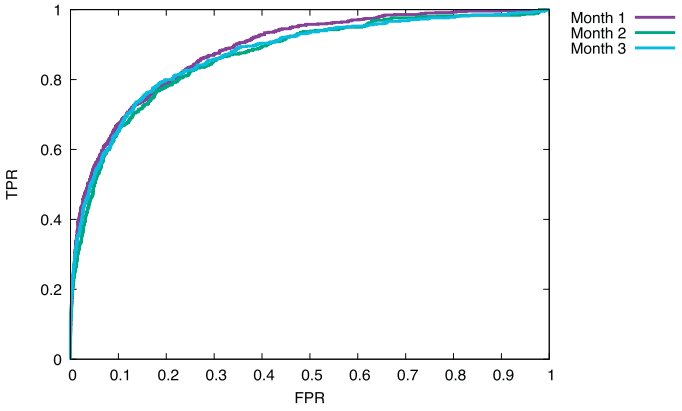

Fig. 3. ROC plot for the three months evaluated.

itative comparison instead of an exact quantitative result. On the other hand, it is important to note the different performance obtained in different months; one of the reasons could be the great variability and seasonality of the fraud techniques employed. These techniques are continuously changing in order to pass the fraud detectors.

6. Conclusions

The need to detect fraudulent credit card operations in almost real time demands the adoption of automatic methods. The goal of our work is to give some insight to data practitioners about how to solve this problem using artificial neural networks, specifically using a cascade of classifiers to reduce unbalancedness and cost metric evaluation.

The experimental part of this paper has shown that using the proposed approach it is possible to achieve state-of-the-art performance with real data.

As future work, we would like to explore other types of DL architectures such as LSTMs (Long Short Time Memory) [39], a type of recurrent neural networks (RNN). The idea behind such architectures is to exploit the inherent temporal information of customer expending behaviors and fraud patterns.

Appendix A. Feature vectors description

Attribute	Description
Amount in EUR	The amount of the transactions in EUR as a float value. The original value is expressed as an integer including the cents.
Amount ratio	The amount in EUR of the current transaction divided by the moving average of this variable. The moving average is computed independently for each card number.
$Pr(fraud \mid currency)$	The probability of fraud per currency computed as the ratio of two counters. The numerator is the *counter of fraud transactions detected for the local currency* and the denominator is the *counter of all transactions corresponding to the same local currency*. Numerator and denominator of a given local currency are updated after processing each transaction.
$Pr(fraud \mid country)$	The probability of fraud per origin country of the transaction. The counters involved for computing this probability are initialised and updated with the same strategy explained for the previous attribute.
$Pr(fraud \mid trader)$	The probability of fraud per trader. The counters involved for computing this probability are initialised and updated with the same strategy explained for previous attributes.
$Pr(fraud \mid card)$	The probability of fraud per card. The counters involved for computing this probability are initialised and updated with the same strategy explained for previous attributes.
Seconds in Day (1)	Time of the operation within the day according to the **operation** timestamp.
Seconds in Day (2)	Time of the operation within the day according to the **original** timestamp.
Time diff. in sec.	Time difference in seconds between the operation timestamp and the original timestamp. Can be negative, but as all the integer or real values this field is scaled to zero mean and variance one.
Time lapse 1	Time lapse in seconds between the current operation and the previous one of the same card number. $time(current) - time(previous)$ i.e. $time(\mathtt{trans}_t) - time(\mathtt{trans}_{t-1})$ The value of this attribute is limited by the number of seconds in a year.
Time lapse 2	Idem but the difference between the current and the previous of the previous. $time(\mathtt{trans}_t) - time(\mathtt{trans}_{t-2})$
Time lapse 3	Idem with respect to three previous transactions. $time(\mathtt{trans}_t) - time(\mathtt{trans}_{t-3})$
Previous rejection	Three bits ($true/false$) indicating if the three previous transactions for the card of the current transactions were rejected for any reason attending to the value in the field *answer code* (OP17) of the transaction message.
Input mode	OP13 in the transaction message. This field is converted into a bitmap with a total of 45 bits ($true/false$). This field has 12 components and each component has a different set of possible values. Not all the components have been used. The used components and the number of bits for representing them are the following: **1** (8 bits), **2** (9 bits), **5** (7 bits), **6** (3 bits), **8** (12 bits) and **9** (7 bits).
Online indicator	Online operation indicator (OP23). One bit ($true/false$).

References

[1] R. Kumar, S. Pranit-Lal, A. Sharma, Detecting denial of service attacks in the cloud, in: IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 2016, pp. 309–316.
[2] E. C. Bank, Annual report, 2015.
[3] E.C. Bank, Card payments in europe – a renewed focus on sepa for cards, 2014.
[4] A.C. Bahnsen, A. Stojanovic, D. Aouada, B. Ottersten, Learned lessons in credit card fraud detection from a practitioner perspective, Expert. Syst. Appl. 41 (10) (2014) 4915–4928.
[5] Y. Sahin, E. Duman, Detecting credit card fraud by Ann and logistic regression, in: Int. symposium on innovations in intelligent systems and applications (INISTA), 2011, pp. 315–319.
[6] Y. Kou, C.-T. Lu, S. Sirwongwattana, Y.-P. Huang, Survey of Fraud Detection Techniques, in: IEEE Int. Conf. on Networking, sensing and control, Vol. 2, 2004, pp. 749–754.

[7] R. Brause, T. Langsdorf, M. Hepp, Neural data mining for credit card fraud detection, in: Tools with Artificial Intelligence, 1999. Proceedings. 11th IEEE International Conference on, IEEE, 1999, pp. 103–106.
[8] P. Save, P. Tiwarekar, K.N. Jain, N. Mahyavanshi, A novel idea for credit card fraud detection using decision tree, Int. J. Comput. Appl. 161 (13) (2017) 6–9.
[9] S. Ghosh, D.L. Reilly, Credit card fraud detection with a neural-network, in: 27th IEEE International Conference on System Sciences, Vol. 3, 1994, pp. 621–630.
[10] J.R. Dorronsoro, F. Ginel, C. Sgnchez, C. Cruz, Neural fraud detection in credit card operations, IEEE Trans. Neural Netw. 8 (4) (1997) 827–834.
[11] A.C. Bahnsen, D. Aouada, A. Stojanovic, B. Ottersten, Feature engineering strategies for credit card fraud detection, Expert. Syst. Appl. 51 (2016) 134–142.
[12] V.V. Vlasselaer, C. Bravo, O. Caelen, T. Eliassi-Rad, L. Akoglu, M. Snoeck, B. Baesens, Apate: a novel approach for automated credit card transaction fraud detection using network-based extensions, Decis. Supp. Syst. 75 (2015) 38–48.
[13] M.I. Jordan, T.M. Mitchell, Machine learning: Trends, perspectives, and prospects, Science 349 (6245) (2015) 255–260.
[14] D.J. Hand, C. Whitrow, N.M. Adams, P. Juszczak, D.J. Weston, Performance criteria for plastic card fraud detection tools, J. Oper. Res. Soc. 59 (7) (2007) 956–962.
[15] S.S. Haykin, Neural Networks: A Comprehensive Foundation, Prentice-Hall, 2004.
[16] M.F. Alonso-Gadi, X. Wang, A.P.d. Lago, Credit card fraud detection with artificial inmune system, in: 5th Int. Conf. of Artificial Immune Systems (ICARIS), Vol. 5132 of Lecture Notes Computer Sciences, 2008, pp. 119–131.
[17] C. Whitrow, D.J. Hand, P. Juszczak, D.J. Weston, N.M. Adams, Transaction aggregation as a strategy for credit card fraud detection, Data Min. Knowl. Discov. 18 (1) (2008) 30–55.
[18] Y. Bengio, Learning deep architectures for ai, Found. Trends Mach. Learn. 2 (1) (2009) 1–127.
[19] H. Li, Z. Lin, X. Shen, J. Brandt, G. Hua, A convolutional neural network cascade for face detection, in: Proc. of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 5325–5334.
[20] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors, coRR abs/1207.0580.
[21] R. Collobert, S. Bengio, Links between Perceptrons, Mlps and Svms, in: 21st Int. Conf. on Machine Learning (ICML), 2004, pp. 23–30.
[22] P.T. de Boer, D.P.K.S. Mannor, R.Y. Rubinstein, A tutorial on the cross-entropy method, Ann. Oper. Res. 134 (2005) 19–67.
[23] C.M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, Inc., New York, NY, USA, 1995.
[24] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Nature 323 (1986) 533–536.
[25] S.I. Amari, Backpropagation and stochastic gradient descent method, Neurocomputing 5 (4–5) (1993) 185–196.
[26] V. Nair, G.E. Hinton, Rectified linear units improve restricted Boltzmann machines, in: Proc of the 27th Int. Conf. on Machine Learning (ICML), 2010, pp. 807–814.
[27] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (2015) 436–444.
[28] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, R. Garcia, Incorporating second-order functional knowledge for better option pricing, in: Proc. of Neural Information Processing Systems (NIPS), 2001.
[29] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: G.J. Gordon, D.B. Dunson (Eds.), Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11), Vol. 15, Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011, pp. 315–323.
[30] A.L. Maas, A.Y. Hannun, A.Y. Ng, Rectifier nonlinearities improve neural network acoustic models, Proc. Inter. Conf. Machine Learnig 30 (2013).
[31] Y. López, A. Novoa, M.A. Guevara, N. Quintana, A. Silva, Computer aided diagnosis system to detect breast cancer pathological lesions, in: Progress in Pattern Recognition, Image Analysis and Applications, Vol. 5197 of Lecture Notes Computer Sciences, 453–460.
[32] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, Smote: synthetic minority over-sampling technique, J.Art.Intell.Res. 16 (2002) 321–357.
[33] P. Viola, M.J. Jones, Robust real-time face detection, Int. J. Comput. Vis. 57 (2) (2004) 137–154.
[34] E. Knorr, How paypal beats the bad guys with machine learning, 2015.
[35] R. Brause, T. Langsdorf, M. Hepp, Neural data mining for credit card fraud detection, in: 11th Inter. Conf. on Tools with Artificial Intelligence, 1999, pp. 103–106.
[36] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. 448–456.
[37] R. Paredes, J. Benedí. http://arxiv.org/abs/1610.01430LAYERS: yet another neural network toolkit, CoRR abs/1610.01430. http://arxiv.org/abs/1610.01430.
[38] V. Ramanathan, Paypal - fraud detection with h2o deep learning, 2015.
[39] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural. Comput. 9 (8) (1997) 1735–1780.