Machine Learning System for Fraud Detection. A Methodological Approach for a Development Platform

Salma El Hajjami, Jamal Malki, Mohammed Berrada, Harti Mostafa, and Alain Bouju

Abstract The democratization and massification use of credit cards lead inexorably to a high number of fraudulent transactions. Generally, the fraud detection is part of the anomaly detection problem. In this field, current approaches and techniques are constantly looking for optimized solutions to detect anomalies. Faced with a massive and growing data volume, these methods are put to the test, and thus lead to a large number of undetected anomalies. Real time fraud detection requires the design and implementation of scalable techniques capable of ingesting and analyzing massive amounts of data continuously. Recent advances in storage, data analytics processing, and open-source solutions open up new perspectives in the anomaly detection field and in particular fraud. In this article, we are interested in the design of a fraud detection system (FDS) based on open-sources Big Data technologies. Thus, a general methodology is proposed based on the formalization, the implementation and the technical design of a platform for fraud detection. The formalization part consists of four layers: distributed storage, data processing, model building, and finally the model evaluation. The implementation part uses Spark distributed data processing system. In particular, we are based on its framework dedicated to machine learning, called MLlib. The technical design part of the platform is based on the latest Big Data technologies such as Hadoop, Yarn, Livy etc.

Keywords Machine learning · Anomaly detection · Fraud detection · Big data · Spark and Hadoop platforms

---

Projet UE-FEDER PLAIBDE https://plaibde.ayaline.com

S. El Hajjami (✉) · M. Berrada
IASSE Laboratoire, ENSA, USMBA, Fès, Marocco
e-mail: salma.elhajjami@usmba.ac.ma

J. Malki · A. Bouju
L3i Laboratoire, La Rochelle Université, La Rochelle, France

H. Mostafa
LIMS Laboratoire, FSDM, USMBA, Fès, Marocco

S. Motahhir and B. Bossoufi (eds.), *Digital Technologies and Applications*,
Lecture Notes in Networks and Systems 211,
https://doi.org/10.1007/978-3-030-73882-2_10

1 Introduction

In recent years, the volume of credit card transactions has increased considerably due to the popularization of their uses, the rapid development of associated services, such as e-commerce, e-finance and all mobile payments. The large-scale adoption of credit card coupled with the development of different use cases where transactions take place without rigid verification and supervision inevitably result in large losses [1]. In this area, it becomes necessary, even crucial, to immediately process the collected data to detect any potential fraud. However, traditional detection tools are unable to handle the captured data volume [2]. Therefore, they cannot detect anomalies and threats. It has therefore become essential to overcome the bottlenecks of existing techniques by using a new generation of artificial intelligence approaches, such as those based on machine learning. These approaches must work closely with techniques for data processing, known as Big Data. They must also provide near real-time responses.

Ultimately, we believe that Machine Learning (ML) models can provide a reliable solution to the anomaly detection problem and therefore its application to the fraud detection [3]. Depending on the needs and scope of ML, there are several scenarios, how these models can be constructed and applied. Recent work in the field of ML highlights two kinds of important platforms: development platform and deployment platform.

Development platforms dramatically reduce the time and cost of building ML models. They guarantee the organization of all levels of maturity of these models. They can be out of the box or tailor-made, based on open-sources or commercial software. Some development platforms can act as model deployment platforms as well, and in a few cases, model life cycle management platforms, but their core is running model building pipelines. The deployment platform brings the models to the production environment. Suppose an ML model has been created and evaluated on a given development platform, the next step is to use that model in a production application. The latter can be a simple web application, embedded in an IoT device, or a service or micro-service forming part of a complex architecture. This is the role of the deployment platform.

The two platforms are linked by what is known as the machine learning pipeline. Indeed, the life cycle of a model should not stop at the development and evaluation stage. It is put into production and then returned to the development stage as much as possible to be refined and improved.

In this work, we present a methodology for building ML models in a development platform approach based on open-source software. This platform dedicated to a fraud detection system (FDS) brings two important contributions:

– A fraud detection methodology which contains most of the design ideas common in the latest FDS, which greatly facilitates the integration of detection algorithms into the workflow.
– A four-layer framework that includes a distributed storage layer, data processing layer, model building layer, and finally model evaluation layer.

– A development platform whose central core is based on Hadoop and Spark. With these technologies, we are able to build a scalable and reliable system.

The remainder of the paper is organized as follows. In Sect. 2, data and system challenges for fraud detection are described. Section 3 describes our methodology for building ML models for fraud detection. The results of the experimental evaluation are illustrated in Sect. 4. Section 5 gives all the details about the development platform. Finally, conclusion and future work terminate our article.

2 Fraud Detection: Data and System Challenges

The design of an effective, real-time, and scalable based FDS is subjected to several data and system challenges enumerated as follows [4–7]:

– Data imbalance: generally, fraudulent transactions represent less than 0.05\% of total transactions. This ratio results in the very imbalanced dataset. A good FDS should be able to handle asymmetric distributions of data.
– Data overlap: some fraudulent patterns behave like normal ones. A good FDS should be able to detect fraudulent transactions that mimic legitimate transactions.
– Concept Drift: FDS targeting fraudulent behavior suffer from the fact that in the real world, the profile of normal and fraudulent behavior changes over time. So, a model that performs well over a period of time can quickly become obsolete and produces inaccurate predictions. Therefore, a good FDS should be dynamic and able to adapt to changes in fraudulent patterns.
– Evaluation metric: evaluation metric used for fraud detection techniques should be chosen with care. Indeed, some measures such as accuracy are not suitable for asymmetric distribution.
– Misclassification cost problem: cost of misclassifying each transaction varies. Fraud detection is a very cost sensitive area. Generally, undetected fraudulent transactions are much more serious and costly than the detection of normal behavior as fraud. Therefore, a good FDS should prioritize transactions with a higher misclassification cost.
– Lack of Real-Time FDS: most of the existing FDS reported in the literature work on archival data to drive future security policies. The fraud must be detected and blocked-in real time to avoid future fraud.

3 Machine Learning Methodology and Application to Fraud Detection

The fraud detection is a Big Data problem. Our ML methodology and application to fraud detection takes into account all the data properties intrinsic to a Big Data system. It includes four main layers: data collection and storage, data processing, model building and evaluation before pre-production.

3.1 Data Collection and Storage

This layer consists of collecting data transactions made by customers. In their raw state, they contain information classified as confidential. The vast majority of payment systems provide interfaces to access transaction data. Legal data protection rules must be observed.

The quality of ML models is measured, among other things, by the quality of the data from which they are built. Therefore, knowing how to use good data collection practices is essential to develop high performing models. The data must be, as much as possible, error-free and contain information relevant to the task at hand [2].

3.2 Data Processing

Data processing is an important and time-consuming task due to its importance to overall performance. The purpose of data processing is to create what is called prepared data. Data processing includes various operations:

1. Data cleaning: deletion or correction of records containing corrupted or invalid values, or for which a large number of columns are missing.
2. Data transformation: converting a numeric characteristic to a categorical characteristic and converting categorical characteristics to a numeric representation. Some models only work with numeric or categorical characteristics, while others can handle characteristics of different types.
3. Feature Selection: selecting a subset of the input features for training the model while ignoring irrelevant or redundant ones.
4. Data Sampling: sampling the dataset before training the predictive model in order to have more balanced data.

3.3 Model Building

Model development begins by partitioning the datasets into one dataset used to train a model, another dataset used to test the trained model. This splitting of the data ensures that the model does not remember a particular subset of data. There are two steps in a learning model building:

Table 1 Kaggle credit card fraud dataset details

Transactions	Majority class	Minority class	Columns
284 807	284 315	492	31

– Training phase: mainly consists of the building or adjusting a model. The fundamental goal is to learn a pattern of trends that lend themselves well to generalization to new data instead of simply memorizing the data he was able to see during his training.
– Test phase: once the model is trained, it is important to check whether it behaves correctly on new examples that are not used for training the model. To do this, the model is used to predict the response on the test dataset. Then, the predicted target is compared to the actual response to measure the performance of the model.

3.4 Model Evaluation

It's important to test, measure and monitor the performance of a predictive model. We must then define measures to be used for performance evaluation. The assessment metrics used depends on several factors. Such as, the task of modeling (classification, regression or segmentation), the context of the problem we are trying to solve as well as the data distribution. These metrics are also used to compare the model's performance and select those that give the best performance.

4 Credit Card Fraud Detection Use Case: Experimentation and Evaluation

4.1 Data Collection

For data collection, we use the Kaggle Credit Card Fraud Detection dataset[1] to illustrate these different steps. It contains the transactions carried out by credit card during two days of September 2013 by holders of European cards. The Table 1 provides statistics for the dataset and shows that the minority class (fraud) accounts for *0.172%* of all transactions. Therefore, this dataset is very imbalanced [8]. It contains 31 digital features. Because some of the input features contain financial information, the PCA transformation of 28 digital input features (named V_1, …, V_{28}) was performed due to privacy concerns. Three of the given features have not been transformed. *Time* feature displays the time between the first transaction and each other transaction in the dataset. The *Amount* feature is the value of the amount

[1]Anonymized credit card transactions labeled as fraudulent or genuin From Kaggle https://www.kaggle.com/mlg-ulb/creditcardfraud.

spent in a single credit card transaction. The *Class* feature represents labels and takes two values: value 1 for fraudulent transaction and value 0 otherwise.

4.2 Data Processing

During the processing phase, we proceed to remove missing data. Then, it was analyzed and all features except Amount and Time were scaled using the PCA transformation technique. Therefore, the Time and Amount columns are scaled and normalized to ensure consistency.

For selection features, we are only interested in data features that are able to separate the two classes (fraudulent, normal). Visualization techniques can be helpful in this process. Consider the example shown in Fig. 1 showing the class distribution for some features of our dataset. We can see for V_{12} a significant divergence in the distribution of the two classes. It is therefore a feature with high predictive power. We therefore keep it when building models. Similarly, we can see for feature V_{13} that the distribution of normal transactions (majority class) corresponds to the distribution of fraudulent transactions (minority class). This feature cannot effectively contribute to the separation between the two classes, we eliminate it from the dataset. We carried out this process for all the 28 features. As a result, 11 relevant features were selected for our experiments: V_3, V_4, V_9, V_{10}, V_{11}, V_{12}, V_{14}, V_{16}, V_{17}, V_{18} and V_{19}.

The main difficulties are the skewness and data overlap in fraud detection cases. Our goal is to correctly classify the fraudulent transactions. So, we use One Side Behavioral Noise Reduction (OSBNR) [9, 10] as a sampling approach. This approach manages behavioral noise in order to improve the classification of fraudulent transactions.

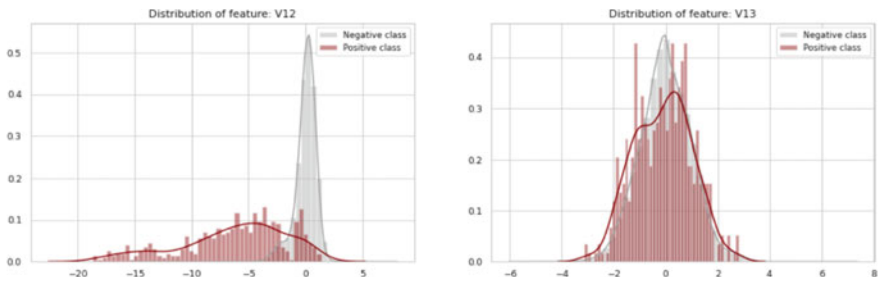

Fig. 1 Class distribution histogram on some features

4.3 Model Building

Since there is no rule of thumb for dividing a data set into training and test sets, we choose the *70/30* rule for training/test sets. Several studies reported Random Forest and Multilayer Perceptron to get the best performance [1, 3, 11, 12], this is one of the reasons we are adopting Random Forest and Multilayer Perceptron in our experiments:

– Random forest (RF) is an algorithm that consist of many decision trees. This algorithm works best when there are more trees in the forest. Each decision tree in the forest gives results. These results are merged in order to obtain a more precise and stable prediction [13].
– MultiLayer perceptron (MLP) is an artificial neural network with direct action which is made up of at least 3 layers of nodes: entry layer, hidden layer and exit layer. Each node uses an activation function. The activation function calculates the weighted sum of its inputs and adds a bias. This allows us to decide which neuron should be removed and not taken into account in the external connections.

4.4 Model Evaluation

Various metrics can be used to measure the predictive accuracy of a model. In our case, the major challenge is to tackle the imbalance problem, since legitimate transactions are much more numerous than fraudulent transactions (less than *1%* of total transactions). This problem often leads to extremely high accuracy where a model can reach up to *99%* of the prediction accuracy, ignoring the *1%* of minority class cases. In other words, accuracy does not reflect reality in this data imbalance case.

The ML model performance is evaluated using AUC (Area Under The Curve) of the ROC Curve (Receiver Operating Characteristics) [14]. The ROC curve is generated by plotting the true positives rate *TPR*, against the false positives rate *FPR*, on all decision thresholds. The true positive rate *TPR* is the proportion of real positives (fraud) that are correctly identified as the positive class, and the false positive rate *FPR* measures the proportion of real positives (fraud) that are wrongly classified. AUC (1) is a concise measure of the performance of the ROC curve with a single value between 0 and 1, where a perfect model has a score close to 1. In addition, AUC has been shown to be effective for class imbalance [14, 15].

$$\mathrm{AUC} = (1 + \mathrm{TPR} - \mathrm{FPR})/2 \tag{1}$$

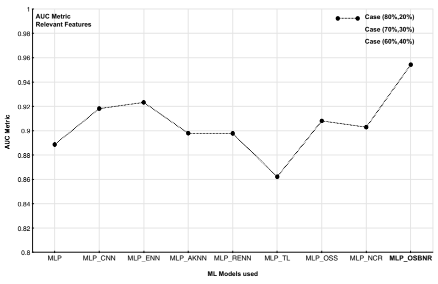

Fig. 2 AUC results for MLP

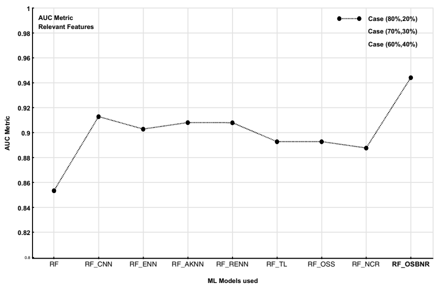

Fig. 3 AUC results for RF

4.5 Results Analysis

Figures 2 and 3 show some results analysis done for both MLP anf RF models using the AUC metric. They also show the model's performance of a new approach called OSBNR. In~\cite{Salma2020b,el2020machine}, MLP and RF algorithms are combined with different resampling methods to study the performance of the resulting ML models.

5 ML Model Development Platform

5.1 General Architecture

Horizontal scaling ensures the platform profitability when data and processing requirements vary between models. Adding more machines to a cluster would not be of much value on its own. What we need is a system that can take advantage of horizontal scalability and that runs on multiple machines seamlessly, regardless of the number of machines in the cluster. The choice of a distributed system for the ML model development platform is then necessary because it operates transparently on a cluster of machines and automatically manages the necessary resources. Figure 4 shows our cluster formed by a master node and three slave nodes. The cluster is

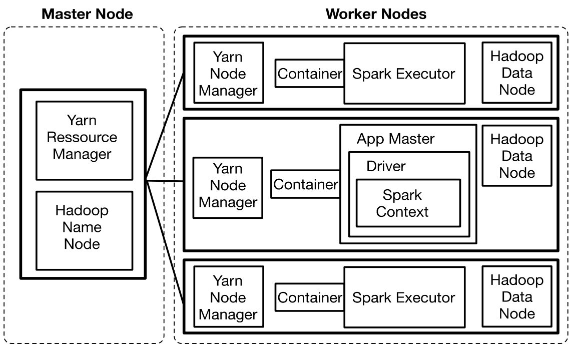

Fig. 4 General architecture of the ML model development platform

managed by a private cloud-type platform within the L3i laboratory as part of the FEDER-PLAIBDE project[2]. It is based on:

- Operating system: Ubuntu-16.04.3-server-amd64
- Hard disk: type SSD 500 Go
- Processors: 16 Go de RAM (4 sockets, 4 cores)
- Network: 2x10 Go SFP +

5.2 *Distributed Data System*

In this design, we made the choice of data storage in raw format. Thus, the system does not depend on any data model. It is therefore natural that the choice of Apache Hadoop Distributed File System (HDFS) is necessary to manage our distributed data system. HDFS is a well-known example of a distributed file system in the big data ecosystem. In HDFS, we use a «master/slave» architecture consisting of a single NameNode which manages the distributed file system and several DataNodes, which generally reside on each node of the cluster and manage the physical disks attached to this node as well as data that is stored physically.

5.3 *Distributed Treatment System*

In our ML model development platform, we use Apache Spark. It is a versatile distributed processing engine capable of handling large volumes of data. The versatility of Apache Spark is that it is suitable for a wide variety of large-scale use cases. In this work, we are particularly interested in its machine learning library called «Spark ML».

[2]https://plaibde.ayaline.com/

Spark uses a «maître/esclave» architecture with a central coordinator called «Driver» and a set of executable workflows called «Executors» located on different nodes of the cluster. Each application written in Apache Spark gives rise to a pilot program or «Driver». It segments a Spark application into tasks which are then partitioned between the worker nodes in the distributed cluster. The pilot program also creates a *SparkContext* that tells the application how to connect to the cluster and its underlying services. The slave nodes are where the computational processing physically occurs. Typically, they are located on the same nodes where the underlying data is available. These nodes generate processes called «Spark Executor». They are responsible for performing computational tasks and storing locally cached data. Executors communicate with the pilot program to receive scheduled functions which are then executed.

5.4 Spark ML Algorithms

Spark ML is an essential software brick of our ML model development platform. It offers all the services dedicated to the implementation of these models. In Apache Spark, can distinguish two APIs:

- Spark ML: based on the DataFrame API. It is a distributed, column-oriented data structure suitable for learning algorithms.
- Spark MLlib: based on RDD API (Resilient Distributed Datasets). This is a distributed, object-oriented data representing data.

As part of our ML model development platform, we use Spark ML. At the heart of it are several layers of software for different services.

5.5 Integrated Development Environment

In our ML model development platform, we use a development environment based on Jupyter Lab. It is a web application for recording the entire process of development, testing, evaluating and documenting ML models. The Jupyter product was originally developed as part of the IPython project. It allows interactive development in several languages. The name Jupyter itself is derived from the combination of Julia, Python, and R. Our development environment provides three cores:

- PySpark: for applications written in Python2.
- PySpark3: for applications written in Python3.
- Spark: for applications written in Scala.

6 Conclusion and Future Work

This work presents a machine learning methodology and a development platform. We presented the fraud detection problem and approached its resolution using an ML approach. Also, we showed that this approach requires the design and implementation of an adequate platform. The presented solution is based on open-source solutions from the world of Big-Data. Our future work covers other aspects of ML modeling and deployment platform.

Acknowledgment This work is carried out thanks to the support of the European Union through the PLAIBDE project of the FEDER-FSE operational program for the Nouvelle-Aquitaine region, France.

References

1. Hajjami SE, Malki J, Berrada M, Bouziane F (2020) Machine learning for anomaly detection. Performance study considering anomaly distribution in an imbalanced dataset. In: Cloudtech 2020. IEEE
2. Roh Y, Heo G, Whang SE (2018) A survey on data collection for machine learning: a big data - AI integration perspective. arXiv:abs/1811.03402
3. Dal Pozzolo A, Caelen O, Le Borgne YA, Waterschoot S (2014) Bontempi G (2014) Learned lessons in credit card fraud detection from a practitioner perspective. Expert Syst Appl 41(10):4915–4928
4. Maes S, Tuyls K, Vanschoenwinkel B, Manderick B (2002) Credit card fraud detection using Bayesian and neural networks. In: Proceedings of the 1st international Naiso congress on neuro fuzzy technologies, pp 261–270
5. Şahin YG, Duman E (2011) Detecting credit card fraud by decision trees and support vector machines
6. Adewumi AO, Akinyelu AA (2017) A survey of machine-learning and nature-inspired based credit card fraud detection techniques. Int J Syst Assur Eng Manage 8(2):937–953
7. Puh M, Brkić L (2019) Detecting credit card fraud using selected machine learning algorithms. In: 42nd international convention on information and communication technology, electronics and microelectronics. IEEE, pp 1250–1255
8. Dal Pozzolo A, Caelen O, Johnson RA, Bontempi G (2015) Calibrating probability with undersampling for unbalanced classification. In: IEEE symposium series on computational intelligence. IEEE, pp 159–166
9. Hajjami SE, Malki J, Bouju A, Berrada M (2020) Machine learning facing behavioral noise problem in an imbalanced data using one side behavioral noise reduction: application to a fraud detection. J Comput Inf Eng
10. Hajjami SE, Malki J, Bouju A, Berrada M (2020) A machine learning based approach to reduce behavioral noise problem in an imbalanced data: application to a fraud detection. In: International conference on intelligent data science technologies and applications (IDSTA). IEEE, pp 11–20
11. Bhattacharyya S, Jha S, Tharakunnel K, Westland JC (2011) Dataminingforcredit card fraud: a comparative study. Decis Support Syst 50(3):602–613
12. Dal Pozzolo A, Boracchi G, Caelen O, Alippi C, Bontempi G (2015) Credit card fraud detection and concept-drift adaptation with delayed supervised information. In: International joint conference on Neural networks. IEEE, pp 1–8

13. Breiman L (2001) Random forests. Mach Learn 45(1):5–32
14. Bekkar M, Djemaa HK, Alitouche TA (2013) Evaluation measures for models assessment over imbalanced data sets. J Inf Eng Appl 3(10) (2013)
15. Jeni LA, Cohn JF, De La Torre F (2013) Facing imbalanced data–recommendations for the use of performance metrics. In: 2013 Humaine association conference on affective computing and intelligent interaction. IEEE, pp 245–251