Fraud Detection Using Apache Spark

Abdelkbir ARMEL
Networks, informatics and mathematics department
National Institute of Posts and Telecommunications
Rabat-Morocco
Email: armel.abdelkbir@ine.inpt.ma

Dounia ZAIDOUNI
Networks, informatics and mathematics department
National Institute of Posts and Telecommunications
Rabat-Morocco
Email: zaidouni@inpt.ac.ma

***Abstract*—Fraud detection methods are continuously developed to defend criminals. They allow us to identify quickly and easily the frauds. In this work, we will focus on the problem of fraud detection in banking transactions. A single algorithm may not be suitable for every problem. Therefore, selecting an algorithm that performs best in a given situation is very crucial. In this work, we give a comparative analysis of four algorithms: Simple Anomaly detection algorithm, Decision Tree algorithm, Random Forest algorithm and Naïve Bayes algorithm. We use the machine learning library (MLlib) of Apache Spark to handle credit card fraud detection. The data used in our simulation is generated randomly following a normal distribution, this data includes two features Price and Distance that allow us to distinguish anomalies and valid transactions. The performance is analysed based on the parameters of the Total Running Time and the Accuracy. The results proved that the Random Forest algorithm gave the best results and the simple anomaly detection algorithm gave the worst results.**

***Keywords*—** Credit card fraud, Fraud detection methods, Machine learning algorithms, Spark MLlib.

I. Introduction

We always listen about the cybercrimes, according to Global Fraud Attack Index 2017: 8.62 % is The total fraud in Q2 2016 and 9.65 % in Q2 2017, so +11.9 % increase in E-Commerce fraud for Consumer Electronics from Q2 2016 to Q2 2017 [1]. It is indeed a real and critical problem we are facing today, effective solutions must therefore be found to minimise the impact of this problem.

In this paper, we are going to focus on the problem of fraud detection in banking transactions. Imagine that we have a bank customer doing his banking transactions always in a regular way in RABAT, then once a costly purchase from PARIS should be a fraud. In this work, we use the framework Apache Spark for performance reasons to handle credit card fraud detection. We chose Apache Spark because it is the most powerful open source project in Big Data with more than 1200 developers that have contributed to Spark, it is rapidly superseding Hadoops MapReduce as the defacto standard for big data processing. Hadoop is an open source, distributed, Java computation framework consisting of the Hadoop Distributed File System and MapReduce (the execution engine).

Spark is similar to Hadoop because its also a distributed, general-purpose computing platform. However, Sparks unique design allows keeping large amounts of data in memory and offers tremendous performance improvements. Spark programs can be 100 times faster than their MapReduce counterparts [2]. It is written in scala language but also support other languages like Python (Pyspark), R and Java. In this study, we will use Scala because it is the native language of Spark. Scala is very fast in execution than Python or Java and also libraries are always updated in scala before python and java.

978-1-7281-1482-8/19/$31.00

In this work, the data used in the experimental section is generated randomly following a normal distribution. This data includes two features Price and Distance that allow us to distinguish anomalies and valid transactions. The performance is analysed based on the parameters of the Total Running Time and the Accuracy.

The rest of the paper is organized as follows. The first section is the state of the art, we present a review of various works that focus on using machine learning techniques for fraud detection. In the second section, we describe the machine learning library (MLlib) of Apache Spark. In Section III, we describe the four machine learning algorithms used in our experimental study which are the Simple Anomaly detection algorithm, Decision Tree algorithm, Random Forest algorithm and Naïve Bayes algorithm. Section IV is devoted to experimental evaluation and results. Finally in section V, we present concluding remarks and perspectives.

II. State of the art

Fraud or anomaly detection is actively and heavily researched. In this section, we present a review of literature and we show some researchers that used several Machine learning techniques for fraud detection.

Harjinder et al. [3] discuss different Machine learning techniques that are useful for anomaly detection. They present Fuzzy Logic technique, Genetic algorithm, Neural Network and Bayesian Network.They also discuss advantages and disadvantages of each technique in the context of fraud detection.

Another work of the author Vaishali [4] used clustering approach for credit card fraud detection. He used K-means clustering algorithm for detecting the transaction whether it is fraud or legitimate. Clusters are formed to detect fraud in credit card transaction which are low, high, risky and high risky.

Sheeraz et al [5] used Apache Spark for analyzing a big dataset for anomaly detection. They performed anomaly detection by using different machine learning algorithms like

Logistic regression, Support vector machine, Naïve bayes, Decision trees, Random forest and Kmeans. They show that all used algorithms are capable to detect anomalies in big data but they need to know how efficiently each performs. Their main objective is to find the most efficient algorithm in the context of anomaly detection. They set to compare their training time, prediction time, and the rate of accuracy.

The Giants of the Internet like Google, Twitter and Netflix have also studied this problem of anomaly detection.

For example, Google uses continuous streams of data. Unexpected drops in traffic can be an indication of an underlying issue. Detecting such drops is non-trivial because streams are variable and noisy, with roughly regular spikes in traffic data. Shipmon et al [6] investigated the question of whether or not they can predict anomalies in Google data streams. Their goal is to utilize Machine Learning and statistical approaches to classify anomalous drops in periodic, but noisy, traffic patterns. They approached the problem in two parts. First they used TensorFlow to train various models including DNNs, RNNs, and LSTMs to perform regression and predict the expected value in the time series. Secondly they created anomaly detection rules that compared the actual values to predicted values. They found that using the intersection of two anomaly detection methods proved to be an effective method of detecting anomalies on almost all used models.

Twitter also released an approach to anomaly detection which considers seasonality using the Seasonal Hybrid Extreme Studentized Deviate test (S-H-ESD) [7]. This approach is build upon the Generalized ESD test for detecting anomalies. S-H-ESD can be used to detect both global and local anomalies. This is achieved by employing time series decomposition and using robust statistical metrics. The technique employed by Twitter is promising as it is suited for breakout anomalies, not just point anomalies.

Netflix also recently released their solution for anomaly detection in big data. It is called Robust Anomaly Detection (RAD) [8]. Their main challenge is high cardinality data sets, especially those with large combinatorial permutations of column groupings which make human inspection impractical. Their solution is settled on uses Robust Principal Component Analysis (RPCA) to detect anomalies. PCA uses the Singular Value Decomposition (SVD) to find low rank representations of the data. This solution helped the Netflix data teams understand and react to anomalies faster which reduces the impact to Netflix customers.

III. Apache Spark MLlib

MLlib is Sparks distributed machine learning library [9]. MLlib targets large-scale learning settings that benefit from data- parallelism or model parallelism to store and operate on data or models [10]. It regroups parallel algorithms that run well on clusters. Some other classic Machine Learning (ML) algorithms are not integrated because they were not designed for parallel models [11]. The idea behind Apache spark MLlib is to make machine learning scalable and useful. Generally, it provides several tools for example [12]: ML Algorithms (including Classification, Clustering, and Collaborative Filtering). Pipelines (for evaluating and tuning ML Pipelines), Persistence (for saving and loading algorithms and pipelines) and Utilities (for data management and optimization algorithms). MLlib is one of the libraries built on top of Spark, as shown in Figure 1:

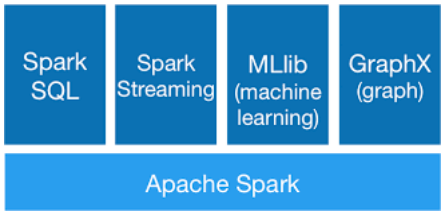


Fig. 1: Spark Ecosystem.

MLlib is written in Scala language [13] and it includes Java, Scala, and Python APIs, and is released as part of the Spark project. It has a large open-source community that is why it has grown rapidly. Spark MLlib includes a large number of algorithms, for example: for classification, we have Logistic regression Binomial and Multinomial, Decision tree classifier, Random forest classifier, Gradient-boosted tree classifier, Multilayer perceptron classifier, Linear Support Vector Machine, Naïve Bayes. For clustering, we have the famous k-means. For recommender systems Collaborative Filtering is commonly used [14]. In this paper, we will try to apply different algorithms (Simple Anomaly detection algorithm, Naïve Bayes Algorithm, Decision trees classifier algorithm and Random Forest algorithm) for the same data, then we compare the performances of each algorithm, and mention the advantages and disadvantages of each algorithm.

IV. Description of Machine Learning algorithms

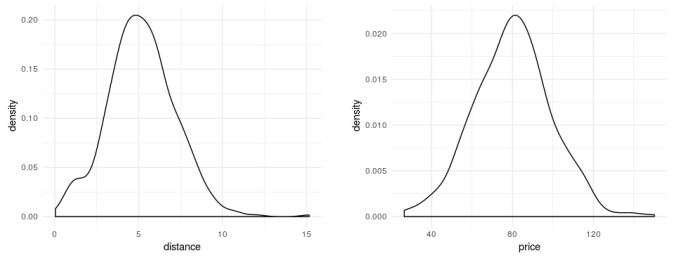
This section describes in detail the three algorithms used in our experimental study.

A. Simple Anomaly detection algorithm

This algorithm allows to build a model p(x) from a dataset. From these dataset, we will calculate an epsilon ε threshold that will maximize a score F1, if p(x) $< \varepsilon$ then, it is an anomaly, otherwise if p(x)$\geq \varepsilon$ then, it is a normal activity. With $x \in \mathscr{R}^N$ and $(x_1, x_2, x_3 ..... x_N)$ these are vectors in the dataset which contain different features, each $x_i \sim \mathrm{N}(\mu_i, \sigma_i^2)$, the visualization of each vector should result in a Gaussian distribution. Each feature x_i is represented by the mean μ_i and the variance σ_i^2: $\mu_i = \frac{1}{m} \cdot \sum_{j=1}^{m} \left(x_i^{(j)}\right), \sigma_i^2 = \frac{1}{m} \cdot \sum_{j=1}^{m} \left(x_i^{(j)} - \mu_i\right)^2$. The F1 score (also called F-score or F-measure) is a measure of a test's accuracy. It considers both the precision and the recall of the test to compute the score. The precision is the number of correct positive results divided by the number of all positive results (True Positives and False Positives) returned by the classifier, and the recall is the number of correct positive results divided by the number of all samples that should have been identified as positive (True Positives and False Negatives). The F1 score is the harmonic average of the

precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst value at 0.

Choosing the Data : In our case, for reasons of simplicity, we have chosen only two features : Price (in euros) and Distance (in kilometers), but nothing prevents to choose other characteristics like the number of transactions, the time elapsed between two transactions, or the type of purchase. The figure 2 represent the density of the price and distance vectors :



(a) Distance distribution (b) Price distribution

Fig. 2: Plotting of the distribution of price and distance

When we visualize the price and the distance together, we can see in the figure 3, for example, if the transaction distance is long and the price is high, it is an anomaly.

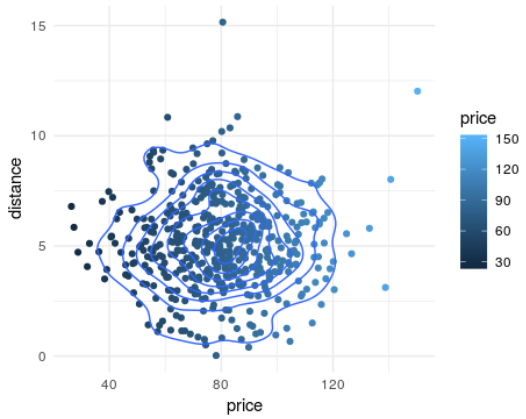


Fig. 3: Distribution of price and distance.

Each cluster groups together a set of points (price and distance) having common characteristics, if a point does not belong to any cluster, then it can be considered as an anomaly.

B. The Naïve Bayes Algorithm

Naïve Bayes is a simple classification algorithm that uses independent data input between each pair of characteristics. The data input often uses the "libsvm" format in the form "Label $< index_1 >:< value_1 > < index_2 >:< value_2 > < index_3 >:< value_3 >.....< index_n >:< value_n >$ ", the label column shows the labels for each transaction if an anomaly or not, and each index is a feature of the our dataset, the value of this index is our point of transactions and the value of those indices is our point in the dataset. The Naïve Bayes algorithm uses Bayes theorem to train a classifier. The model trained by

Algorithm 1 Simple Anomaly Detection algorithm

1. Load the datasets.
2. Convert raw data to vectors.
3. Choose epsilon ε that maximize the score F1.
$F1 = \frac{2*Recall*Precision}{Recall+Precision}$,
$Precision = \frac{TruePositives(TP)}{FalsePositives(FP)+TruePositives(TP)}$
$Recall = \frac{TruePositives(TP)}{FalseNegatives(FN)+TruePositives(TP)}$,
4. Calculate for each feature the probability p(x_{price}) and p($x_{distance}$) using the gaussien function distribution :
$f(x) = \frac{1}{\sqrt{2.\pi.\sigma^2}} \cdot \exp \frac{-(x-\mu)^2}{2\sigma^2}$.
5. Calculate the product p of p(x_{price}) and p($x_{distance}$)
6. **if** p $< \varepsilon$ then anomaly **else** normal.

the Naïve Bayes algorithm is a probabilistic classifier. For a given observation, it calculates a probability distribution over a set of classes. Bayes theorem describes the conditional or posterior probability of an event. The mathematical equation for Bayes theorem is : $P(A|B) = \frac{P(B|A).P(A)}{P(B)}$

In the preceding equation, A and B are events. $P(A|B)$ is the posterior or conditional probability of A knowing that B has occurred. $P(B|A)$ is the posterior probability of B given that A has occurred. P(A) and P(B) are the prior probabilities of A and B respectively. The Naïve Bayes algorithm assumes that all the features or predictor variables are independent. That is the reason it is called naïve. In theory, the Naïve Bayes algorithm should be used only if the predictor variables are statistically independent; however, in practice, it works even when the independence assumption is not valid. Naïve Bayes is particularly suited for high dimensional datasets.

An assumption is also made about the feature distributions (the parameters of which are estimated from the data). MLlib implements multinomial naïve Bayes that assumes that the feature distribution is a multinomial distribution that represents non-negative frequency counts of the features [15].

In our case and as in the previous section and for reasons of simplicity, we have chosen price and distance as characteristics, only in this case we have transformed the "csv" data into "libsvm" using a simple algorithm with Scala language.

Algorithm 2 The Naïve Bayes algorithm

1. Load the datasets.
2. Split the data to be used for testing and training (0.7,0.4)
3. Train a Naïve Bayes Model
4. Use the model to predict and calculate the accuracy
5. Filter the values in the dataset. The predicted value 0 is for normal and 1 is for attack.

Calling n the number of training sample and p the number of features. The complexity of Naïve Bayes algorithm is $O(np)$ for the training and $O(p)$ for the prediction.

C. Decision trees classifier Algorithm

Decision tree model is a powerful, nonprobabilistic technique that can capture more complex nonlinear patterns and

feature interactions. They have been shown to perform well on many tasks, are relatively easy to understand and interpret, can handle categorical and numerical features, and do not require input data to be scaled or standardized. They are well suited to be included in ensemble methods (for example, ensembles of decision tree models, which are called decision forests) [15]. Decision Tree is a flow chart similar to a tree structure, in which every internal node represents a test on an attribute, every branch represents the result of the test, and every leaf node holds a class label. The topmost node in the tree is called the root node. When a tuple T, is given for which the related class label is not known, the attribute values of the tuple are tested alongside the decision tree. A path is outlined strating from the root node to a leaf node that holds the prediction of the class for that tuple.

In our case the choice of the data always remains the same, the price and the distance. Decision Tree makes it possible to predict to which class the target variable belongs, in this case the prediction is a class label, it will try to divide the data into classes and it is a point very far from the classes, therefore it predicts it as anomaly.

Algorithm 3 Decision trees classifier Algorithm

1. Load and parse the data file.
2. Split the data into training and test sets (0.3 held out for testing).
3. Train a DecisionTree model and give maxDepth, maxBins, and numClasses.
4. Evaluate model on test instances and compute test error.
5. Filter the values in the dataset. The predicted value 0 is for normal and 1 is for attack.

MLlib allows to define several parameters the decision tree model, for example :

- maxDepth : The max depth of the tree where we will stop splitting the nodes.
- maxbins : MLlib computes an approximate set of split candidates. The ordered splits create bins and the maximum number of such bins can be specified using the maxBins parameter.
- numClasses: Number of classes (for Classification only).

Calling n the number of training sample and p the number of features. For the complexity of the Decision Tree algorithm for continuous values using MLlib, first it is necessary to find the best possible distribution would be to sort the data in ascending order along this data point, then to consider all possible partition points between these data points and to take the one that minimizes the Gini impurity in our case, this step it takes a time of $O(n \log n)$, we do this for each of the p features. Thus, the final complexity for the training is $O((n \log n)p)$ and the complexity for prediction is $O(p)$.

The figure 4 below represents a simple decision tree used in our case.

In the next section, we will move to a more advanced decision tree principle, we will talk about the random forest when we find several decision trees in a forest, so it will

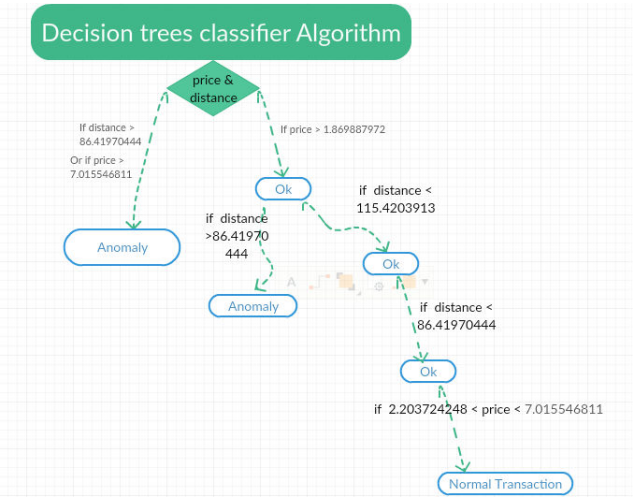


Fig. 4: A simple decision tree

be more efficient on term of accuracy but these decision tree algorithms need many features to get accurate results, which is not the case in our case, we always use the same data to compare the performance of each algorithm.

D. Random Forest algorithm

Random forests are ensembles of decision trees. Random forests are one of the most successful machine learning models for classification and regression. They combine many decision trees in order to reduce the risk of overfitting. Like decision trees, random forests handle categorical features, extend to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions [14]. The concept is similar to the decision tree, a decision tree set builds a forest, the decision tree also is built from a data set, using the characteristics, random forest randomly selects features and build multiple decision trees. In our case, we have a classification problem the random forest use the majority vote. The prediction of each tree is counted as one vote for one class. It is anticipated that the label will be the class that receives the most votes.

Random Forest algorithm as the name suggests, is an algorithm that creates forest with a number of trees. Random Forest algorithm is an adaptable ML method able to perform classification tasks as well as the regression tasks. It is a kind of ensemble learning method, in which a group of fragile models merge to form a dominant model.The Random Forest algorithm, gives highly accurate results with the higher number of trees. In Random Forest algorithm, we generate several trees as opposed to a single tree. In order to classify a new tuple based on the attributes, each tree from the multiple trees provides a classification and thus each tree. The figure 5 below represents a simple decision tree used in our case.

Calling n the number of training sample, p the number of features and N_{trees} the number of trees. The complexity of

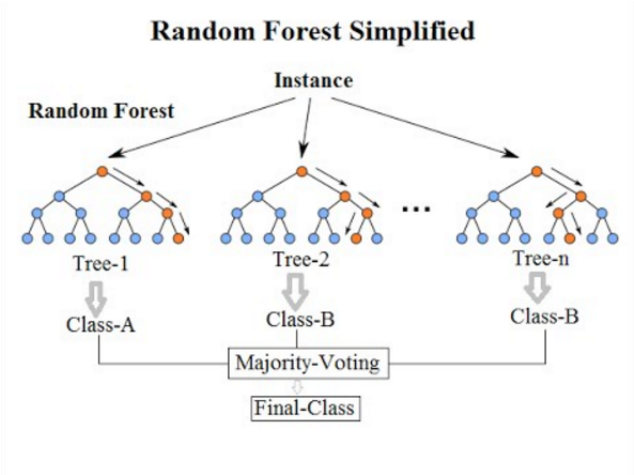


Fig. 5: A simple random forest

Algorithm 4 Random Forest classifier Algorithm

1. Load and parse the data file.
2. Split the data into training and test sets (0.3 held out for testing)
3. Train a RandomForest model and give maxDepth ,maxBins, and numClasses (used for Classification only)
4. Define numTrees and featureSubsetStrategy as 'auto'.
5. Evaluate model on test instances and compute test error.
6. Filter the values in the dataset. The predicted value 0 is for normal and 1 is for attack.

Random Forest algorithm is $O((n \log n) p N_{trees})$ for the training and $O(p N_{trees})$ for the prediction.

V. EXPERIMENTAL EVALUATION AND RESULTS

A. Synthetic data generation

We generate the synthetic data following a normal distribution with a density : $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2 / 2\sigma^2}$

We chose the R programming language which is a free software environment for statistical computing and graphics.

Since the data type is numerical, we used the function **rnorm** (n, mean, var) [16] with the parameters : n = number of observations, mean = μ^2 and var = σ^2 to generate a vector for the price and to generate a vector for the distance between two transactions. Those vectors are distributed normal random. The last step before consuming our data is to transform it into .csv format and save it in the HDFS as a storage system in order to use it.

B. Configuration and environment

In our experimental study, we generated random data transactions containing the price and distance between transactions to apply the different Mllib Spark algorithms.

Table I and II shows the characteristics of the machine on which we launched the simulations.

Hostname	armel-Lenovo-IdeaPad-Z500
IP address	192.168.43.111/24
CPU(s)	4
Hard Disk	1TB
Memory	12GB
CPU max MHz	3600,0000
CPU min MHz	1200,0000

TABLE I: Hardware Information

Apache Spark	2.1.0
Scala	2.11.8
Java	1.8.0_171
Intellig IDEA	2018.1.6
sbt	1.0
OS	Ubuntu 16.04

TABLE II: Software Information

C. Spark jobs

We used the Spark SQL, especially the DataFrame API. It allow us to work with data as tables. The figures below show the different completed jobs of each algorithm and also the total running time :

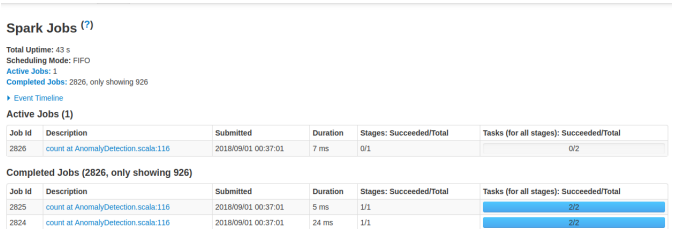


Fig. 6: Simple anomaly detection algorithm

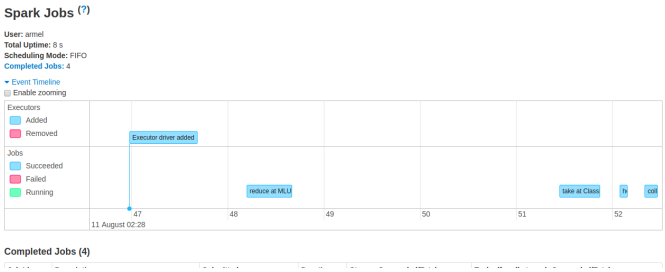


Fig. 7: Naïve Bayes algorithm

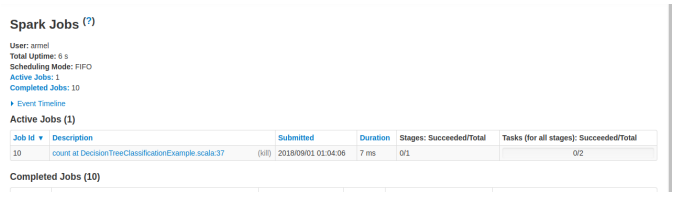


Fig. 8: Decision Tree Classification

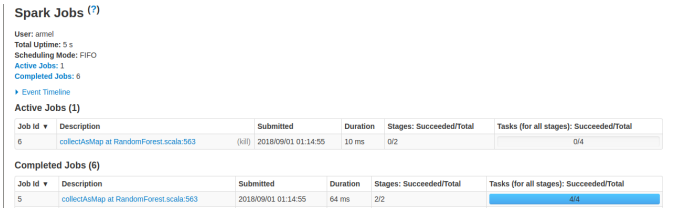


Fig. 9: Random Forest Classification

D. Total Running Time

All algorithms take less time to predict the results, except the first algorithm which is simple to implement but takes more time because in each case you have to calculate the probability and also to choose the bestepsilon and calculate the F1 score. The random Forest gives the best total running time.

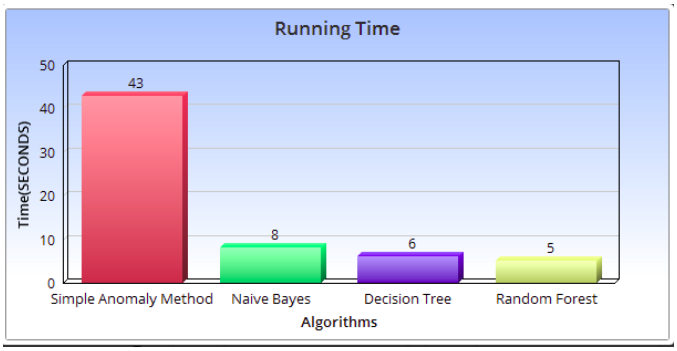


Fig. 10: Running Time

E. Accuracy

The algorithms (Naïve Bayes, Decision tree and Random Forest) implemented directly in spark MLlib takes more than 91% of the accuracy. The random Forest algorithm gives the best accuracy with 98.18%. and the simple anomaly algorithm has the worst accuracy with 77% (based on F1 score). For other algorithms implemented using MLlib, the accuracy is computed using the following function :

```
val accuracy = evaluator.evaluate(predictions)
```

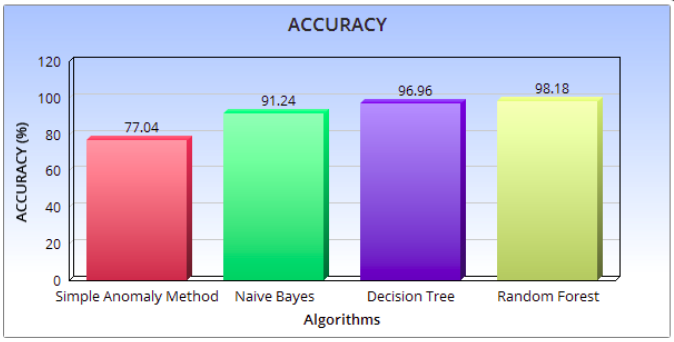


Fig. 11: Accuracy

VI. Conclusion and perspectives

In this paper, we have created distributed normal random data with two features of banking transaction data : Price and Distance, and we have analyzed the datasets with MLlib of Apache Spark as a very powerful big data tool. For some algorithms like random forest and decision tree you need a lot of features to get good results. In our study, we give a comparison of different alogorithms based on two criteria : the total execution time and the accuracy of each algorithm. On one hand, the random forest algorithm gives the best total execution time and is even more accurate. On the other hand, the classical simple anomaly algorithm takes more execution time and gives the worst accuracy.

As perspectives, we will investigate other techniques used for fraud detection (Times series databases and other statistical techniques) and we will use Apache Flink to simulate a real-time processing. Finally, we will also try to get real data transactions for simulations.

References

[1] Global Fraud Index https://www.pymnts.com/global-fraud-index/.

[2] P. Zecevic and M. Bonaci, *Spark in Action*, 1st ed. Shelter, Island: MANNING, 2016.

[3] Harjinder Kaur, Gurpreet Singh and Jaspreet Minhas, A Review of Machine Learning based Anomaly Detection Techniques, in Proc. of the International Journal of Computer Applications Technology and Research,Vol 2, Issue 2,185 - 187, 2013.

[4] Vaishali M.Tech., Fraud Detection in Credit Card by Clustering Approach, in Proc. of the International Journal of Computer Applications,Volume 98, No.3, July 2014.

[5] Sheeraz Niaz Lighari Dil and Muhammad Akbar Hussain, Testing of algorithms for anomaly detection in Big data using apache spark, in Proc. of the 9th International Conference on Computational Intelligence and Communication Networks,DOI 10.1109/CICN.2017.23.

[6] Dominique T. Shipmon, Jason M. Gurevitch, Paolo M. Piselli, Steve Edwards, Time Series Anomaly Detection, in a report of a summer internship at Google.

[7] A. Kejariwal, Introducing practical and robust anomaly detection in a time series, 2015, January 6. [Online]. Available: https://blog.twitter.com/engineering/en_us/a/2015/introducing-practical-and-robust-anomaly-detection-in-a-time-series.html

[8] RAD - Outlier Detection on Big Data, 2015, February 10.[Online]. Available: :https://medium.com/netflix-techblog/rad-outlier-detection-on-big-data-d6b0494371cc

[9] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., ... and Xin, D. (2016). Mllib: Machine learning in apache spark. The Journal of Machine Learning Research, 17(1), 1235-1241.

[10] Xiangrui Meng et al, MLlib: Machine Learning in Apache Spark , Journal of Machine Learning Research 17 (2016) 1-7.

[11] Karau, H., Konwinski, A., Wendell, P., and Zaharia, M. (2015). Learning spark: lightning-fast big data analysis. " OReilly Media, Inc.".

[12] Guide Apache Spark https://spark.apache.org/docs/latest/ml-guide.html.

[13] Spark, A. (2017). *Machine Learning Library (MLlib) for Spark*.

[14] Spark Mllib Docshttps://spark.apache.org/docs/latest/ml-guide.html.

[15] N. Pentreath, *Machine Learning with Spark*, 2nd ed. Packt Publishing Ltd. Livery Place 35 Livery Street Birmingham B3 2PB, UK., 2015.

[16] https://www.rdocumentation.org/packages/compositions/versions/1.40-2/topics/rnorm