A Big Data Classification Solution for Fraud Detection

Ruijie XIONG
CSAI
UNNC
scyrx1@nottingham.edu.cn

Xinyu GAO
CS
UNNC
scyxg1@nottingham.edu.cn

Hongyan DENG
CS
UNNC
scyhd1@nottingham.edu.cn

Boya WANG
CSAI
UNNC
slybw2@nottingham.edu.cn

***Abstract*—Credit fraud detection grows rapidly with the development of electronic payment. IEEE-CIS Fraud Detection dataset was raised for technique innovation to tackle this problem. For the consideration of the size of the available fraud dataset, this project proposed a solution with the combination of the bigdata technique and the machine learning algorithms. Aside from the raised pipeline of the solution, other identified works include a comparision of four implemented machine learning algorithms, three surveys into the computing time with respect to the data size, number of executors and number of cores.**

***Index Terms*—Credit fraud detection, Machine learning, Spark**

I. Introduction

As the rapid development of online shopping, the number of online transaction increases significantly, unfortunately, the cases of online fraud are also improved. Based on Global Payments Report 2015, applying credit card is the most prevalent payment method around the world compared with other payment methods, such as e-wallet and Bank Transfer [1]. Therefore, it is reasonable to infer that credit card fraud is the most popular items among other frauds, since users' personal information could easily be revealed by the hackers [2], which includes card number, CVV code, password etc. In general, credit card fraud could be categorized to three types, conventional frauds, such as stealing the physical card, online frauds, for instance faking the merchant sites by using the sensitive credit card information hacked from the users, and merchant related frauds, which means colluding with the merchants to swindle customers [5].

Furthermore, credit card frauds negatively affects the global finance supported by observable data. In 2013, US retailers lost near 23 billion dollars due to credit card fraud, and the cost increased to around 32 billion dollars in the next year [3]. One of the major causes of credit card fraud is the weak security of users' credit cards. As for the resent years, this issue has been trending alarmingly. According to the Nilson Report, the financial loss of credit card fraud will exceed 35 billion dollars approximately in 2020 [4]. Thus, it is necessary to develop sophisticated and frequently updated credit card fraud detection methods aiming at reducing the tremendous loss.

Among those methods, data mining and machine learning methods to detect frauds are the most popular. Data mining techniques could be applied to extract the most significant information from a large-scale dataset, which involve statistics and mathematics to distinguish normal or abnormal transition information [5]. Nevertheless, data mining techniques could only discover the valuable information, compensating with machine learning techniques the overall method could construct suitable models to detect fraud intelligently.

In this paper, we first apply data mining techniques to clean the raw data. Then, we perform feature engineering, such as Principle Component Analysis (PCA), to extract significant features. For experiment efficiency, we use Spark and its machine learning library, MLlib, to construct pipeline. Several models are trained and tuned in order to classify the data with satisfactory accuracy. Then we investigate the efficiency and scalability of Spark while handling big data with more executors or cores.

II. Literature Review

A. BigData

The term big data is often used to describe massive datasets that are relatively hard to analyze with existing data processing tools [6]. Big data has great potential to increase the efficiency and quality in industries such as healthcare, manufacturing, retail and public administration, benefiting companies and customers [7]. However, processing big data has challenges and differences with traditional datasets in terms of variety, velocity and volume [8]. This requires big data processing tools to be able to cope with very large datasets from various sources in different structures,and to perform all processes in a relatively short amount of time. Popular open source software for big data include MapReduce [9], Spark [10] and Hive [11].

B. Spark

Apache Spark is a popular big data analytics framework developed at UC Berkeley in 2009 and was made open source in 2010 [10]. Spark is integrated in Scala, Java, Python, SQL and R. Spark has various upper level libraries such as Spark SQL for structured data processing, Spark Streaming for stream processing, MLlib for machine learning, and GraphX for graph analysis available to perform tasks in different steps of big data analysis. These upper level librarires are built upon Spark core, which implements the Resilient Distributed Dataset (RDD) abstraction for efficient processing of largescale dataset. Spark monitors a graph of transformations to recover from failure and only computes RDDs when an action has been called, which means RDDs are fault tolerant and lazily evaluated [12]. In addition to RDDs, a DataFrame API [13] in Spark SQL have been developed to integrate relational data processing. A DataFrame represents a table in relational database, it can perform operations supported in relational databases such as querying. Operations similar to those on RDDs and operations integrated in other libraries including MLlib are also implemented with DataFrames. The performance of Spark and MapReduce are compared on tasks including Word Count, Sort, K-means and PageRank using datasets as large as 372 GB and 500 GB [14]. Their results showed that Spark is 5 times faster than MapReduce onK-meansandPageRank and 2.5 time faster for WordCount, while MapReduce outperforms Spark on Sort. It was shown by [15] that Spark can significantly outperform MapReduce in KNN classification on datasets of different sizes using the 8 GB Higgs dataset. Similar results were obtained by [16] that Spark is twice as fast on K-means clustering compared to MapReduce.

C. Machine Learning with Spark

MLlib is a Spark distributed machine learning library that aims to provide solution for large scale machine learning tasks by data and model parallelism, open sourced in 2013 [17]. MLlib implements common machine learning algorithms and supporting utilities with the Pipeline API and provides optimization for scalable machine learning. Additionally, it is also integrated with other Spark libararies such as Spark SQL.

Spark and MLlib has been used in various studies. It was used to develop a Network Intrusion Detection System with the UNSW-NB15 public dataset [18]. A proposed sentiment analysis classification framework was implemented with Spark using two public Twitter datasets with over 1 million records [19]. The result shows that Spark is efficient and scalable for big data analysis. [20] implemented a Hybrid Collaborative Filtering Recommender Engine with Spark using the Movielens dataset containing 20 million ratings. This model outperforms traditional collaborative filtering methods and improves scalability.

D. Solution to Fraud Detection

The survey [21] classified the fraud detection techniques into two general categories: the anomaly detection and misuse detection.

Anomaly detection is based on the unsupervised methodology. In this method, the legitimate user behavioral model (e.g. user profile) for each account is required to be built and then the fraudulent activities will be detected based on it. The abnormal data will be detected by the implemented unsupervised models. External support from human experts is required to provide knowledge for the explanations of the abnormal data detected. The solution to the verified fraud pattern will be built into the detection model.

Misuse detection deals with supervised classification task. The transactions are labeled as fraudulent or not fraudulent. There are numerous model creation methods for a typical two class classification task such as rule induction [22], decision trees [23] and neural networks [24]. The problem of this method is that it can not identify the innovative frauds. For the labelled data is provided, this project adopts the misuse detection.

This project implements the big data technique with machine learning algorithms to implement the credit fraud detection on the IEEE-CIS Fraud Detection public dataset from Kaggle [25]. For this competition dataset was published two years ago, there has been series of solutions proposed. According to competition summary [26], the team winning the first prize implemented three main models: the Catboost, LightGBM and Xgboost. The Catboost with a precision of 96.39% on the public testing data is the best model for fraud detection. LGBM with the precision of 96.17% and XGB with the precision of 96.02% won the second position on the public test set and the private test set from Kaggle respectively. Chen further implemented an improved Catboost [27] model and achieved the accuracy of 98.3% on this dataset. Lightgbm [28] raised by Ge achieved an accuracy of 98.2%, it beat the models of Xgboost, SVM and logistic regression in its experiments. Zhang [29] raised the Xgboost model which achieved an accuracy of 98.1%, which is proved to perform better than the logistic regression with an accuracy of 93.1%, SVM with an accuracy of 95.8% and the random forest model with an accuracy of 96.5%. Yu [30] proposed an deep neural network model with an accuracy of 95.7%, and this NN model was verified to perform better than the SVM, then followed by the logistic regression, and the Naïve Bayes was ranked as the last. The performances of the random forest and the NN model require further comparison.

In summary, the Catboost model was ranked as the best model, then followed by the Lightgbm model. Xgboost is at the third place. The random forest model and NN model are better than SVM and the logistic regression models. Naïve Bayes was ranked as the last.

III. Methodology

A. Dataset Introduction

The dataset used in this paper is called IEEE fraud detection from Kaggle. There are four major files in the dataset: train_transaction.csv with 394 columns, train_identity.csv with 41 columns, test_transaction.csv with 393 columns, and test_identity.csv with 41 columns. The specific information of transaction files and identity files are attached in the appendix A. The transaction files and the identity files are connected by key transaction ID. But not all transaction IDs in transaction files can find matching transaction IDs in identity files. This problem is solved by using transaction IDs in the transaction files as the standard. After combination, the training file has 433 features and 590540 instances.

B. Data Preprocess and Feature Extraction

The processing flow of the feature extraction is as Fig. 1 shown.

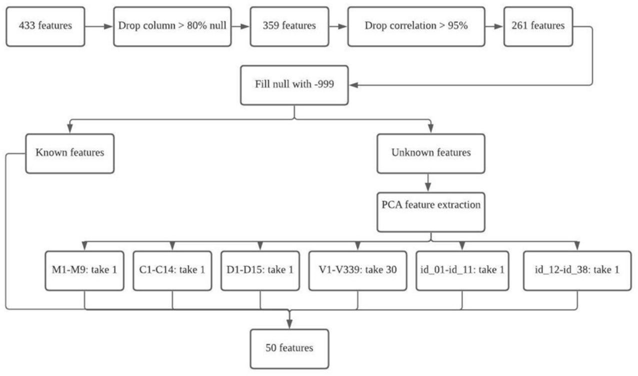

Fig. 1. Workflow of Feature Extraction.

The basic steps of data preprocessing are shown in Figure 1. The first step is dropping features that have more than 80% null values. The reason for choosing 80% is based on other people's experience in processing this data set. Some people have tried to drop more than 87% of features that contain a certain amount of nulls [31]. However, some people have chosen to remove features that contain more than 90% null, and the number of dropped features only accounts for about 3% of the total [32]. Thus, to ensure that a large amount of data is not lost and at the same time, the number of meaningless features can be reduced as much as possible, 80% is our final choice. This dropping null step can remove 74 features. The second step is for C1-C14, D1-D15, and V1-V339 features. The purpose of this step is to reduce the number of special features and prevent the occurrence of overfitting. Take V1-V339 as an example. These 339 features have the same prefix which means that they probably describe the same kind of things. Based on this assumption, the number of such features can be greatly reduced by deleting highly correlated features and finding the representative features among them. To be more specific, calculate the correlation of each feature, if the correlation of two features is greater than 95 %, delete one of the features [28]. This step can remove 98 features. Through the first two steps, the remaining features are considered to affect the performance of the algorithm to some extent. Before using these features, the null values among them must be dealt with first. Using -999 to make up those null values is a feasible way [27]. Although many features can be removed in the first two steps, there are still 261 features left, which will cost a lot of computation and space resources. We rely on PCA to solve this problem because PCA can reduce the dimensionality of the dataset and at the same time minimize information loss [33]. As described in Appendix A, features M1-M9, C1-C14, D1-D15, V1-V339, id_01 - id_11, and id_12 - id_38 are anonymous features with unknown meanings. Although they are anonymous, features with the same prefix is related, thus they are further reduced using PCA. The principle behind this step is to put the features with the same prefix into the PCA and then use the specific number of features generated by the PCA to represent all the features of the prefix.

C. Model Description

1) Support Vector Machine (SVM)
The model of performing classification using linear support vector machines (SVM) is also trained for credit card fraud detection. It is a binary classifier optimizing the Hinge Loss by using the OWLQN optimizer. This model only supports L2 regularization currently. Linear separable datasets are required since the predicted results are not promising for the linear inseparable data. After randomly splitting the training and testing data to 7:3, the model tunning is performed by applying cross validation with 5 folds. The highest accuracy for 10k dataset is 96.46% with the maximum number of iterations 200 and the regularization parameter 1.
2) Multilayer Perceptron (MLP)
A multilayer perceptron (MLP) is a feedforward artificial neural network, which includes at least three layers: an input layer, a hidden layer, and an output layer. In this question, the number of input nodes and output nodes is fixed, 50 and 2 respectively. Thus, the number of hidden layers and the number of nodes in each hidden layer determine the performance of this model. Also, iteration times can affect the result to a certain extent. During the parameter tuning process, the maximum number of hidden layers tried is 2 because for any desired shape, MLP with two hidden layers is sufficient to create classification areas [34]. The highest accuracy for 10k dataset is 96.98% with the maximum number of iterations 500 and 1 hidden layer with 7 nodes.
3) Logistic Regression
Logistic regression is a generalized linear model used for classification problems by fitting a sigmoid function. Logistic regression is usually used for binary classification by calculating the possibility of the two possible outcomes. This model is tuned in terms of number of iterations, the regularization parameter, method and parameter of penalty. The best performance for the 10k dataset is 96.41% with L2 penalty and a regularization parameter of 0.01.
4) Random Forest
Random forest is a type of ensemble learning methods. Multiple decision trees form an ensemble in the random forest algorithm. The decision trees and their predictions are aggregated to produce the result. In random forest classification, the classification result is generated through a majority vote by the decision trees. The main hyperparameters for random forest model are the number of trees, number of features sampled for each tree and the depth of each tree. The highest accuracy for 10k dataset is 96.88% with 30 trees and a depth of 6 for each tree.

D. Model Inplementation

The preprocessing, model training and testing are combined into a pipeline. The four algorithms are tested and tuned using the pipeline with 5 fold cross validation. The model with the best performance is selected as the model used for big data experiments. Different sized datasets are sampled from the original dataset with respect to the ratio of the label classes to ensure the sampled datasets are valid. The sampled datasets contain 1k, 10k, 100k, 200k number of records respectively. This implementation is also tested in terms of number of executors and cores to find the best parallelization method using the 100k dataset.

The performances of the four implemented models are summarized in Table. I. The neural network achieved the best result on the testing dataset and it was selected as the model for further experiments in III-E.

TABLE I
MODEL PERFORMANCE

Model Name	SVM	MLP	Logistic Regression	Random Forest
Precision	0.9646	0.9698	0.9641	0.9688

E. Results and Analysis

Apart from adopting and tunneling different machine learning models, we also attempt to verify Spark's efficiency of dealing with large datasets. As introduced before, Spark is suitable for big data since it is more efficient and scalable than its counterparts, such as MapReduce, concluded from the experiment implemented with Spark using two public Twitter datasets [35]. Since the provided datasets of fraud information are relatively large with more than half a million instances, it is significant to process the data with an efficient tool, such as Spark. Accordingly, the datasets are also suitable for verifying the efficiency and scalability of Spark.

The experiments are performed on our local machine, which obtains Windows 10 Pro operating system and i7 CPU processors with 2.70 GHz, 4 physical processing cores, and 8 logical processing cores. The most updated version, Spark-3.1.1 is used for the experiments under the environment of PYSPARK_PYTHON. For the tests of Spark's scalability, we allocate 20 GB memory for the driver program, which is related to deploy. Then we distribute 1 GB for each executor. This is suggested by the Spark official documentation that allocating only at most 75% of the memory for Spark should be sufficient [36]. Besides, the cluster manager will allocate 4 executors in total and 2 cores for each executor configured through *spark-submit* command line.

The experiment results of driver program executing time are illustrated in Fig. 2. The original dataset is divided into different size of subsets whose unit is kilo. The best tunning model, MLP and its relative parameters are applied to this experiment. After enlarging the size of the test datasets from 1k, 10k, 100k to 200k, the execute time is gradually increased. It is acknowledged that if the correlation coefficient is significantly different from zero, we say that their correlation coefficient is significant, therefore, there is a significant linear relationship between the variables [37]. Analyzing from the experiment results, it is rational to conclude that the relationship between distinguished data sizes and the execute time conform to linear regression since the input values obtain p value = 0.0096, which is less than the significant level 0.05. Additionally, it is straightforward to calculate that R^2 =0.9808, which demonstrates two variables are positive correlated conforming to our common sense that larger datasets require more processing time. Instead of corresponding to exponential relationship, the processing time grows linearly with the increase of datasets. Thus, Spark obtains powerful scalability to handle big data more efficiently with feasible time-consuming.

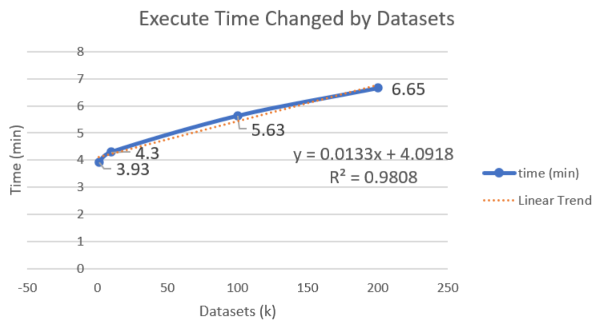

Fig. 2. This experiment measures how the execute time of the driver program changed by different size of the datasets. The size of the datasets increases gradually with 1k, 10k, 100k, and 200k. The blue line illustrates the experiment results in unit of minute and the orange line demonstrates linear trend of the experiment results. The formula below expresses the linear relation between distinguished size of the datasets and the execute time.

Then we investigate how the number of executors affects the processing time of the driver program. By configurating different number of executors and 2 cores for each, their corresponding execute time are shown in Fig. 3. However, the results contrast with our prediction that the process time should reduce as the number of executors promotes. On the contrary, the execute time decrease at first, and then increase after configurating 3 or more executors, which seems unreasonable. The same phenomenon appears when investigating how the number of cores affect Spark efficiency. Two executors are configured in this test and the number of cores distributed for the executors are varied from 1 to 4. Similarly, the best result shows up while configuring 2 cores and 2 executors, and then the process time improve with more distributed cores illustrated in Fig. 4. Results from two tests both show inconformity with linear trend.

Possible explanation is speculated as configured "over-parallelism" issue. It is suggested by [38] that tunning partition should be reasonable as too many partitions lead to excessive overhead in managing many small tasks. As informed by the official documentation that for local mode, the number of cores on the local machine is equivalent to the default number of parallelism [39]. The number of tasks could be processed at once is the total number of cores could be utilized in Spark application. If the configured task number is more than the actual amount of tasks could be processed, other tasks have to wait until there is a core available. For our local machine, there are 4 physical cores available. When the configured degree of parallelism, the number of executors times the number of cores, larger than the actual number of parallel, tasks beyond the partition number have to wait. Therefore, those tests configured executors or cores more than the actual degree of parallel would take more processing time. We can learn from this that in order to execute driver program more efficiently, the setting partition number should be close to the actual parallel number.

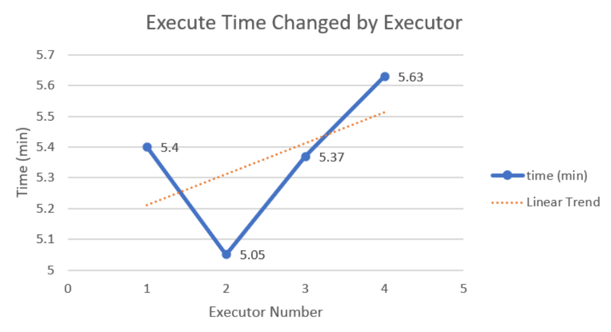

Fig. 3. This experiment measures how the execute time of the driver program changed by different number of the executors. The number of the executors increases from 1 to 4. The blue line illustrates the experiment results in unit of minute and the orange line demonstrates linear trend of the experiment results.

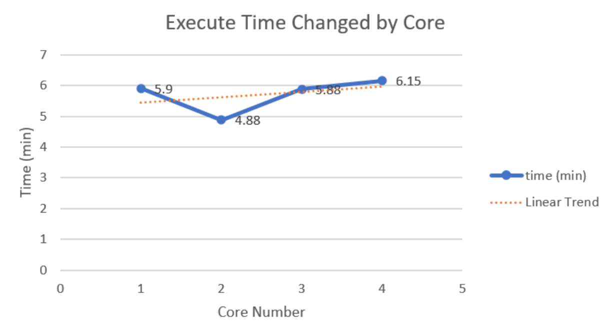

Fig. 4. This experiment measures how the execute time of the driver program changed by different number of cores. The number of cores promotes from 1 to 4. The blue line illustrates the experiment results in unit of minute and the orange line demonstrates linear trend of the experiment results.

IV. Conclusion

In this study, we used an effective feature reduction strategy to reduce 433 features to 50 features. This strategy could significantly reduce the running time in model training and testing, while the accuracy is not compromised. After testing four models, it was concluded that MLP, compared to SVM, logistic regression, and random forest, have the best performance in credit card fraud detection with the IEEE-CIS dataset. Also, our implementation using Spark has good scalability because the increase in the dataset size did not bring the same degree of increase in running time. Thus, it is reasonable to assume that our implementation can handle real world fraud detection problems with larger datasets. In the process of using Spark, keeping the balance of Spark partition is very important. Lack of partitions can cause the waste of available cores and too many partitions lead to the excessive overhead problem. Both of them can cause a substantial increase in running time. The best parallelization strategy for this implementation is studied and compared. For future work, we plan to use other algorithms such as Xgboost and Lightgbm to further increase the model performance. Additionally, the use of larger datasets and creating new features may improve the accuracy of predictions. Currently, the dataset we used was collected 2 years ago. Newer credit card fraud detection datasets with data from in the past two years could be used to test our models and update our models based on the testing result to produce a solution more suitable for real world deployment.

Appendix

TABLE II
Transaction Table

Columns	Description	Type
TransactionID	ID of transaction	ID
isFraud	binary target	categorical
TranscationDT	transaction time	time
TranscationAmt	transaction payment amount	numerical
ProductCD	product code	categorical
card1-card6	payment card information	categorical
addr1-addr2	address	categorical
dist1-dist2	country distance	numerical
P_emaildomain	purchaser email domain	categorical
R_emaildomain	recipient email domain	categorical
C1-C14	counting (actual meaning is masked)	numerical
D1-D15	time delta (actual meaning is masked)	numerical
M1-M9	match (actual meaning is masked)	categorical
V1-V339	Vesta features (actual meaning is masked)	numerical

TABLE III
Transaction Table

Columns	Description	Type
TransactionID	ID of transaction	ID
id_01-id_11	identification data	numerical
Id_12-id_38	identification data	categorical
DeviceType	device type	categorical
DeviceInfo	device information	categorical

References

[1] Global payments report preview: your definitive guide to the world of online payments. 2015.
[2] Ge D, Gu J, Chang S, et al. Credit Card Fraud Detection Using Lightgbm Model[C]//2020 International Conference on E-Commerce and Internet Technology (ECIT). IEEE, 2020: 232-236.
[3] ”Payments companies are trying to fix the massive credit-card fraud problem with these 5 new security protocols”, Business Insider, 2015. [Online]. Available: http://www.businessinsider.com/how-payment-companies-are-trying-to-close-the-massive-hole-in-credit-card-security-2015-3. [Accessed: 02- May- 2021].
[4] Wire B. Global card fraud losses reach $16.31 Billion—will exceed $35 Billion in 2020 according to the Nilson report[J]. Business Wire, 2015, 8.

[5] Yee O S, Sagadevan S, Malim N H A H. Credit card fraud detection using machine learning as data mining technique[J]. Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 2018, 10(1-4): 23-27.

[6] S. Madden, ”From Databases to Big Data”, IEEE Internet Computing, 2012, vol. 16, no. 3, pp. 4-6, doi: 10.1109/mic.2012.50.

[7] M. Chen, S. Mao and Y. Liu, ”Big Data: A Survey”, Mobile Networks and Applications, 2014, vol. 19, no. 2, pp. 171-209, doi: 10.1007/s11036-013-0489-0.

[8] S. Sagiroglu and D. Sinanc, ”Big data: A review”, 2013 International Conference on Collaboration Technologies and Systems (CTS), 2013, pp. 42-47, doi: 10.1109/CTS.2013.6567202.

[9] J. Dean and S. Ghemawat, ”MapReduce: simplified data processing on large clusters”, Commun. ACM 51, 1, 2008, pp. 107–113, doi: 10.1145/1327452.1327492.

[10] M. Zaharia et al., ”Apache Spark: a unified engine for big data processing”, Communications of the ACM, 2016, vol. 59, no. 11, pp. 56-65, doi: 10.1145/2934664.

[11] A. Thusoo et al., ”Hive: a warehousing solution over a map-reduce framework”, Proceedings of the VLDB Endowment, 2009, vol. 2, no. 2, pp. 1626-1629, doi: 10.14778/1687553.1687609.

[12] S. Salloum, R. Dautov, X. Chen, P. Peng and J. Huang, ”Big data analytics on Apache Spark”, International Journal of Data Science and Analytics, 2016, vol. 1, no. 3-4, pp. 145-164, doi: 10.1007/s41060-0160027-9.

[13] M. Armbrust et al., ”Spark SQL: Relational Data Processing in Spark”, Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, 2015, pp. 1383–1394. doi: 10.1145/2723372.2742797.

[14] J. Shi et al., ”Clash of the titans”, Proceedings of the VLDB Endowment, 2015, vol. 8, no. 13, pp. 2110-2121, doi: 10.14778/2831360.2831365

[15] A. Mostafaeipour, A. Jahangard Rafsanjani, M. Ahmadi and J. Arockia Dhanraj, ”Investigating the performance of Hadoop and Spark platforms on machine learning algorithms”, The Journal of Supercomputing, 2020, vol. 77, no. 2, pp. 1273-1300, 2020, doi: 10.1007/s11227-020-03328-5.

[16] S. Gopalani and R. Arora, ”Comparing Apache Spark and Map Reduce with Performance Analysis using K-Means”, International Journal of Computer Applications, 2015, vol. 113, no. 1, pp. 8-11, doi: 10.5120/19788-0531.

[17] X. Meng et al. ”Mllib: Machine learning in apache spark.” The Journal of Machine Learning Research, 2016, 17.1, pp. 1235-1241.

[18] M. Belouch, S. El Hadaj and M. Idhammad, ”Performance evaluation of intrusion detection based on machine learning using Apache Spark”, Procedia Computer Science, 2018, vol. 127, pp. 1-6, doi: 10.1016/j.procs.2018.01.091.

[19] N. Nodarakis, S. Sioutas, A.K. Tsakalidis, & G. Tzimas, ”Large Scale Sentiment Analysis on Twitter with Spark”, 2016, EDBT/ICDT Workshops, pp. 1-8.

[20] S. Panigrahi, R. Lenka and A. Stitipragyan, ”A Hybrid Distributed Collaborative Filtering Recommender Engine Using Apache Spark”, Procedia Computer Science, 2016, vol. 83, pp. 1000-1006, doi: 10.1016/j.procs.2016.04.214.

[21] Z. Zojaji, R. E. Atani, and A. H. Monadjemi, “A survey of credit card fraud detection techniques: Data and technique oriented perspective,” arXiv preprint arXiv:1611.06439, 2016.

[22] D. Excell, “Bayesian inference–the future of online fraud protection,” Computer Fraud & Security, vol. 2012, no. 2, pp. 8-11, 2012.

[23] M. Zareapoor, and P. Shamsolmoali, “Application of credit card fraud detection: Based on bagging ensemble classifier,” Procedia computer science, vol. 48, no. 2015, pp. 679-685, 2015.

[24] A. Roy, J. Sun, R. Mahoney, L. Alonzi, S. Adams, and P. Beling, ”Deep learning detecting fraud in credit card transactions.” pp. 129-134.

[25] Kaggle. ”IEEE-CIS Fraud Detection Can you detect fraud from customer transactions?” 2 May, 2021; https://www.kaggle.com/c/ieee-fraud-detection/data.

[26] K. Yakovlev. ”Very short summary,” 2 May, 2021; https://www.kaggle.com/c/ieee-fraud-detection/discussion/111257.

[27] Y. Chen, and X. Han, ”CatBoost for Fraud Detection in Financial Transactions.” pp. 176-179.

[28] D. Ge, J. Gu, S. Chang, and J. Cai, ”Credit Card Fraud Detection Using Lightgbm Model.” pp. 232-236.

[29] Y. Zhang, J. Tong, Z. Wang, and F. Gao, ”Customer Transaction Fraud Detection Using Xgboost Model.” pp. 554-558.

[30] X. Yu, X. Li, Y. Dong, and R. Zheng, ”A Deep Neural Network Algorithm for Detecting Credit Card Fraud.” pp. 181-183.

[31] H. Najadat, O. Altiti, A. A. Aqouleh and M. Younes, ”Credit Card Fraud Detection Based on Machine and Deep Learning,” 2020 11th International Conference on Information and Communication Systems (ICICS), 2020, pp. 204-208, doi: 10.1109/ICICS49469.2020.239524.

[32] W. Deng, Z. Huang, J. Zhang and J. Xu, ”A Data Mining Based System For Transaction Fraud Detection,” 2021 IEEE International Conference on Consumer Electronics and Computer Engineering (ICCECE), 2021, pp. 542-545, doi: 10.1109/ICCECE51280.2021.9342376.

[33] Ian T. Jolliffe and Jorge Cadima (2016) Principal component analysis: a review and recent developments, Available at: https://doi.org/10.1098/rsta.2015.0202 (Accessed: 26 - April -2021).

[34] R. Lippmann, ”An introduction to computing with neural nets,” in IEEE ASSP Magazine, vol. 4, no. 2, pp. 4-22, Apr 1987, doi: 10.1109/MASSP.1987.1165576.

[35] N. Nodarakis, S. Sioutas, A.K. Tsakalidis, & G. Tzimas, ”Large Scale Sentiment Analysis on Twitter with Spark”, 2016, EDBT/ICDT Workshops, pp. 1-8.

[36] ”Hardware Provisioning - Spark 3.1.1 Documentation”, Spark.apache.org. [Online]. Available: https://spark.apache.org/docs/3.1.1/hardware-provisioning.html$\#$cpu-cores. [Accessed: 03- May-2021].

[37] ”Testing the Significance of the Correlation Coefficient — Introduction to Statistics”, Courses.lumenlearning.com. [Online]. Available: https://courses.lumenlearning.com/introstats1/chapter/testing-the-significance-of-the-correlation-coefficient/. [Accessed: 04-May- 2021].

[38] ”Testing the Significance of the Correlation Coefficient — Introduction to Statistics”, Courses.lumenlearning.com. [Online]. Available: https://courses.lumenlearning.com/introstats1/chapter/testing-the-significance-of-the-correlation-coefficient/. [Accessed: 04-May- 2021].

[39] ”Configuration-Spark 3.1.1 Documentation”, Spark.apache.org. [Online]. Available: https://spark.apache.org/docs/3.1.1/configuration.html$\#$memory-management. [Accessed: 04- May- 2021].