Credit Card Fraud Detection using Big Data Analytics: Use of PSOAANN based One-Class Classification

Sk. Kamaruddin
Centre of Excellence in Analytics, Institute for Development and Research in Banking Technology, Castle Hills Road No. 1, Masab Tank, Hyderabad-500057, India.

SCIS, University of Hyderabad, Hyderabad-500046, India.
+91 7842480180
skkamaruddin@gmail.com

Vadlamani Ravi*
Centre of Excellence in Analytics, Institute for Development and Research in Banking Technology, Castle Hills Road No. 1, Masab Tank, Hyderabad-500057, India.
+91 9440803818
padmarav@gmail.com
*Corresponding Author

ABSTRACT
Banking and financial industries are facing severe challenges in the form of fraudulent transactions. Credit card fraud is one example of them. In order to detect credit card fraud, we employed one-class classification approach in big data paradigm. We implemented a hybrid architecture of Particle Swarm Optimization and Auto-Associative Neural Network for one-class classification in Spark computational framework. In this paper, we implemented parallelization of the auto-associative neural network in the hybrid architecture.

CCS Concepts
• Computing methodologies→Semi-supervised learning settings • Mathematics of computing→Bio-inspired optimization • Information systems→Clustering and classification.

Keywords
Auto-associative neural network; Auto-encoder; One-class classification; Particle swarm optimization; Single class classification.

1. INTRODUCTION
A credit card is a payment card provided by every bank to eligible customers (cardholders) to make day-to-day transactions. Using the card, a cardholder can pay for goods and services without having money in their account at the particular moment and can be paid back to banks later point in time. The legitimate transactions made by the cardholder provide a pattern of his/her expenditures. If a card is stolen or accessed by some fraudsters, the transactions show an abnormal expenditure pattern and such transaction is called a fraudulent transaction. But, compared to large voluminous legitimate transactions, these types of transactions are relatively rare. Therefore, identification of such fraudulent transactions is a quite complex task, and it is a part of fraud analytics. Due to the complexity involved in fraud analytics, identification of fraudulent transactions always has been an interesting research problem for banking and financial industries, research communities and academia. The fraud analytics can be achieved using different data mining tasks like classification, outlier detection, etc.

ICIA-16, August 25-26, 2016, Pondicherry, India

DOI: http://dx.doi.org/10.1145/2980258.2980319

Classification can be performed in various ways viz. binary, multi-, and One-Class Classification (OCC). Binary classification is the process to classify a set of samples into two classes. Similarly, the multi-class classification is used to classify a sample into three or more classes. In the case of OCC, we have sufficient amount of samples available for one class, whereas samples for other classes will be rare. In this case, some rare samples do not belong to any of the known classes. Some examples of rare events area failure of a nuclear plant, credit card fraud, network intrusion, etc. Hence, whenever we came across the normal /regular samples in abundance and the targeted event in scarce, we can employ one-class classification approach to detect the rare event.

In this study, we employed OCC for credit card fraud detection in Big Data framework. Big Data can be attributed using 4 V's viz. Volume, Velocity, Variety, and Veracity [20]. Volume refers to a huge amount of data. Velocity implies how fast data are generated. Variety attributed to various formats of data. Finally, veracity represents the accuracy of data. In the case of credit card transactions, every bank deals with huge amount of credit card transactions every hour. Here, huge amount refers to volume and number of transactions per hour implies to velocity. So, credit card transactions can be attributed to two V's of Big Data i.e. volume and velocity. For our study, we relied upon one-class classification. Since there is a scarcity of fraudulent credit card transactions and a binary classification approach needs sufficient amount of historical data for both classes like legitimate and fraudulent, we extended one-class classification model proposed by Paramjeet et al. [16] in big data environment using Apache Spark framework for credit card fraud detection. Henceforth, we referred "Apache Spark" with Spark only.

In this paper, we developed parallelization of a hybrid architecture involving Particle Swarm Optimization (PSO) and Auto-Associative Neural Network (AANN) which is referred to as PSOAANN architecture. The PSOAANN architecture was proposed by Paramjeet et al. [16]. We implemented the AANN in a parallel manner over a Spark standalone cluster for one-class classification. The weight updation steps using PSO is implemented in a serial manner.

The remaining part of the paper is organized as follows: Literature review is presented in Section 2. Section 3 elaborates the proposed methodology. The dataset details are given in section 4. The results and discussion are presented in Section 5. In Section 6, the study is concluded with some future directions of the work.

2. LITERATURE REVIEW

In this section, we reviewed study related to one-class classification, auto-encoders for OCC, and spark for machine learning.

2.1 One-class Classification

Ravi and Singh [18] performed OCC using Auto-Associative Extreme Learning Factory (AAELF). They employed AAELF for bankruptcy prediction, credit risk prediction, and phishing detection. Tax and Duin [23] proposed Support Vector Data Description (SVDD). They separated the target class (negative data objects) from the outliers (positive class data object) by creating a hypersphere around the target data with a minimal volume. Strackeljan et al. [22] demonstrated a fault detection system using SVDD and a feature selection method for OCC. Another approach used for OCC was One-class Support Vector Machine (OC-SVM). Mourão-Miranda et al. [13] used OC-SVM for patient classification.

2.2 Auto-encoders for OCC

As far as one-class classifier using auto-encoder or auto-associative neural network is concerned, Pandey and Ravi [14] have proposed a model using PSOAANN for phishing detection in emails. Ravi et al. [17] presented classifying capability of PSOAANN in bankruptcy prediction using different bank datasets.

2.3 Spark for Machine Learning

Spark is a computational framework which provides iterative and interactive computation in a clustered environment using grand scale distributed setting. The native language for Spark is Scala [19], a high-level language for the JVM, and provides a functional programming interface. Spark is very efficient for machine learning. There are different APIs in Spark for machine learning. Meng et al. [12] have presented an open-source distributed machine learning library, MLLib. MLLib offers a high-level API that takes advantage of Spark's powerful environment. Sparks et al. [21] proposed MLI, an API for distributed machine learning.Kraska et al. [9] developed MLbase, a distributed machine learning system. MLbase provides a simple way to specify ML tasks. It dynamically selects a learning algorithm, which helps an ML researcher to work with ease. MLbase is quite optimized for data access.

Spark has been successfully used in a variety of applications of ML. Bharill et al. [3] proposed a clustering algorithm called Scalable Random Sampling with Iterative Optimization Fuzzy c-Means algorithm(SRSIO-FCM) with Spark to handle the challenges associated with big data clustering. Panigrahi et al. [15] have proposed a Hybrid Distributed Collaborative Filtering Recommender Engine (HDCFRE) using Spark. They experimented with MovieLens dataset. The parallel implementation of HDCFRE using Spark outperformed other traditional collaborative filtering algorithms. McNeil et al. [11] implemented Scalable Real-time Anomalies Detection and Notification of Targeted Malware in Mobile Devices (SCREDENT) using Spark. The authors developed a system to detect the mobile malwares in real-time. Bello-Orgaz et al. [2] surveyed the different tools for machine learning for social media data on a big data paradigm. They used Spark for massive data processing for the efficient utilization of ML algorithms in different domains.

3. PROPOSED METHODOLOGY

3.1 Particle Swarm Optimization (PSO)

PSO, developed by Eberhartand Kennedy [7] is a population-based optimization algorithm. It is a bio-inspired algorithm which mimics the behavior of a flock of birds or a school of fish in the search of food. PSO implementation is easy due to the tweaking of a few parameters. PSO generates solution by sharing knowledge mutually among all the particles present in the population to achieve the goal.

The process of PSO involves two basic steps. First, an update of the velocity of a particle. Second, the update of the position with the help of updated velocity of the particle. The two equations are given below:

$$V_d^{i+1} = w * V_d^i + c_1 * r_1^i * \left(P_d^i - x_d^i\right) + c_2 * r_2^i * \left(P_d^g - x_d^i\right) \quad (1)$$

$$x_d^{i+1} = x_d^i + V_d^{i+1} \quad (2)$$

where V_d^i is the velocity of the particle at the instant of i[th] iteration i.e. old velocity, V_d^{i+1} is the velocity of the particle at the instant of (i+1)[th] iteration i.e. new velocity, $P_d^i$is the best position travelled through a particle,P_d^g is the best position travelled through all the particles, x_d^i and $x_d^{i+1}$are the positions of a particle at the instant of i^{th} iteration and $(i+1)^{th}$ iteration respectively. The subscript$d$is the d^{th} dimension of the data object. The variable w is the inertia weight value, c_1 and $c_2$are two predefined positive constants, r_1 and r_2 are random numbers generated through uniform distribution U(0, 1).

3.2 Auto-Associative Neural Network (AANN)

AANN, developed by Kramer [8], is a variant of neural network where the number of nodes in the input layer is same as the output layer. Thus, it is named as auto-associative. The proposed model of Kramer contains three hidden layers viz. mapping layer, bottle-neck layer, and de-mapping layer. The AANN is a supervised model where the inputs are compared against the outputs of the model. Thereupon the error between inputs and outputs were optimized.

3.3 Architecture of PSOAANN

Paramjeet et al. [16] proposed PSOAANN, which comprises three layers, i.e. input, hidden, and an output layer. The input and output layers contain an equal number of nodes that is to represent the same variables in the input as well as the output layer. The number of hidden nodes is specified by the user. Each input node of the input layer is connected to all the nodes of the hidden layer. Similarly, each hidden node, in turn, is connected to all nodes of the output layer. The Sigmoid activation function is used in the hidden and output layers [16].

At the end of the training phase, the nodes in the output layer contain the perturbed values of the original input variables. In other words, the PSOAANN is non-linearly transforming the

original set of input variables into a set of perturbed values. There are well-known drawbacks in back propagation algorithm such as slow convergence and entrapment in local minima. To overcome these drawbacks, we have used a swarm intelligence technique, PSO which is an evolutionary approach for weight update, instead of the back-propagation.

We presented the architecture of PSOAANN in Figure 1. Figure 1 depicts the training and testing modules of PSOAANN. We considered mean squared error (MSE) as the error function. The advantages of using a three layered AANN for one-class classification over the five layered architecture of AANN [8] are decreasing computation time and the associated complexity.

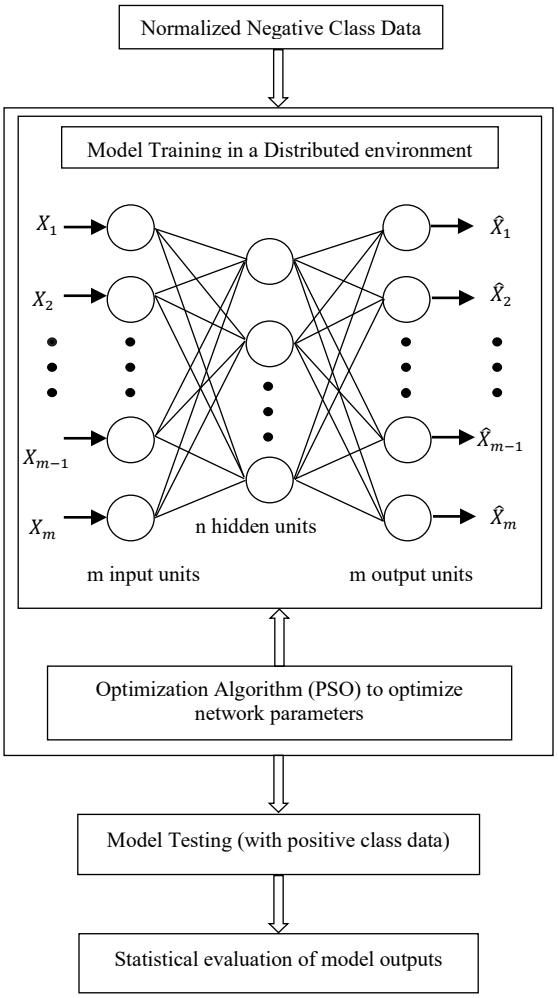

Figure 1: Architecture of hybrid model of PSOAANN

The algorithm for PSOAANN is presented below.

1. *Training phase of PSOAANN algorithm*: Specify the required number of hidden nodes. Initialize randomly the weight values between the input and the hidden layers (W) and also between the hidden and the output layers (W') using uniform distribution in the range (–5, 5). The PSOAANN model is trained with negative class samples (i.e. legitimate credit card transactions) only. The input nodes take the normalized input which is calculated as below. The output nodes contain the input variables as the target variables thereby bringing in the auto association concept.

$$x_{dk}^{normalised} = \frac{x_{dmax} - x_{dk}}{x_{dmax} - x_{dmin}} \quad (3)$$

where x_{dmax}, x_{dmin} are the maximum and minimum value of the dimension *'d'* in the input data object and x_{dk} is the input value of dimension *'d'* of k[th] input object.

2. Compute $\hat{x_{dk}}$ as follows: x_{dk} is the actual input and $\hat{x_{dk}}$ is the predicted input. Let m be the number of nodes in the both input and output layer; n be the number of nodes in the hidden layer. The predicted output is calculated as follows:

Computation at Hidden layer for each hidden node:

The input value is multiplied by the randomly initialized weight values W is given by (H_j):

$$H_j = \sum_{k=1}^{m} x_{dk} * w_{kj} \quad (4)$$

where j = 1 to n, for n hidden nodes; x_{dk} is the input value of dimension *'d'* of k[th] input object and w_{kj} is the weight from k[th] input node to j[th] hidden node.

Then sigmoid function is applied to H_j, given as below.

$$S(H_j) = \frac{1}{1+e^{-H_j}} \quad (5)$$

where H_j is given by (4).

Computation at Output layer for each output node:

The output of sigmoid function at hidden layer ($S(H_j)$) is multiplied by the randomly initialized weight values W' is given by (O_i):

$$O_i = \sum_{j=1}^{n} S(H_j) * w_{ji} \quad (6)$$

where i = 1 to m, for, m output (= input) nodes, $S(H_j)$ is given by (5) and w_{ji} is the weight from j[th] hidden node to i[th] output node.

Then sigmoid function is applied to O_i, given as below:

$$S(O_i) = \frac{1}{1+e^{-O_i}} \quad (7)$$

where O_i is given by (6).

As the sigmoid function results in a predictive value, $\hat{x_{dk}}$ in [0, 1] which is compared with the normalised input, x_{dk} that results into the Mean Square Error (MSE).

3. Compute error measure (MSE) E as follows:

$$E = \frac{1}{mn} \sum_{k=1}^{m} \sum_{d=1}^{n} (\hat{x_{dk}} - x_{dk})^2 \quad (8)$$

4. The MSE is optimized using PSO by updating all weights. The user-defined parameters in (1) are taken as, inertia weight = 0.9, $c_1 = c_2 = 2$.

5. Repeat Steps 2 to 4 until convergence is achieved or the max number of iterations is completed.

6. *The Testing phase of PSOAANN algorithm*: The PSOAANN model was trained on negative samples. The model learns the characteristics of samples belonging to majority class. The objective is to minimize the MSE. When the model was properly trained, it was tested on positive samples. The model will produce larger MSEs at the testing time compared to training phase. Pattern classification is achieved at the test phase by computation of relative error for each of the features present in the input and output data. In this case, the outputs

are approximations of inputs. If the relative error is having a larger value than a threshold value for all the input features, then a sample is considered to belong to the positive class. Here, the threshold value is specified by the user, and we have taken it as 0.05. Otherwise, the sample is identified to belong to the negative class.

$$\text{Relative Error } = \frac{|\hat{x_{dk}} - x_{dk}|}{x_{dk}} \quad (9)$$

The classification rate is computed using equation (10)

$$\text{Classification rate} = \frac{\text{No. of transactions classified as Fraud}}{\text{No. of total transactions in Positive Class}} * 100 \quad (10)$$

The training of PSOAANN falls under both unsupervised and supervised learning. It is unsupervised because we do not provide the class or output variable information to the AANN during training. The input variables are mapped onto themselves. It is supervised because the weights of AANN are updated by PSO where the fitness function of the particles viz., MSE is minimized. Therefore, PSO-based training algorithm provides supervised learning.

3.4 Hadoop MapReduce vs. Spark

Hadoop MapReduce (MR) [6] is difficult to program and needs abstraction, whereas Spark is easy to program and does not require any abstraction. The Spark has interactive mode whereas Hadoop MR does not possess interactive mode except Pig and Hive. Hadoop MR processes data in batch mode and produces reports for answering the queries on historical data. On the other hand, Spark can handle streaming data and allows us to modify the data in real time. Hadoop MR has more latency as the partial results are stored in the disk. In the case of Spark, it uses primary memory for caching partial results across the memory of distributed workers, which helps in faster execution. Zaharia et al. [26] claimed that Hadoop falls behind Spark by a factor of 10 in iterative machine learning workloads. A machine learning algorithm involves the iterative and interactive mode of execution. In this case, Hadoop falls behind in latency of execution time in comparison to Spark.

3.5 Spark and Data Parallelization

Spark is a computational framework for cluster computing environment involving distributed data processing. A Cluster is a group of machines connected to LAN and communicating through Secure Shell (SSH).It provides iterative, interactive, and scalable data processing. It provides high-level APIs in Java, Scala, Python and R and also interactively can be used from Scala, Python and R shells. Spark can run on a single machine as well as over several machines with existing cluster managers. Spark has the following options for deployment: (i) Amazon EC2, (ii) Standalone Deploy Mode, (iii) Apache Mesos, and (iv) Hadoop YARN. It can access diverse data sources including Hadoop Distributed File System (HDFS), Cassandra, HBase, and Amazon S3 (Simple Storage Service)[5].

Spark uses Resilient Distributed Datasets (RDDs), which is a distributed memory abstraction. RDDs allow in-memory computations on a distributed environment with fault-tolerance. RDDs present influential role in two types of applications viz. iterative algorithms and interactive approach. Other computing frameworks like Hadoop MR does not provide the same. Each RDD is having five pieces of information which can be accessed through a common interface. First, a set of partitions, which are atomic pieces of the dataset. Second, the preferred location for a partition, which is required for faster access due to data locality. Third, a set of dependencies on parent RDDs, which is needed for computation on RDDs. Fourth, an iterator, which is required for computation of elements in the partition with the help of iterators of parent RDDs. Finally, the fifth one is metadata about partitioning scheme and data placement [25]. The concept of RDD provides the parallelization of the algorithm through the partitions of data. Apart from providing in-memory storage, RDDs can also automatically recover from failures. Each RDD tracks the graph of transformations that was used to build it, called its lineage graph. These lineage graphs help in the reconstruction of any lost partitions through re-execution of operations on base data [24].

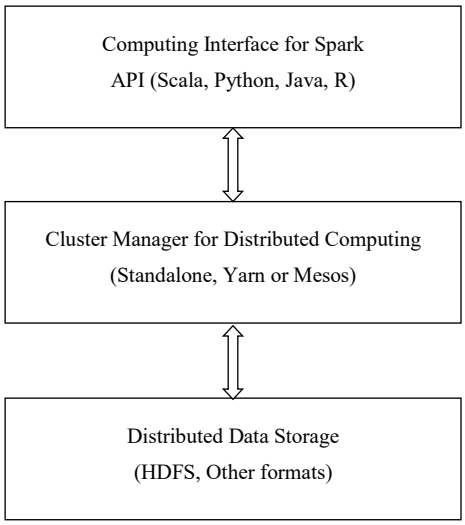

Figure 2: Components of Spark computational environment

Spark has an inherent feature of executing iterative programs and interactive mode of execution for user-friendly data analysis. Figure 2 depicts the components of Spark computational environment. The user interacts with the top layer through the computing interface. The top layer provides usage of different APIs for the Spark application. The Spark applications interact with cluster manager for accessing the data from the distributed data storage.

A Spark cluster has two main components: *master node* and *worker nodes*. Each machine in the cluster is known as a *node*. There is a single *master node* and more than one *worker nodes* in a cluster. The system which is the master node can also serve as a worker node. The *master node* allocates *jobs* to the *worker nodes*. A distributed file system e.g. HDFS, a cloud storage system e.g. S3 or a local file system is used for data storage. Spark can run on a single-node cluster setup having both *master node* and the *worker node* on the same machine. It also runs in a multi-node cluster setup. Spark applications run as independent sets of processes on a cluster. The Spark applications coordinate among themselves using a SparkContext object in the main program, which is also known as the *driver program*. For running spark on a cluster, the SparkContext connects to one of cluster managers. A Spark cluster can have several types of cluster managers (i.e. either Spark's standalone cluster manager, Mesos or YARN), which allocate resources across applications. When the SparkContext

is connected with the cluster manager, Spark acquires *executors* on *worker nodes* of the cluster. Here, an *executor* is a process that performs computations and storage operations

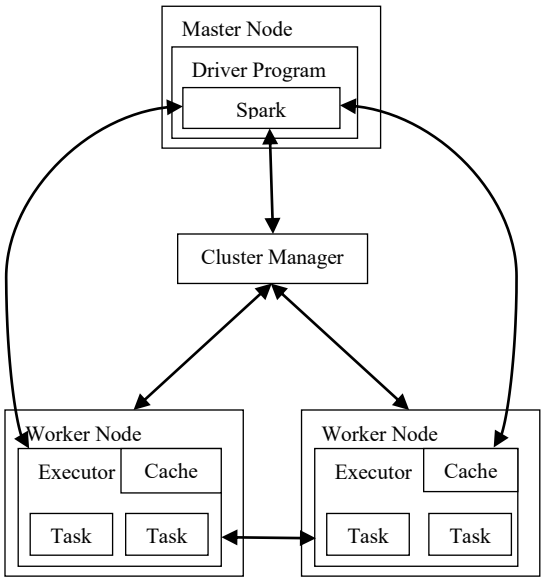

Figure 3: Spark Cluster Components

for an application. Next, it sends application code to the executors. Finally, SparkContext assigns *tasks* to the *executors* to run [5]. The cluster components are presented in Figure 3.

A *job* is a part of the code in the Spark application which takes inputs from HDFS/local file system, performs computations on them. A *job* writes outputs to an HDFS/local file system. Figure 4 depicts how a *job* is distributed on a Spark Cluster. A *driver process* runs on the master node and executes a *job* over the Spark Engine. Each *job* is split into a number of stages which can be either *map* or *reduce* stages. One *stage* may be dependent on the outcomes of a previous stage. So the stages are executed sequentially. Each *stage* comprises several *tasks* which can run in parallel. The data is split and spread over the cluster in several *partitions*. The computations of one stage are

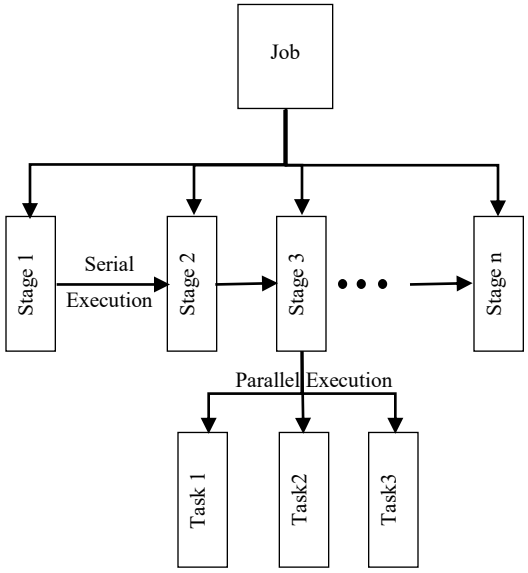

Figure 4: Execution of Job, Stage and Task in Apache Spark

performed on each partition in the form of a *task*. These tasks are executed as processes running in parallel over the worker nodes by the executor processes. Only one task is performed on one partition on each executor [3].

4. DATASET DESCRIPTION

The number of credit card transactions is growing day-by-day rapidly. This problem of credit card fraud detection can be considered as a big data problem because of the following reasons. This growth leads to a high Volume of data. The speed of credit card transactions results in a high Velocity of generation of credit card transaction data. In general, credit card transaction data is structured, but we can make a better detection of a fraudulent transaction if we include the profile information of the card holder and the transaction into the fraud detection model. The Inclusion of profile information incorporates the Variety feature of the Big Data. The portfolio data are not always complete which can play a significant role in the prediction. Veracity dimension refers to the biases, noise, and uncertainty in data. The credit card transactions dataset is highly biased one as a fraudulent transaction is rare occurrence. , thus, resulting in the veracity or uncertainty in the data.

We experimented with credit card fraud dataset i.e. "*ccFraud*" [4]. The *ccFraud* dataset has a high volume which cannot be processed on a single machine. Thus compelling to the use of distributed processing using Apache Spark. This dataset is a snapshot at a particular instant of time for processing, as there is non-availability of credit card dataset having real time inflow of transactions in the public domain. The *ccFraud* dataset is a highly unbalanced dataset with only 5.96% of fraudulent transactions, rendering the veracity in the data. Making the variety and veracity present by the inclusion of portfolio data is an identified problem area.

The *ccFraud* dataset contains ten million samples. We considered the genuine transactions as negative samples and fraudulent transactions as positive samples. The negative class has 9,403,986 samples. The positive class has 596,014 samples. The dataset contains 9 features with a total size of 291.7Mb. The different variables in the dataset are: (i) *custID* : customer ID, auto-incrementing integer value, (ii) *gender* : taking two values either 1 or 2 for male and female, (iii) *state* : state number given as integer, (iv) *cardholder*: number of cards per customer with a maximum value of 2, (v) *balance* : credit balance, (vi) *numTrans* : number of transactions made in integer, (vii) *numIntlTrans* : number of international transactions made in integer, (viii) *creditLine* : credit limit of a customer in integer, and (ix) *fraudRisk* : whether a given transaction is fraud or not. The "*fraudRisk*" is a binary feature having 1 and 0 as two discrete values. Here, 1 represents a fraudulent transaction and 0 is used for a non-fraudulent transaction. In the proposed model, 7 features have been considered to train the PSOAANN model. We discarded the "*custID*", since it contains unique values in all samples, which will disturb in the generalization of patterns. The class variable "*fraudRisk*" is also not provided at the time of training a PSOAANN. Since the model is to be trained with only one class i.e. negative samples.

5. EXPERIMENTAL SETUP AND RESULT & DISCUSSION

The experimental setup consists of standalone Spark cluster using the local file system as the storage system and Apache

Zeppelin as an editor. The Spark cluster comprises 9 *worker nodes* and a master node running the *driver program*. All the 10 nodes had same configuration i.e. Intel® Core™ i7-6700 CPU @ 3.40GHz with 8 logical cores. We allocated 24 GB of memory to worker nodes and 28GB of memory to the master node. Out of 8 logical cores, we assigned 6 logical cores to all ten nodes.

The model was executed in Spark 1.6.1 and Apache Zeppelin 0.5.6. The best execution time was achieved by tweaking the memory used by the executors in each worker node with the right number of data partitions. The data locality was achieved by using the local file system, thus improving the performance in execution time.

The current literature does not have any experimental result with credit card data set in the Apache Spark or MapReduce distributed environment. So our result could not be compared with any other result to confront.

The dataset is having 94.04% of legitimate or genuine credit card transactions and only 5.96% of fraudulent transactions. Hence the dataset is highly unbalanced, yielding to the complexity involved in the classification task. Without going for undersampling or oversampling of the biased data, we have conducted the experiment with one-class classification using legitimate transactions only and later tested the performance of the model by the fraudulent transactions.

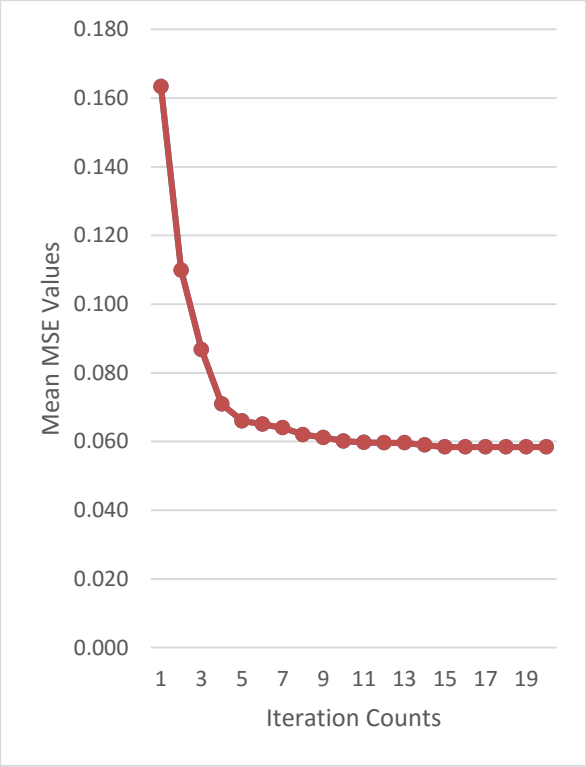

Figure 5: Mean MSE Convergence Plot

The training of the PSOAANN model was completed with 30 runs where each run is of 20 iterations. The computational time for each iteration is about 51sec on an average. The model was dependent on the evolutionary algorithm, PSO, which uses a random seed. Every run is dependent on the random seed, thus producing varying results for different runs. Hence, inorder to nullify the effect of randomness caused, we ran the model for 30 times. The execution time for each run is observed to be on average about 17m 05sec.

The AANN had three layered architectures with six nodes in the hidden layer. The weights between input and hidden layer as well as hidden and output layers are initialized using uniform distribution in the range of [-5, 5). The MSE calculation resulting from AANN was minimized by PSO. The objective function was to minimize the MSE between the actual input and predicted output. PSO was used to minimize the MSE. The convergence plot for mean MSE value over 30 runs versus 20 iterations is depicted in Figure 5.

We had conducted several experiments with different PSO parameters and found that the inertia weight w = 0.9 gives the best result. We also varied the values of c_1 and c_2 so that their summation equals 4 and with several experiments, it is found that $c_1 = c_2 = 2$ yields a better result. Incidentally, many research papers e.g. [1, 10] support the values of c_1 and c_2 to be 2 for producing a superior result. Bansal et al. [1] describe that for minimizing error, the best strategy for inertia weight is to take a constant value for it. In our case, it is found to be the value 0.9 which produces a better result.

The classification rate of the above runs was in the range of 85% to 95%. The statistical inferences are as follows. The minimum value of Classification Rate (CR): 85.89%, maximum value of CR: 95.312%, arithmetic mean of CR: 89.16%, median of CR: 88.645%, mode of CR: 87.16%, and standard deviation: 2.67%.

The median of CR being 88.64% indicates that for 50% of the runs, CR lies above 88.64%. The arithmetic mean lies on the right side of the median at 89.16% indicating that the data is right-skewed. The mode of CR at 87.15% indicates more chances to get 87% of classification rate. The SD of 2.67% clearly shows that the observations for CR are closely placed, and hence the algorithm yielded a stable result.

The proposed parallel approach uses Apache Spark for parallelization of datasets in a distributed clustered computation. In addition to that, we have implemented parallelization of the algorithm for the AANN. The implementation, involved parallelization of computations in the AANN in training as well as the test phase of the model (refer (3) to (9) in PSOAANN algorithm). Each instruction in AANN is carried out in a parallel manner over multiple worker nodes in the Spark cluster. The weight updation scheme in AANN using PSO is not parallelized. This aspect is left for future work. The whole model could not be constructed in a single system, as the dataset volume is large enough to fit in the memory of a single system. Hence, the speedup and efficiency measure could not be computed due to the lack of serial computation of the algorithm on a single machine.

6. CONCLUSIONS

In this paper, we employed a hybrid architecture involving particle swarm optimization (PSO) and auto-associative neural network (AANN) to get a solution for one-class classification (OCC) in big data paradigm in a SAPRK cluster. In this work, we parallelized of AANN and achieved an average of 89% true classification of the credit card fraud transactions. In future, we want to extend the work further with parallelization of PSO. Finally, we will compare the performance of our model with other machine learning tool like one-class support vector machine (OC-SVM).

7. REFERENCES

[1] Bansal, J. C., Singh, P. K., Saraswat, M., Verma, A., Jadon, S. S., and Abraham, A. (2011). Inertia weight strategies in particle swarm optimization. In *Nature and Biologically Inspired Computing (NaBIC)*, (Salamanca, Spain, October 19 - 21, 2011).IEEE NaBIC '11, 633-640. DOI=http://dx.doi.org/10.1109/NaBIC.2011.6089659.

[2] Bello-Orgaz, G., Jung, J. J., & Camacho, D. (2016). Social big data: Recent achievements and new challenges. Information Fusion. 28 (Mar. 2016), 45-59. DOI= http://dx.doi.org/10.1016/j.inffus.2015.08.005.

[3] Bharill, N., Tiwari, A., and Malviya, A. (2016). Fuzzy Based Clustering Algorithms to Handle Big Data with Implementation on Apache Spark. In Proceedings of the IEEE 2nd International Conference on Big Data Computing Service and Applications, (Oxford, UK, March 29 – April 01, 2016). IEEE BigDataService '16, 95-104. DOI= http://dx.doi.org/10.1109/BigDataService.2016.34.

[4] ccFraud Dataset: Apr. 2013. http://packages.revolutionanalytics.com/datasets/. Accessed: 2016- 06- 14.

[5] Cluster Mode Deployment Spark 1.6.1: Mar. 2016. http://spark.apache.org/docs/latest/cluster-overview.html. Accessed: 2016- 06- 14.

[6] Dean, J., andGhemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM. (Jan. 2008), 51, 1, 107-113. DOI= http://dx.doi.org/10.1145/1327452.1327492.

[7] Eberhart, R. C., and Kennedy, J. 1995. A new optimizer using particle swarm theory. In *Proceedings of the 6th International Symposium on Micro Machine and Human Science,* (Nagoya city, Aichi, Japan, October 04 - 06, 1995). MHS '95. 1, 39-43. DOI= http://dx.doi.org/10.1109/MHS.1995.494215.

[8] Kramer, M. A. 1991. Nonlinear principal component analysis using autoassociative neural networks. AIChE journal. 37, 2, (Feb. 1991), 233-243. DOI= http://dx.doi.org/10.1002/aic.690370209.

[9] Kraska, T., Talwalkar, A., Duchi, J. C., Griffith, R., Franklin, M. J., and Jordan, M. I. 2013. MLbase: A Distributed Machine-learning System. In *Proceedings of the 6th Biennial Conference on Innovative Data Systems Research,* (Asilomar, California, USA, January 6 – 9, 2013).CIDR '13, 1, 2-1.

[10] Maheshkumar, Y., Ravi, V., and Abraham, A. (2013). A particle swarm optimization-threshold accepting hybrid algorithm for unconstrained optimization. Neural Network World, 2013, 23, 3, 191 - 221.

[11] McNeil, P., Shetty, S., Guntu, D., & Barve, G. (2016). SCREDENT: Scalable Real-time Anomalies Detection and Notification of Targeted Malware in Mobile Devices. In *Proceedings of the2nd International Workshop on Mobile Cloud Computing systems, Management, and Security.* (Madrid, Spain, May 23-26, 2016). MCSMS '16. Procedia Computer Science, 83, 1219-1225. DOI= http://dx.doi.org/10.1016/j.procs.2016.04.254.

[12] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D.B., Amde, M., Owen, S., and Xin, D. 2015. Mllib: Machine learning in apache spark. *arXiv preprint arXiv:1505.06807*.

[13] Mourão-Miranda, J., Hardoon, D. R., Hahn, T., Marquand, A. F., Williams, S. C., Shawe-Taylor, J., and Brammer, M. (2011). Patient classification as an outlier detection problem: an application of the one-class support vector machine. NeuroImage. 58, 3, (Oct. 2011), 793-804. DOI= http://dx.doi.org/10.1016/j.neuroimage.2011.06.042.

[14] Pandey, M., and Ravi, V. 2013. Phishing Detection Using PSOAANN Based One-Class Classifier. In *Proceedings of the 6th International Conference on Emerging Trends in Engineering and Technology,* (Nagpur, Maharashtra, India, December 16 - 18, 2013). ICETET '13, 148-153. DOI= http://dx.doi.org/10.1109/ICETET.2013.46.

[15] Panigrahi, S., Lenka, R. K., & Stitipragyan, A. (2016). A Hybrid Distributed Collaborative Filtering Recommender Engine Using Apache Spark. In *Proceedings of the International Workshop on Big Data and Data Mining Challenges on IoT and Pervasive Systems,* (Madrid, Spain, May 23-26, 2016). BigD2M '16. Procedia Computer Science, 83, 1000-1006. DOI= http://dx.doi.org/10.1016/j.procs.2016.04.214.

[16] Paramjeet, Ravi, V., Naveen, N., and Rao, C. R. (2012). Privacy preserving data mining using particle swarm optimisation trained auto-associative neural network: an application to bankruptcy prediction in banks. International Journal of Data Mining, Modelling and Management, 4, 1, 39-56. DOI= http://dx.doi.org/10.1504/IJDMMM.2012.045135.

[17] Ravi, V., Nekuri, N., & Das, M. 2012. Particle swarm optimization trained auto associative neural networks used as single class classifier. In *Proceedings of the 3rd International Conference on Swarm, Evolutionary, and Memetic Computing,* (Bhubaneswar, Odisha, India, December 20 - 22, 2012). SEMCCO '12, 577-584. DOI= http://dx.doi.org/10.1007/978-3-642-35380-2_67.

[18] Ravi, V., and Singh, P. 2014. Auto-associative extreme learning factory as a single class classifier. In *Proceedings of the IEEE International Conference on Computational Intelligence and Computing Research,* (Coimbatore, Tamilnadu, India, December 18 – 20, 2014). ICCIC '14, 1-6. DOI= http://dx.doi.org/10.1109/ICCIC.2014.7238402.

[19] Scala programming language. http://www.scala-lang.org.

[20] Schroeck, M., Shockley, R., Smart, J., Romero-Morales, D., and Tufano, P. 2012. Analytics: The real-world use of big data: How innovative enterprises extract value from uncertain data, Executive Report, IBM Global Business Services, Business Analytics and Optimization. (New York, USA, October 2012). 1-22.

[21] Sparks, E. R., Talwalkar, A., Smith, V., Kottalam, J., Pan, X., Gonzalez, J., Franklin, M.J., Jordan, M.I., and Kraska, T. (2013). MLI: An API for distributed machine learning. In *Proceedings of the 13th IEEE International Conference on Data Mining,* (Dallas, Texas, USA, December 07 – 10, 2013). ICDM '13, 1187-1192. DOI= http://dx.doi.org/10.1109/ICDM.2013.158.

[22] Strackeljan, J., Goreczka, S., and Behr, D. 2012. A fault detection concept for single class problems.In *Proceedings of the 9th International Conference on Condition Monitoring and Machinery Failure Prevention Technologies,* (London, UK, June 12 – 14, 2012). CM '12 and MFPT '12, 675-682.

[23] Tax, D. M., and Duin, R. P.2002. Uniform object generation for optimizing one-class classifiers. The Journal of Machine Learning Research. 2, (Jan. 2002), 155-173.

[24] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., Mccauley, M., Franklin, M., Shenker, S. and Stoica, I. (2012). Fast and interactive analytics over Hadoop data with Spark. In USENIX; login. 37, 4, 45-51.

[25] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., and Stoica, I. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation,* (San Jose, CA, USA, April 25 - 27, 2012). NSDI '12, 2-2.

[26] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark: Cluster Computing with Working Sets. In *HotCloud*, 10, 10-10.