A Parallel Grid Optimization of SVM Hyperparameter for Big Data Classification using Spark Radoop

Ahmed Hussein Ali

Al-Iraqia University Baghdad, Iraq, msc.ahmed.h.ali@gmail.com

Mahmood Zaki Abdullah

Mustansiriyah University, Iraq, drmzaali@uomustansiriyah.edu.iq

Follow this and additional works at: <https://kijoms.uokerbala.edu.iq/home>

Part of the [Computer Sciences Commons](#)

Recommended Citation

Ali, Ahmed Hussein and Abdullah, Mahmood Zaki (2020) "A Parallel Grid Optimization of SVM Hyperparameter for Big Data Classification using Spark Radoop," *Karbala International Journal of Modern Science*: Vol. 6 : Iss. 1 , Article 3. Available at: <https://doi.org/10.33640/2405-609X.1270>

This Research Paper is brought to you for free and open access by Karbala International Journal of Modern Science. It has been accepted for inclusion in Karbala International Journal of Modern Science by an authorized editor of Karbala International Journal of Modern Science.

University of
Kerbala

A Parallel Grid Optimization of SVM Hyperparameter for Big Data Classification using Spark Radoop

Abstract

The big data phenomenon is currently a challenge to the process of relevant knowledge extraction using classical machine learning technique. This is due to the need for efficient data reduction and new fast-distributed machine learning algorithms for such process on big data. The extensive application of SVM demands efficient methods of constructing the classifier to be suitable for big data and high classification capability. In reality, the efficiency of SVM relies on the efficient derivation of the optimal feature subset and the algorithmic parameters. The grid search optimization method usually presents global optima and high learning accuracy compared to PSO and GA, but its larger computation takes much time. The grid search is more attractive because it can simultaneously take part in the learning of every SVM since they do not rely on each other. A novel parallel implementation of grid optimization using Spark Radoop is proposed in this paper to minimize the great computation load and make it suitable for big data processing issues. A major contribution of this study is a significant reduction in the algorithmic computational time when compared to the serial version of gridSVM, as well as the high classification accuracy compared to the other parallel optimization techniques.

Keywords

Big data Machine learning Parallel computing Support vector machine Optimization Grid search

Creative Commons License

This work is licensed under a [Creative Commons Attribution-Noncommercial-No Derivative Works 4.0 License](https://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction

The need for fast machine learning (ML) and computational intelligence methods has increased due to the exponential increase in data generation and the complexity of computing systems [1,2]. Most of the existing computational intelligence methods have many learning problems such as persistent human intervention and convergence time. Most organizations rely on data to survive [3]; as such, the level of data generation has increased in several folds compared to the level over a few years ago. However, the major task is how to extract valuable information from this data [4] and this is the reason big data-related technologies were introduced. This has made big data a trending technology recently [5] as most of the active open-source research projects are big data-related, and the number of such research projects is rapidly increasing. Big data is popularly defined as data characterized by 3Vs [6], representing volume, velocity, and variety [7]. Volume with respect to big data has just been discussed. Regarding Velocity, it relates to the fast rate of data generation. Variety refers to the various way data can exist—semi-structured, multi-structured, or unstructured. Meanwhile, the use of efficient learning frameworks provides noticeable benefits such as ease of implementation, fast learning speed, and reduced human input. The implementation of fast and reliable ML methods in big data is facing several challenges. The past decade witnessed the application of ML techniques in several complex data-intensive fields, such as astronomy, biology, medicine, etc. These applications were intended to provide efficient solutions to information mining from big data. With the emergence of big data, data collection becomes complicated and difficult to handle using ordinary learning techniques since the established learning process from traditional data sets is not applicable to high data volumes. For example, most of the conventional ML algorithms are developed for data that will be wholly loaded into memory, but this is no longer applicable in the big data context. Despite the fact that significant advancements can be achieved by learning from these huge data, it is also associated with numerous challenges. There are three general subdomains of ML, these are supervised, unsupervised, and reinforcement learning. Regarding supervised learning [8,9], labeled data (with the inputs and desired outputs) are required during the training phase, but for unsupervised learning

[10], there is no need for labeled training data as the inputs and desired outputs are provided by the environment. For reinforcement learning [11], it permits learning from the response received via communication with the outside environment. Despite the recent advancements in the ML field, there is still much need to address most of the significant problems associated with big data.

The classification of the issues related to ML techniques [12] with respect to big data is presented from 5 different perspectives in this section (refer to Fig. 1). The 5 perspectives are learning for large-scale data, learning for high-speed data streaming, learning for various types of data, learning for incomplete and uncertain data, as well as learning for valuable information extraction from huge data volumes. Data volume is obviously the basic feature of big data which comes with several ML challenges.

The SVM, which was first introduced by Vapnik [13] has served as a reference for most classification tasks owing to its computational efficiency, flexibility, and ability to handle highly-dimensional datasets. The SVM provides promising results but despite that, the construction of an efficient SVM classifier which can predict unseen new samples accurately is still a challenge. The generalization capability of SVM mainly depends on parameter optimization and feature selection [14]. The classification capability of a trained SVM cannot only be improved by a proper parameter setting but can also ensure an efficient classification of unknown new samples. In the SVM, the parameters which must be tuned are the kernel function parameter and the error penalty parameter. The choice of parameter largely influences the performance of SVM; hence, parameters selection has become an important study area in the study of SVMs. The main challenge of the classification with big data is how to build an efficient parallel classification algorithm that selects the best hyper-parameters and feature subset to learning fast and high accuracy. The work aims to develop an effective way of constructing SVM classifier with parallel hyper-parameter optimization and feature subset selection so that the parallel classification method can be applied in various practical aspects and provide promising outcomes. A novel parallel implementation of grid optimization using Spark Radoop is proposed in this paper to minimize the great computation load and make it suitable for big data processing issues. A major contribution of this study is

Fig. 1. The critical issues of machine learning for big data.

a significant reduction in the algorithmic computational time when compared to the serial version of gridSVM, as well as the high classification accuracy compared to the other parallel optimization techniques. The rest of this work is presented in the following manner:

The related literature survey was presented in section 2 while the definition of the problem solved in this study was presented in section 3, section 4 introduced SVM while section 5 presented the Parallel Grid Search Method using Spark Radoop platform. In section 6, the proposed method was presented while section 7 presented the experimental setup. The datasets used in this study were detailed in section 8 while section 9 presented the discussion of the achieved results. The last section (10) presented the conclusions derived from the study, as well as the directions for future studies.

2. Literature survey

Several studies have proposed several optimization techniques which can be categorized into several categories based on the classification process. There are several optimization approaches proposed by the research community. Among the existing efforts on optimization hyper-parameters and feature. In this study, the literature review covered works that are related to the parallel implementation of various SVM classification frameworks. Zhi et al. [15] proposed a parallel genetic algorithm in order to select best features and parameters for SVM classifier. This method was implemented in the MATLAB development environment. Experiment results obtained from several real world datasets of UCI database showed promising performance in terms of 10-fold accuracy, the size of

selected feature subset, the number of support vectors, and training time. They revealed that the proposed approach efficiently optimized SVMs' model parameters, but not correctly obtained the discriminating feature subset. Tatjana and Bruno [16] proposed a parallel tuning scheme which selects learning parameters for unbalanced datasets using original SVM classifier. The parallel method was implemented in Asynchronous parallel pattern search (APPSPACKv3). But the method not integrating different kernels into a single SVM model and a hierarchical parallelization combining parallel SVM training with parallel parameter optimization to speed up the model selection. Neng Hou et al. [17] proposed a parallel genetic algorithm in order to enhance the process of dispersion correction that was implemented on open source message passing interface OpenMPI. the paper presents a CPU-GPU parallel genetic algorithm with dispersion correction for HW/SW partitioning. The contributions of this work are as follows. Firstly, an enhanced genetic algorithm with dispersion correction is presented. The under-constraint individuals are marched to feasible region with dispersion correction step by step. Secondly, the individuals processing including costs computation and dispersion correction are run in parallel. For a given problem size, the overall running time can be reduced while keeping the diversity of genetic algorithm. Lacrimioara et al. [18] presented a grid search method to SVMs' kernel parameters optimization. The method addressed the problem of multiclass classification with imbalanced datasets. The author presented a step-by-step grid search approach in log2-space to optimize the kernels' parameters for SVMs. The considered kernels were: linear, radial basis, and sigmoid. We have shown that the parameters optimization improves the recognition

performance for audio classification, especially when using the sigmoid kernel. All the above-discussed algorithms are not suitable for a high-dimensional large-sized dataset classification because the process of obtaining the best hyper-parameter of the classifier takes much time. The PSO and GA sometimes fall into local optima, so, grid search always finds global optima, but its process is time-consuming. The grid search is desirable as it simultaneously takes part in the training of any SVM since they do not depend on each other.

3. Problem definition

The existing literature has suggested several classification approaches. SVM is one of the effective classifiers that minimize the bounds on the generalization error rate of a learning process by maximizing the class separation margins. The optimal selection of the optimal feature subset and parameter of SVM ensures its efficient performance. All the above-discussed algorithms in the literature have a problem of local and global optima during parameter optimization, especially with large-scale data sets. The paper addressed this problem by proposing a parallel grid optimization of SVM hyper-parameter for big data classification issues using Spark Radoop.

4. Support vector machine (SVM)

The SVM is a supervised ML method for pattern recognition which was first introduced by Vapnik [14]. It is one of the attractive ML techniques due to its remarkable results. The SVM is based on the Structural Risk Minimization principle as it constructs an optimal demarcating hyperplane which can split 2 or more classes. During the building process, the SVM strives

to maximize the splitting of the two types of data. Fig. 2 depicts the edge of 2 large demarcated classes of data. This move minimizes the expected generalization error.

Despite the successful application of SVM in many fields, its practical application still faces a serious problem which bothered on the selection of some of its parameters to enhance its performance efficiency among the parameters that need to be carefully selected include the penalty constant C, the kernel function parameters, and the loss function parameter. The kernel function of SVM mainly controls its prediction capability and efficiency. Some of the common kernel functions include:

Linear kernel function:

$$K(x_i, x_j) = x_i^T \cdot x_j \quad (1)$$

Polynomial kernel function:

$$K(x_i, x_j) = (x_i^T \cdot x_j + 1)^d \quad (2)$$

RBF kernel function:

$$K(x_i, x_j) = \exp(-\|x_i - x_j\|^2 / \sigma^2) \quad (3)$$

Sigmoid kernel function:

$$K(x_i, x_j) = \tanh[b \cdot (x_i^T \cdot x_j) + \theta] \quad (4)$$

The kernel function parameters portray the attributes of the training data and greatly influence the systems' generalization capability. Regarding C, it determines the balance between the training error minimization and the minimization of the complexity of the model. The range of the value of C can influence the generalization of SVM. The value of the loss function parameter γ reflects the expected error in the data classification process. It also affects the number of generated support vectors by the classifier, consequently affecting the classifiers' generalization error. A high value of γ will manifest in a high separating error and a small number of generated support vectors.

5. Parallel grid search

It is time-intensive to tune a non-trivial number of parameters; hence, the use of parallel computing resources may be ideal. Parallelism can be inserted into the SVM model at the selection stage in three different ways; these include parallelizing a single SVM training, parallelly training several SVMs, as well as parameter optimization using a parallel algorithm.

Fig. 2. SVM hyperplane.

5.1. Grid search

The principle of the standard grid search (GS) [19,20] method involves the splitting of a given range of C and γ within meshes and grid node values for the parameter combination. Assume that C has M numbers of parameter values and γ has N number of values; to combine the parameters, (C, γ) will have $M \times N$ numbers, each being the combination of training SVM. The best combination is achieved by comparing the error results. If the deviation from the selected is minimal, M and N will be large and will result to a combinatorial expansion. Being that each grid node must explore their operation, it will take much time and the selection of the step distance will equally affect the accuracy.

5.2. Apache Spark

In the big data paradigm, Apache Spark [21] is a publicly available distributed in-memory computational tool which uses a collection of cost-effective low-performance systems for data analytics. It offers more advantages compared to other big data analytical tools like Hadoop [22], Storm, and MapReduce [23]. In terms of speed, Spark is quicker than MapReduce [24] and ensures low latency because of the minimized disk input and output activities. Spark is built with in-memory computation capability [25], making data processing with Spark to be faster compared to the other MapReduce. The intermediate results are maintained in the memory in Spark compared to Hadoop where every intermediate output is written on disk. This ensures a reduction in job execution time and improves the speed of job execution. When the threshold of the storage memory has been exceeded, the excess data is transferred to the disk. Spark depends on data abstraction for data processing by using the Resilient Distributed Dataset (RDD) [26]. So, we can utilize Spark for executing parameter combination as a task in Spark engineTasks are not executed immediately in Spark, rather, a range of operations is maintained as the jobs' meta-data (referred to as DAG) owing to the transformation operation. Several applications are concurrently executed in Spark using independent cluster processes and resources. The driver comprises an object known as SparkContext which coordinates the applications. Applications are run on a cluster by connecting the SparkContext to the cluster manager which could either be YARN, mesos, or standalone. Resource allocation across the applications is the duty of the cluster manager; however, the

standalone cluster can also be responsible for managing a single application. Upon a successful connection of Spark to the cluster manager via the SparkContext, it will acquire resources and executors for the executors on the worker nodes in the cluster. These executors are the processes that execute the computational processes and data storage for the application. Tasks are sent to the executors by SparkContext for execution (Fig. 3).

This kind of data can be conveniently managed using Apache Hive (a data storage software constructed on Hadoop). With this software, large datasets stored in distributed storage can be read, written, and managed with SQL support. Additionally, it requires a command line tool and a database connection which must be established via the Java Database Connectivity (JDBC) to enable linkage to the Apache Hive software (hive.apache.org).

5.3. Radoop

Radoop is an extension which we have created to ensure the integration of Hadoop in RapidMiner. As a data science software platform, Radoop eliminates the difficulty of data preparation and ML on Hadoop and Spark (see Fig. 4).

Radoop offers the extra operators for RapidMiner and interacts with the Hadoop cluster for job execution. SparkRM provides the platform for parallel running of all operations and data processes in RapidMiner Studio within the Hadoop environment.

This is ensured using Apache Spark as the task execution tool, thereby expanding the use cases and enabling stronger algorithms compared to MLlib. Due to the highly optimized nature of some data analytics functions of Hive and Mahout, they were reused in this

Fig. 3. Apache Spark components.

Fig. 4. The overall architecture of the RapidMiner- Hadoop integration [27].

study. Fig. 2 presents the overall framework of the RapidMiner- Hadoop integration. This study designed an extension that will help to achieve a close integration and offer the same Hadoop functions commonly employed in memory-based RapidMiner processes. The first step of the Radoop creation process is the addition of the RadoopNest meta-operator which contains the general cluster settings, such as the IP address of the Hadoop master node. The remaining Radoop operators can be used only within this meta-operator.

6. The proposed methodology

The proposed parallel grid optimization of SVM hyper-parameter method (PGOSVM) was executed in Apache Spark Radoop computational framework with distributed data storage in HDFS. Hence, the execution of the PGOSVM algorithm followed a parallel distribution manner.

Step 1: Upload dataset D to HDFS for distributed storage. In this step, uploading the whole dataset into Hadoop distributed file system to solve the problem of hardware failure. So there are many components in the proposed method, each component has a probability of failure, HDFS has the ability to fast detect the failure and automatic recover it.

Step 2: Data preprocessing step (normalize the dataset using min-max normalization). This work normalize the dataset using either MIN-MAX or log 2 normalization techniques. The end-points of the range of the distance is specified by the quantities $D_{\max}$ and $D_{\min}$, and D_n is given by:

$$D_n = \frac{D - D_{\min}}{D_{\max} - D_{\min}} \quad (5)$$

where $D_{\min} = \min (d_1, d_2, \dots \dots d_m)$ and $D_{\max} = \max (d_1, d_2, \dots \dots d_m)$.

Step 3: transform the raw dataset into an RDD of LabelPoint to be suitable for Spark operations lines: `org.apache.Spark.rdd.RDD`

`[String] = MapPartitionsRDD.`

Step 4: Split data into training and test sets. In this step we used 10-fold cross validation to deciding whether the numerical results quantifying hypothesized relationship between variables. Cross validation available as an operator on radoop called “Performance Classification Radoop” to expect a test Hadoop example set.

Step 5: Extract the SVM hyperparameter by grid search operator in Radoop. The set of parameters to be tuned in Radoop SVM operator can optimized such as " show_convergence_plot, kernel_type, kernel_gamma, kernel_sigma1, kernel_sigma2, kernel_sigma3, kernel_degree, kernel_shift, kernel_b, C, max_evaluations, generations_without_improval, popution_size, inetia_weight, global_best_weight, use_local_random_seed local_random_seed.

Step 6: Divide the SVM hyperparameter into N combinations parts to make the parallel execution efficient. We divided the SVM hyperparameter according to the number of workers (Spark worker nodes) as follow: SVM hyperparameter/number of available workers.

Step 7: Each set of parameter combination forwarded to Spark worker node to train SVM with the training set.

Step 8: Obtain the accuracy result for each Spark worker node.

Step 9: Select the combination set of parameter values that yields the best accuracy results. The criterion is the highest accuracy obtained.

Step 10: Build a final model with the best parameters.

Fig. 5 depicts the workflow of the PGOSVM parallel tuning. The execution of the map-reduce operation in Spark was done via transformation and action operations while PGOSVM was implemented

Fig. 5. The workflow of PGOSVM parallel tuning system.

parallelly by mapping it the whole data partitions using a map operation:

```
val classRDD = partitioned data.mapPartitions
(SVM)
```

Then, all the sub-clusters generated in each partition were collected by the Reduce operation which was performed using a collect action operation:

```
val classRDD = combinationParaRDD.collect ()
```


The data represented by an RDD is fragmented and these fragments are generally distributed across a group of nodes. When Spark is executed on a single machine, it is mandatory that all the data fragments are available on such a machine. Fig. 6 showed the system architecture of the suggested method.

7. Experimental setup

In this study, the experimental setup comprised of a standalone Spark cluster which uses Apache Zeppelin 0.7.1 as an editor and an HDFS storage system. The components of the Spark cluster include

Fig. 6. The system architecture of the proposed method.

Table 1
System description.

Operating system	Windows 10
CPU	Intel® Core™ i7-6700 CPU @ 3.40 GHz with 8 logical cores
Memory	16 GB
Number of workers	3
Computational framework	Apache Spark 2.2.1
compatible framework	Radoop
Distributed storage system	HDFS (Hadoop 2.7.3)
Editor for code development	Apache Zeppelin 0.7.1
Language used for coding	Scala 2.11.8

a master node which runs the driver program and 3 worker nodes (which includes 1 worker node that runs on the master node). The 3 nodes have a similar configuration (running on Intel® Core™ i7-6700 CPU @ 3.40 GHz, 16 GB RAM, 8 logical cores, Windows 10 OS (refer to Table 1)). The 3 worker nodes were allocated memory of 48 GB. Each worker node was configured with 4 executors of 4 GB memory and 2 cores. In the master node, each worker was configured with 3 executors of 5 GB memory and 2 cores each. A memory of 16 GB was allocated to the driver process. The execution of the PGOSVM was performed using Scala 2.11.8 programming language in Spark 2.2.1 cluster with Hadoop 2.7.3 as distributed storage (refer to Table 1). The optimal execution time was obtained by altering the memory available to the executors in each worker node with the optimal number of data partitions.

In the Radoop, every process initiates by the addition of a RadoopNest operator as shown in Fig. 7. The general settings comprise of a project-defined Hive table prefix, the Spark master nodes' IP address, as well

Table 2
Datasets description.

Data	No of record	No of attributes	No of classes
Covtype	581012	54	7
Covtype-2	581012	54	2
Higgs	11,000,000	28	2

as the other port settings. The rest of the operators will run in the nest and communicate with the cluster using these general settings. Access to all the Radoop operators can be initiated from the RapidMiner's usual operator panel within the Radoop cluster. The aim of these integration attempts is to ensure that the Radoop user interface is as identical to the built-in RapidMiner functionalities as possible. The design process of the RadoopNest is similar to that of the core RapidMiner as there are similar operators which can be used on Hadoop.

8. Dataset description

This study employed big classification datasets sourced from the UCI data repository. The main features of these datasets are summarized in Table 2, showing the number of records, attributes, and classes for each data set.

The forest cover type is a classification problem which can only be predicted from cartographic variables (and not from remotely sensed data). For a given observation (30×30 m cell), the actual forest cover type can be determined from the USFS Region 2 RIS data which consists of 54 features and 7 classes (size = 11 Mb). Table 3 described the different variables in this dataset.

Fig. 7. Settings of the Radoop Spark connections.

Table 3
Details of Covtype dataset.

Feature Name	Data Type	Measurement	Description
Elevation	Q	meters	Elevation in meters
Aspect	Q	azimuth	Aspect in degrees azimuth
Slope	Q	degrees	Slope in degrees
Horizontal_distance_to_hydrology	Q	meters	Horz Dist to nearest surface water features
Vertical_distance_to_hydrology	Q	meters	Vert Dist to nearest surface water features
Horizontal_distance_to_hydrology	Q	meters	Horz Dist to nearest roadway
Hillshade_9am	Q	0 to 255 index	Hillshade index at 9am, summer solstice
Hillshade_Noon	Q	0 to 255 index	Hillshade index at noon, summer solstice
Hillshade_3pm	Q	0 to 255 index	Hillshade index at 3pm, summer solstice
Horizontal_distance_to_fire_points	Q	meters	Horz Dist to nearest wildfire ignition points
Wilderness_Area (4 binary columns)	Q	0 (absence) or 1 (presence)	Wilderness area designation
Soil_Type (40 binary columns)	Q	0 (absence) or 1 (presence)	Soil type designation
Cover_Type (7 types)	I	1 to 7	Forest Cover Type designation

Table 4
Details of HIGGS dataset.

Feature name	Feature details
lepton pT, lepton eta, lepton phi, missing energy magnitude, missing energy phi, jet 1 pt, jet 1 eta, jet 1 phi, jet 1 b-tag, jet 2 pt, jet 2 eta, jet 2 phi, jet 2 b-tag, jet 3 pt, jet 3 eta, jet 3 phi, jet 3 b-tag, jet 4 pt, jet 4 eta, jet 4 phi, jet 4 b-tag	21 low-level features. These are kinematic attributes
m_jj, m_jjj, m_lv, m_ljlv, m_bb, m_wbb, m_wvbb	7 high-level features employed to distinguish between the two classes

Table 5
The accuracy results for each Spark worker node.

Iteration	SVM.k.t	SVM.e.p	SVM.q.l	SVM.s	SVM.k.c	SVM.m.i	Accuracy
(a) Worker node results 1							
1.0	dot	true	true	true	0.0	1.0	0.5161
2.0	radial	true	true	true	0.0	1.0	0.5483
3.0	neural	true	true	true	0.0	1.0	0.5
4.0	anova	true	true	true	0.0	1.0	0.5161
5.0	dot	false	true	true	0.0	1.0	0.5483
(b) Worker node results 2							
200.0	anova	false	true	true	60.0	1.0	0.3870
201.0	dot	true	false	true	60.0	1.0	0.5483
202.0	radial	true	false	true	60.0	1.0	0.5322
203.0	neural	true	false	true	60.0	1.0	0.5483
204.0	anova	true	false	true	60.0	1.0	0.5645
(c) Worker node results 3							
1000.0	anova	false	true	true	90.0	21.0	0.8548
1001.0	dot	true	false	true	90.0	21.0	0.7580
1002.0	radial	true	false	true	90.0	21.0	0.5967
1003.0	neural	true	false	true	90.0	21.0	0.5967
1004.0	anova	true	false	true	90.0	21.0	0.8709
(d) Worker node results 4							
3868.0	anova	true	false	false	100.0	100.0	0.6774
3869.0	dot	false	false	false	100.0	100.0	0.6774
3870.0	radial	false	false	false	100.0	100.0	0.8709
3871.0	neural	false	false	false	100.0	100.0	0.5967
3872.0	anova	false	false	false	100.0	100.0	0.6935

Execution Time comparison Serial Vs Parallel GridSVM

Figure 8. Execution time comparison of Serial SVM vs. PGOSVM for Covtype and Higgs datasets.

Another dataset used in this study is the HIGGS dataset, a Physics dataset generated from particle detectors in the accelerator. Monte Carlo simulations were used to produce the data which is almost balanced as it consists of 53% positive samples. There are 11,000,000 samples in the dataset (total size = 4 GB). The first feature in this dataset is the class variable which has 2 values (1 and 0 which represents signal and background, respectively). There are more 28 features in the dataset which follow the class variable; the first 21 features are classified as low-level features while the remaining 7 features are high-level features. The 21 low-level features are kinematic attributes which are measured in the accelerator by the particle detectors and mapped to high-level features for defining the class value. Table 4 described the features of the HIGGS dataset.

9. Results and discussion

The platform used in this study is as follows: Intel® Core™ i7-6700 CPU @ 3.40 GHz with 8 logical cores, 16 GB RAM, and Windows OS. The experiments employed 3 real-world datasets from the UCI database which were used to evaluate the proposed approach in terms of its classification ability in different classification problems. Table 2 presented the number of classes, samples, and original features used during the experiments.

The evaluation of the classification performance of the proposed method was done based on 3 indices which are the overall accuracy, sensitivity, and specificity. Regarding sensitivity, it portrays the actual number of positive instances which are rightly classified as positive. Sensitivity is computed thus:

$$sensitivity = \frac{TP}{TP + FN}$$

where TP = number of true positives, FN = number of false positives.

The specificity is the percentage of the correctly identified negatives by the classifier; it is calculated thus:

$$specificity = \frac{TN}{TN + FP}$$

where TN and FP = number of true negatives and false positives, respectively.

The overall accuracy denotes the models' capability of correctly assigning unseen data to its correct class. It is computed thus:

$$overallaccuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

The partitioning of the SVM hyper-parameter into N combinations is aimed at making the parallel execution efficient. Each set of parameter combination is forwarded to Spark worker node to train SVM with

the training set. The high accuracy results for each combination will be selected by voting from each worker node. Table 5 (a), (b), (c), and (d) showed the accuracy results for each Spark worker node.

The PGOSVM represents a significant improvement compared to the serial SVM in terms of the process execution time (SVM required 40 s to complete Covtype dataset execution while PGOSVM required only 12 s for the same task). On the Higgs dataset, SVM required 98 s to complete the execution while PGOSVM completed the execution within 28 s (refer to Fig. 8). The estimation of the speedup latency is done as follows:

$$S_{\text{Latency}} = \frac{L_{\text{Serial}}}{L_{\text{Parallel}}}$$

where L_{Serial} = latency attributable to serial execution, L_{Parallel} = latency attributable to parallel execution, and S_{Latency} = the speedup. Hence, S_{Latency} for the Covtype dataset is computed as follows:

$$S_{\text{Latency}} = \frac{L_{\text{Serial}}}{L_{\text{Parallel}}} = \frac{40}{12} = 3.33$$

Similarly, S_{Latency} for HIGGS dataset is calculated thus:

$$S_{\text{Latency}} = \frac{L_{\text{Serial}}}{L_{\text{Parallel}}} = \frac{98}{28} = 3.5$$

Thus, the achievement of 3.33 times speedup for the Covtype dataset and 3.5x speedup for Higgs dataset is remarkable when compared to serial execution. The execution time of Higgs dataset was faster compared to that of Covtype dataset despite the larger size of the Higgs dataset compared to the Covtype dataset. This variation is a function of the data distribution.

10. Conclusions and future work

The huge volume of available data in many fields is posing new problems for ML techniques. The extraction of important information from these huge datasets in order to generalize the obtained knowledge to the unknown new data requires a proper tackling of the feature selection and parameter optimization problems. Proper classification of the unknown new patterns can only be ensured through a proper setting of the parameters. In this study, a hybrid PGOSVM-based model which is a combination of the SVM with parallel grid search

optimization was proposed for ensuring classification accuracy even when the number of feature subsets is small but appropriate. This study implemented the PGOSVM with Spark Radoop with distributed data points storage using HDFS. The proposed PGOSVM achieved a better classification efficiency on a large dataset and was faster in terms of execution time compared to the benchmarked classification method. The PGOSVM was executed using the Scala programming language. The analyzed datasets in this study were the Covtype and Higgs datasets sourced from the UCL database. The major contribution of this study is the development of a simple but innovative PGOSVM for SVM parameters which can participate parallelly in the training of every SVM on the Radoop platform. The PGOSVM was found to achieve 3.3 times faster computation time on the Covtype dataset and 3.5 times faster computation time on the Higgs dataset compared to the performance of serial SVM. An interesting aspect is that the PGOSVM can be hybridized with any other ML techniques and used to solve several data mining problems. For instance, PGOSVM can be hybridized with PNN for solving classification problems, with GRNN to solve regression problems, as well as with RBFN to solve both regression and classification tasks. Lastly, the future direction for PGOSVM application is scaling-up and applying the technique to big data analytics problems with streaming dataset.

Acknowledgments

The authors are grateful to ICCI, Informatics Institute for Postgraduate Studies (IIPS_IRAQ) for their moral support. Many thanks for Mustansiriya university (www.uomustansiriyah.edu.iq) for its support to finish this paper. Special thanks to the anonymous reviewers for their valuable suggestions and constructive comments.

References

- [1] B. Chambers, M. Zaharia, Spark: the Definitive Guide: Big Data Processing Made Simple, O'Reilly Media, Inc., 2018.
- [2] M. Mittal, V.E. Balas, L.M. Goyal, R. Kumar, Big Data Processing Using Spark in Cloud, Springer, 2019.
- [3] B. Hosseini, K. Kiani, A Big Data Driven Distributed Density Based Hesitant Fuzzy Clustering Using Apache Spark with Application to Gene Expression Microarray, Engineering Applications of Artificial Intelligence, 2019, pp. 100–113.

- [4] A.H. Ali, M.Z. Abdullah, Recent trends in distributed online stream processing platform for big data: Survey, in: 2018 1st Annual International Conference on Information and Sciences (AiCIS), 2018, pp. 140–145.
- [5] A.H. Ali, A survey on vertical and horizontal scaling platforms for big data analytics, *Int. J. Int. Eng.* (2019) 138–150.
- [6] R.A. Hasan, I. Alhayali, A. Royida, N.D. Zaki, A.H. Ali, An Adaptive Clustering and Classification Algorithm for Twitter Data Streaming in Apache Spark, *Telkomnika*, 2019.
- [7] T.R. Rao, P. Mitra, R. Bhatt, A. Goswami, The Big Data System, Components, Tools, and Technologies: a Survey, *Knowledge and Information Systems*, 2018, pp. 1–81.
- [8] J.C. Reis, A. Correia, F. Murai, A. Veloso, F. Benevenuto, E. Cambria, Supervised Learning for Fake News Detection, *IEEE Intelligent Systems*, 2019, pp. 76–81.
- [9] M.A. Mohammed, N. Tăpuș, A novel approach of reducing energy consumption by utilizing enthalpy in mobile cloud computing, *Stud. Inf. Contr.* (2017) 425–434.
- [10] D. Krotov, J.J. Hopfield, Unsupervised learning by competing hidden units, *Proc. Natl. Acad. Sci. U. S. A.* (2017) 7723–7731.
- [11] M. Botvinick, S. Ritter, J.X. Wang, Z. Kurth-Nelson, C. Blundell, D. Hassabis, Reinforcement learning, fast and slow, *Trends Cognit. Sci.* 23 (5) (May 2019) 408–422.
- [12] A.-H.A. Salih, A.H. Ali, N.Y. Hashim, in: *Jaya: an Evolutionary Optimization Technique for Obtaining the Optimal Dthr Value of Evolving Clustering Method (ECM)*, Vol. 11, 2018, pp. 1901–1912. Number 12.
- [13] J.A. Suykens, J. Vandewalle, Least Squares Support Vector Machine Classifiers, *Neural processing letters*, 1999, pp. 293–300.
- [14] A.J. Smola, B. Schölkopf, A Tutorial on Support Vector Regression, *Statistics and computing*, 2004, pp. 199–222.
- [15] Z. Chen, T. Lin, N. Tang, X. Xia, A Parallel Genetic Algorithm Based Feature Selection and Parameter Optimization for Support Vector Machine, *Scientific Programming*, 2016.
- [16] T. Eitrich, B. Lang, Parallel tuning of support vector machine learning parameters for large and unbalanced data sets, in: *International Symposium on Computational Life Science*, 2005, pp. 253–264.
- [17] N. Hou, F. He, Y. Zhou, Y. Chen, X. Yan, A Parallel Genetic Algorithm with Dispersion Correction for HW/SW Partitioning on Multi-Core CPU and Many-Core GPU, *IEEE Access*, 2017, pp. 883–898.
- [18] L. Grama, L. Tuns, C. Rusu, On the optimization of SVM kernel parameters for improving audio classification accuracy, in: 2017 14th International Conference on Engineering of Modern Electric Systems (EMES), 2017, pp. 224–227.
- [19] Y. Bao, Z. Liu, A fast grid search method in support vector regression forecasting time series, in: *International Conference on Intelligent Data Engineering and Automated Learning*, 2006, pp. 504–511.
- [20] H.N. Dinh, M. Van der Baan, A grid-search approach for 4D pressure-saturation discrimination, *Geophysics* (2019) IM47–IM62.
- [21] G.K. Thiruvathukal, C. Christensen, X. Jin, F. Tessier, V. Vishwanath, A benchmarking study to evaluate Apache Spark on large-scale supercomputers, *arXiv preprint arXiv:1904.11812* (2019).
- [22] P. Asha, T.P. Jacob, A. Pravin, A. Asbern, Mining the associated patterns in big data using Hadoop cluster, in: *International Conference on Intelligent Data Communication Technologies and Internet of Things*, 2018, pp. 1255–1263.
- [23] A. Kuo, D. Chrimes, P. Qin, H. Zamani, A Hadoop/MapReduce Based Platform for Supporting Health Big Data Analytics, *Studies in health technology and informatics*, 2019, pp. 229–235.
- [24] A. Snow, *Data Locality Practices of MapReduce and Spark: Efficiency and Effectiveness*, 2018.
- [25] E. Ghasemi, P. Chow, *Accelerating Apache Spark with Fpgas, Concurrency and Computation: Practice and Experience*, 2019, p. e4222.
- [26] A. Alnafessah, G. Casale, Anomaly detection for big data technologies, in: 2018 Imperial College Computing Student Workshop (ICCSW 2018), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
- [27] <https://rapidminer.com/>.