

# **Implementation and Experiments on Distributed Ensemble Learning System (DELS) With Several Partitioning Methods and Classifiers**

**Azadeh Zamani**

Submitted to the  
Institute of Graduate Studies and Research  
in partial fulfillment of the requirements for the degree of

Master of Science  
in  
Computer Engineering

Eastern Mediterranean University  
February 2021  
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

---

Prof. Dr. Ali Hakan Ulusoy  
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science in Computer Engineering.

---

Prof. Dr. Hadi Işık Aybay  
Chair, Department of Computer  
Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

---

Assoc. Prof. Dr. Alexander Chefranov  
Supervisor

---

Examining Committee

1. Assoc. Prof. Dr. Adnan Acan

2. Assoc. Prof. Dr. Alexander Chefranov

3. Asst. Prof. Dr. Öykü Akaydın

## ABSTRACT

Nowadays, Machine Learning in Big Data is one of the challenges. As the large datasets are too big to handle in the single node memory using distributed method is mandatory. Hence, the methods of distributing data return the results with high accuracy and better performance in time is the goal of this research. Using various learning processes to train multiple classifiers from distributed data sets increases the possibility of achieving higher accuracy, particularly on a big datasets. This is because the combination of classifiers can represent an integration of different learning biases which may compensate for each other's inefficiencies.

Implementation and Experiments on Distributed Ensemble Learning System (DELS) With Several partitioning Methods and classifiers in single and multiple systems have been chosen. The user should choose the input dataset, the number of partitions and the classifier. Classification and regression tree (CART) and multilayer perceptron (MLP) are the selected classifier used of decision tree and neural network methods, respectively. We assume that number of partition is related to the number of disjoint bagging which will be used for division of data and consequently the number of parallel processors to which data is sent. Algorithms of bagging the data are disjoint partitions (D), disjoint bags (DB), small bags (SB) and No-replication small bags (NRSB) classification. These stratified inputs are proposed as training samples and will train in single machine. The distribution of each part of this stratified input is done by MPI. This service is responsible for performing several tasks with its own resources separately. The task includes implementing the learning algorithm and extracting the learning model. The results are N training models which

are collected using the majority vote method. The model with higher prediction rank is selected in major voting. This final model is used to check the test data and extract the Scoring test result. The previous test is repeated in multi-node system with random input dataset.

In single-node, SB (Small Bag) has highest and D (Disjont Partition) has lowest accuracy. CART has 0.998 in accuracy while MLP has 0.96. MLP requires 2 to 11 more times for learning than CART. In multi-node run time in CART is 5 to 11 times faster than MLP. The best test score we reach was 0.955. As the number of disjoint partitions is increased scoring time will increase, thus in 2 partitions scoring time is 37 minutes while in 12 partitions it is 210 minutes.

In DELS, better training time get with LADEL and MLP algorithm than CART. It takes 4.6 seconds in 2 nodes while training time decrease to 0.11 second in 12 nodes by using MLP in multi-node. These results are obtained by the CART algorithm in a multi-node system, 207 and 7.01 seconds for 2 and 12 nodes, respectively.

**Keywords:** distributed systems, parallel processing, ensemble learning, bagging, classification, decision tree, neural network, disjoint partition

## ÖZ

Günümüzde, Büyük Veride Makine Öğrenimi zorluklardan biridir. Büyük veri kümeleri, dağıtılmış yöntem kullanılarak tek düğüm belleğinde işlenemeyecek kadar büyük olduğundan zorunludur. Bu nedenle, sonuçları yüksek doğrulukta ve zamanında daha iyi performansla döndüren yararlı algoritmalar kullanılarak verilerin dağıtılması yöntemi bu araştırmanın amacıdır.

Tekli ve çoklu sistemlerde Çeşitli Bölümleme Yöntemleri ve Sınıflandırıcılar ile Dağıtılmış Toplu Öğrenme Sisteminde (DELS) Uygulama ve Deneyler seçilmiştir. Kullanıcı, giriş veri setini, bölüm sayısını ve sınıflandırıcıyı seçmelidir. Sınıflandırma ve Regresyon Ağacı (CART) ve Çok Katmanlı Algılayıcı (MLP), sırasıyla karar ağacı ve sinir ağı yöntemlerinde kullanılan seçili sınıflandırıcıdır. Bölüm sayısının, verilerin bölünmesi için kullanılacak ayrık torbalama sayısı ve dolayısıyla verilerin gönderildiği paralel işlemcilerin sayısı ile ilişkili olduğunu varsayıyoruz. Verileri torbalama algoritmaları, Ayrık bölümler (D), Ayrık Torbalar (DB), Küçük Torbalar (SB) ve Çoğaltmasız Küçük Torbalar (NRSB) sınıflandırmasıdır. Bu tabakalandırılmış girdiler eğitim örnekleri olarak önerilmektedir ve tek makinede eğitilecektir. Bu tabakalandırılmış girdinin her bir bölümünün dağılımı MPI tarafından yapılır. Bu hizmet, çeşitli görevleri kendi kaynakları ile ayrı ayrı yapmaktan sorumludur. Görev, öğrenme algoritmasının uygulanmasını ve öğrenme modelinin çıkarılmasını içerir. Sonuçlar, ana oylama tekniği kullanılarak bir araya getirilen N eğitim modelidir. Büyük oylamada tahmin sıralaması daha yüksek olan model seçilir. Bu Son model, test verilerini test etmek ve

Puanlama test sonucunu çıkarmak için kullanılır. Önceki test, rasgele giriş veri kümesiyle Çoklu Düğümlü sistemde tekrarlanır.

single-node SB'de (Small Bag) en yüksek, D (Disjont Partition) ise en düşük doğruluğa sahiptir. CART'ın doğruluğu 0.998 iken MLP 0.96'dır. MLP, öğrenme için CART'tan 2 ila 11 kez daha fazla gerektirir. Çok Düğümlü yürütme işleminde CART'ta, MLP'den 5 ila 11 kat daha hızlıdır. Test veri setini test ederken ulaştığımız en iyi test puanı 0,955 idi. Ayrık bölüm sayısı arttıkça çalışma süresi artacaktır, dolayısıyla 2 bölümde çalışma süresi 37 dakika, 12 bölümde 210 dakikadır. LADEL modelinde en iyi ve en kötü test puanı 6 bölümde 0,931 ve 10 bölümde 0,192'dir.

**Anahtar Kelimeler:** dağıtık sistemler, Paralel işleme, Topluluk öğrenimi, Torbalama, Sınıflandırma, Karar ağacı, Sinir Ağı, Ayrık Bölüm

## **DEDICATION**

I dedicate my dissertation to my loving family.

In memory of my father Asghar Zamani who always encouraged me to be independent and strong as long as he was with me. A special feeling of gratitude to my mother who has always been the myth of patience in my life and has always encouraged me to continue on my path with her boundless love and affection. I dedicated it to my brothers Bahram and Shahram who have always been my supporters in this direction and have motivated me to continue. I also dedicate this dissertation to Leila for all her compassionate counseling by love like a sister, and finally to my beloved nephews Anisa and Ava who are always give me energy and love.

I will always be appreciating all they have done for me. Thank you for all your support and love.

## ACKNOWLEDGMENT

I wanted to have a special thanks to Assoc. Prof. Dr. Alexander Chefranov, my supervisor, because of the countless hours of reflection, reading, encouragement and most of all patience in the whole process. He was the one who led me to acquire and expand new knowledge so that I could finish this dissertation in the best possible way. Thanks to him, I experienced real research and my knowledge on the subject has expanded.

I sincerely appreciate Assoc. Prof. Dr. ÖNSEN TOYGAR, Assoc. Prof. Dr. MEHMET BODUR, Prof. Dr. EKREM VAROĞLU, Assoc. Prof. Dr. DUYGU ÇELİK ERTUĞRUL, Assist. Prof. Dr. YILTAN BITIRIM and Assist. Prof. Dr. NİLGÜN HANCIOĞLU and all of the instructors I have been in direct contact with and who have influenced me during this program.



# TABLE OF CONTENTS

ABSTRACT .....	iii
ÖZ .....	v
DEDICATION .....	vii
ACKNOWLEDGMENT .....	viii
LIST OF TABLES .....	xii
LIST OF FIGURES .....	xiii
1 INTRODUCTION .....	1
2 RELATED WORKS AND PROBLEM DEFINITION .....	4
2.1 Classification and regression tree (CART), and multilayer perceptron (MLP) classifiers .....	4
2.1.1 CART classifier .....	6
2.1.2 MLP classifier .....	8
2.2 Single and multiple-node classifier ensembles .....	13
2.2.1 Single-node Bagging-like ensembles .....	13
2.2.2 Multi-node Bagging-like ensembles .....	16
2.3 Partitioning methods for Bagging-like ensembles .....	17
2.3.1 Disjoint partitioning .....	18
2.3.2 Small bags partitioning .....	20
2.3.3 No-replication small bags partitioning .....	21
2.3.4 Disjoint bags partitioning .....	23
2.4 The label-aware distributed ensemble learning (LADEL) .....	24
2.5 LADEL evaluation .....	28
2.6 Parallel computing .....	30

2.7 Client-server architecture .....	33
2.8 Message passing interface (MPI) .....	34
2.9 MPI in Python .....	35
2.10 Problem definition.....	37
<b>3 DESIGN, IMPLEMENTATION AND EXAMINATION OF DELS .....</b>	<b>39</b>
3.1 DELS design and architecture.....	40
3.2 Design and implementation of IOS.....	44
3.3 Design and implementation of partitioning subsystem.....	46
3.4 Design and implementation of training subsystem .....	48
3.5 Design and implementation of testing subsystem.....	50
3.6 Testing DELS .....	52
3.6.1 Accuracy comparison for Bagging-like partitioning method.....	52
3.6.1.1 Accuracy in single-node.....	52
3.6.1.2 Accuracy in multi-node architecture.....	55
3.6.2 Prediction and test-score and runtime comparison for Bagging-like partitioning method and LADEL algorithm.....	57
<b>4 EXPERIMENTS ON DELS .....</b>	<b>58</b>
4.1 Experimental setup.....	59
4.1.1 Environment setup .....	59
4.1.2 Datasets .....	60
4.2 Accuracy comparison for single-node classifier ensemble .....	61
4.3 Accuracy comparison for multi-node classifier ensemble.....	64
4.3.1 Prediction comparison for Bagging-like partitioning method .....	607
4.4 Training time and scoring time comparison for LADEL.....	68
<b>5 CONCLUSION .....</b>	<b>72</b>

REFERENCES.....	75
APPENDICES.....	78
Appendix A: Software source code- Implementation of DELS.....	79
Appendix B: Experimental results of DELS .....	102
Appendix C: Experimental results of DELS system.....	107
Appendix D: Experimental results of DELS system.....	112
Appendix E: Experimental results of DELS system.....	117
Appendix F: Comparison on “KDDCUP” dataset with CART classifier on single node system.....	122
Appendix G: Comparison on “KDDCUP” dataset with MLP classifier on single node system. ....	123
Appendix H: Comparison on “KDDCUP” dataset with CART classifier on multi node system.....	124
Appendix I: Comparison on “KDDCUP” dataset with MLP classifier on multi node system.....	125
Appendix J:Experimental results of DELS system.....	126
Appendix K: Experimental results of DELS system on HIGGS .....	130
Appendix L: Experimental results of DELS system on HIGGS.....	132
Appendix M: Experimental results of DELS system on KDD Cup 99.....	133

## LIST OF TABLES

Table 4. 1: Input data set attributes .....	60
Table 4.2: Run-time in single-node mode.....	64
Table 4.3: Run-time in multi-node mode.....	66
Table 4.4: Accuracy of model in training dataset by Cross-validation & accuracy of model in testing dataset by test scores in Bagging-like methods by MLP .....	67
Table 4.5: Training and scoring time in Bagging-like methods by MLP in multi-node mode.....	68

## LIST OF FIGURES

Figure 2.1: Splitting training data into regions .....	6
Figure 2.2: Decision tree model for Figure 2.1 .....	6
Figure 2.3: Splitting training data into $t$ regions .....	8
Figure 2.4: Decision tree model for Figure 2.3 .....	8
Figure 2.5: Perceptron .....	9
Figure 2.6: Multilayer perceptron (MLP) .....	10
Figure 2.7: Single-node Bagging ensemble classifier schema.....	15
Figure 2.8: Multi-node Bagging-like classifier ensemble schema.....	17
Figure 2.9: Four Bagging-like strategies: D, SB, NRSB, DB [2] .....	17
Figure 2.10: Disjoint partitioning method D [2] .....	18
Figure 2.11: An illustration of small bags, SB [2] .....	20
Figure 2.12: An illustration of No-replication small bags, NRSB [2] .....	22
Figure 2.13: An illustration of disjoint bags, DB [2] .....	23
Figure 2.14: LADEL execution overview [3] .....	25
Figure 2.15: single-node stratified sampling [3] .....	26
Figure 2.16: Distributed stratified sampling (DSS) [3].....	27
Figure 2.17: Ratio of the training time of distributed ensemble to a sequential single node classifier [3].....	29
Figure 2.18: Ratio of the scoring time of single machine ensemble to a sequential single node classifier [3] .....	30
Figure 2.19: Ratio of distributed scoring time on 10 machines of an ensemble to that of a single classifier on a single machine [3] .....	30
Figure 2.20: Distributed memory model [17] .....	34

Figure 2.21: Shared memory model [19] .....	34
Figure 2.22: Hybrid memory model [17] .....	35
Figure 3.1 DELS architecture .....	40
Figure 3.2 Input/output subsystem (IOS).....	45
Figure 3.3 Partitioning subsystem.....	47
Figure 3.4 Training subsystem.....	49
Figure 3.5 Testing subsystem.....	51
Figure 3.6 Program execution in terminal, single-node.....	53
Figure 3.7 Resource monitoring in master system with 8 processors.....	54
Figure 3.8 Program execution in terminal, multi-node .....	56
Figure 3.9 Resource monitoring in client system with 4 processors .....	57
Figure 4.1 Accuracy comparison on Bagging-like ensembles by MLP for 2 disjoint partitions [Appendix G] .....	62

# Chapter 1

## INTRODUCTION

Nowadays, extracting useful information from large amounts of data stored in databases and warehouses known as data mining is important in system analysis. Data mining [1] is the process of finding meaningful statistical patterns in a large amount of data. It applies in many fields including business, science, telecommunication, and medicine to predict protein structure [2], prevents network intrusion detection, and recognizes breast cancer from mammography tests. Finding these statistical patterns based on some techniques including classification, regression, clustering, and association rules.

In 1959, Arthur Samuel defined machine learning as a “field of study that gives computers the ability to learn without being explicitly programmed” [4]. Generating disjoint subsets of training dataset fitting memory [2] allows decreasing training time. Thus, breaking big data into several smaller files and processing each part is necessary. The training algorithms execute on each part to produce multiple models aggregated into an ensemble. Prediction results of the ensemble models are combined by majority voting to produce improved results. Partitioning data into smaller parts (disjoint partitions) affects accuracy and computational efficiency of the ensembles of classifiers.

There are several methods to partition big data. For example, Chawla et al [2] used various Bagging-like partitioning methods. They were tested on small-, medium-, and large-sized data. There are two difficulties in using Bagging-like methods. 1- When the training dataset order by the class labels. A class label attributes a data to a particular class. If a training dataset arrange on the class label, small partitions probably contain data belonging to a single label. Therefore, the algorithm always predicts the same single class. 2- When the proportion of records in one class is significantly different than another class. Having highly skewed training dataset may prevent some partitions from including minority class labels records, which may result in not recognizing minority class records.

The label\_aware distributed ensemble learning (LADEL) [3] algorithm has an add-on model for the Bagging-like model that does not have the above problems by including all class labels in each partition, while no class labels are included in Bagging-like partitions. Classifiers can be created in a distributed way on disjoint partitions. Each classifier can be learned in parallel on a separate processor.

In this work, we shall consider a problem of large-scale data breaking down into smaller parts and train several classifiers from distributed data sets separately. These partitions are generated by the Bagging-like method and LADEL distributed over multiple processors' memories, and classifiers are trained separately on them. To generate the final classifier, the trained classifiers are aggregated using majority voting to produce the final classifier. These distributed and parallel processing may improve accuracy and the training time.



The rest of the thesis is set as follows: Chapter 2 surveys classifiers, Bagging classifier ensembles, the segmentation methods for Bagging-like distributed ensembles of classifiers, LADEL model, describes experimental settings, and results for them. The problems of the thesis are defined. Chapter 3 explains the system architecture design, implementation, and testing of the distributed ensemble learning system (DELS). Chapter 4 introduces experimental settings and results on DELS study and comparison of DELS versus LADEL and other models. Chapter 5 is conclusion.

## Chapter 2

### RELATED WORKS AND PROBLEM DEFINITION

In data mining and Machine learning, discovering the patterns in data is used to predict (classify) new data. Data maybe labeled or unlabeled. Labels tag labeled data. For example, a collection of patient data containing their test results in the diagnosis of cancer label with one (positive), or zero (negative). Machine learning approaches classify into supervised and unsupervised based on whether data labeled or not. If the training data labeled, it is in the supervised group, otherwise, in the unsupervised group. Supervised learning algorithms use Classification and Regression, while unsupervised learning algorithms use Clustering and Association Rules.

In this chapter, CART and MLP classifiers (Section 2.1), single and multiple-node classifier ensembles, partitioning methods for Bagging-like ensembles (Section 2.3), LADEL, and LADEL evaluation (Section 2.4 and 2.5), Parallel computing (Section 2.6), Client-server architecture (Section 2.7), MPI and MPI with Python (Section 2.8 and 2.9) and problem definition (Section 2.10) are considered.

#### **2.1 Classification and regression tree (CART), and multilayer perceptron (MLP) classifiers**

One of the main purposes of data mining is to predict the unknown value of a new sample based on previous samples. Achieving such a result is performed in two steps:

A) Training step: Create a predictive model [6] on training samples [7] using some

learning algorithm.

B) Test step: The created predictive model is used to classify test samples.

In machine learning, usually an input dataset is divided into two parts, training and testing samples. The training dataset is a dataset of records used during learning process; it is used to tune the parameters of a classifier (train model). The testing dataset is a dataset that is independent of the training dataset, and is a set of records used only to assess the accuracy of the trained model (learned classifier)]. In supervised learning, training data has an attribute named “class label” for classifying the data based on their common attribute and predicting a class of testing data. In learning process on a training dataset, algorithm builds a model based on the class of data specified by the class labels. Then, the model built is used to predict (classify) test data. Identifying the class of a test sample is called classification.

Decision tree learning [8] is a method of machine learning that creates a model for predicting the amount of output variables depend on input values. A decision tree is a tree structure used to categorize data. It breaks down an input dataset into smaller subsets and decision tree is incrementally developed. A decision tree defined as a group of nodes which is started with root node as first parent node and continue breaking down the parent node recursively into child nodes based on a series of decisions using the variables in the dataset. The final child nodes in tree are called leaves and the final result is a tree with decision (parent) nodes and leaf (child) nodes. The examples will be considered in the CART section (section 2.1.1).

### 2.1.1 CART classifier

Classification and Regression Tree (CART) [9] is tree-based method which creates a binary decision tree model, meaning each parent node has two children. It is supervised learning because the training data is pre-classified using labels. The strong point of this method is that it allows diverse types of input data. The input data can be numerical, like price, or non-numerical, or categorical, like location. CART versatility makes it a popular tool for a wide range of input data types. It works with all types of input data by “classification tree” or “regression tree”.

For classification tree, Figure 2.1 shows points in the plane either marked with cross or circle as an illustration of input data. This display shows the values of two variables for a set of training data. The training data is displayed as a set of points, each of the points having two values, one is the position-determining variable on the horizontal axis (x) and the other is the value of the position-determining variable on the vertical axis (y). These points have two shapes: “cross”, or “circle”.

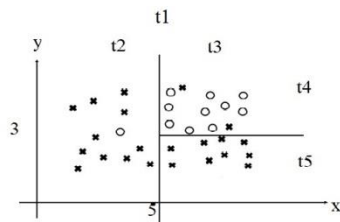


Figure 2.1: Splitting training data into regions

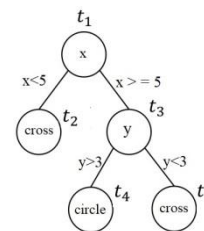


Figure 2.2: Decision tree model for Figure 2.1

Cross and circle are two class labels. The purpose of the classifier is to partition the plane into smaller regions and assign a class label to each region by majority number of class labels. First, the training dataset of the points  $(x_i, y_i)$ ,  $i=1, 33$ , labeled by stars and circles, is in the whole region labeled by  $t_1$  (single-node). Then, split  $t_1$  into

two regions (nodes),  $t_2$  (where  $x_i < 5$ ), and  $t_3$  (where  $x \geq 5$ ).  $T_2$  label with a cross because it has more crosses than circles. Therefore, this area no longer needs further division and known as the leaf in the decision tree, but for  $t_3$ , we cannot decide on the labeling of this area. Continue splitting  $t_3$  into  $t_4$  (for  $y_i \geq 3$ ) and  $t_5$  (for  $y_i < 3$ ). Now assign the class label into each region. The region  $t_4$  is marked by “circle”, because has more circles than crosses, and  $t_5$  is marked by “cross”. Figure 2.2 shows decision tree for Figure 2.1. Decision on splitting each region made by some rules. These rules split each region in a decision tree into two sub-regions, so decision tree is called binary decision tree. Once the decision tree is built, the trained model is ready. This model is used to find new data class labels and prediction of new data class labels is possible. For example, assume the new data point is  $A = \{(10, 2, ?)\}$ , it has not class label. The prepared decision tree is a model to predict the class label for point A. As  $x = 10$  and its greater than 5, it belongs to  $t_3$ . Then  $y = 2$  and it is less than 3, hence, it belongs to  $t_5$ . According to model, A belongs to  $t_5$  with the class label cross. Thus,  $A = \{(10, 2, \text{cross})\}$ .

For Regression Tree method, Figure 2.3 shows points in the xy-plane by crosses. In this case, goal is to find a function  $y = f(x)$  whose graph lies close to the given data points. Figure 2.4 shows decision tree for the data in Figure 2.3. The method of the regression tree is similar to classification tree. The only difference is that instead of finding majority number of class labels for decision making, in regression tree, the average y-value of the data is used as the value of the regression function on each region and the target value of decisions can take continuous values (typically real numbers). For creating the decision tree firstly, the training dataset of the points  $(x, y)$  is in the whole region labeled  $t_1$ . Then, split  $t_1$  into two regions (nodes),  $t_2$  (where

$x < \bar{x}$ , and  $y(x) = \bar{y}$  is the average of  $y$ , for example  $\bar{x} = 4$ ), and  $t_3$  (where  $x \geq \bar{x}$  with what average of  $y$ ?). The region  $t_2$  can again be split into  $t_4$  (where  $x < \bar{x} = 2$  and average of  $y$  is  $y_4$ ) and  $t_5$  (where  $x \geq \bar{x} = 2$  and average of  $y$  is  $y_5$ ). For  $t_3$ , split it into  $t_6$  (where  $x < \bar{x} = 6$  and average of  $y$  is  $y_6$ ), and  $t_7$  region (where  $x \geq \bar{x} = 6$ , and average of  $y$  is  $y_7$ ). For prediction,  $y$  value for the known  $x$  value by decision tree, region of  $x$  value will be selected in decision tree, then the related  $y$  value will be assigned to  $y(x)$ .

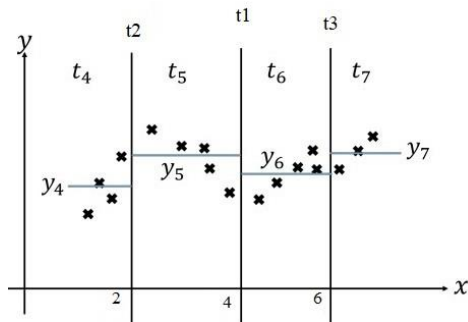


Figure 2.3: Splitting training data into  $t$  regions

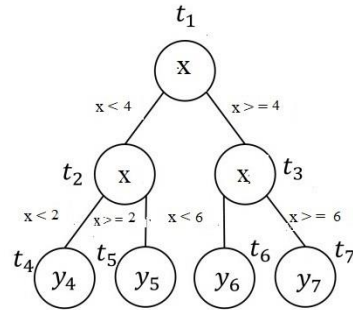


Figure 2.4: Decision tree model for Figure 2.3

### 2.1.2 MLP classifier

Multilayer perceptron (MLP) [10] is one of the artificial Neural Networks (ANN) [11] classifier. Neural networks include a set of nodes and the connections between them. MLP is a supervised learning method that has multiple layers of nodes.

Perceptron [10] is used for supervised learning and making decisions based on a linear predictor function. A function that maps its input  $x$  (a real-valued vector) to an output value  $f(x)$  (a single binary value) is an activation function of the perceptron.

$$f(x) = o(x_1, x_2, \dots, x_n) = \begin{cases} 1, & w \cdot x + b > 0 \\ 0, & \text{otherwise} \end{cases}, \quad (1)$$

where  $n$  is the number of  $x$  input values,  $o(\vec{x})$  is output of  $x$  vector,  $w$  is a vector of real-valued weights,  $w \cdot x = \sum_{i=1}^n w_i \cdot x_i$  is the scalar product of the vectors  $w$  and  $x$ , and  $b$  is a bias, or threshold, a term that shifts the decision boundary away from the origin. The perceptron is triggered only when weighted input reaches a certain threshold value (0, in our case,  $w \cdot x + b > 0$ ). An output of one specifies that the perceptron is triggered; otherwise, the perceptron was not triggered. If  $b$  is negative, then to force the classifier perceptron over the 0 threshold, the scalar product shall exceed  $b$ .

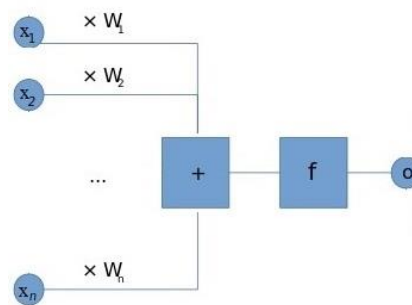


Figure 2.5: Perceptron

Figure 2.5 demonstrates the  $n$  desired input values of  $x(x_1, x_2, \dots, x_n)$  and  $W(w_1, w_2, \dots, w_n)$  as input weights. The input values multiply by weights and then summed up. The activation function  $f(x)$  applies on the sum value to determine, trigger, or not to generate output (O). Training dataset includes input value and desired output value,  $d(x)$ . Learning process starts by setting of the weights randomly and continues to optimize weights in a way that the perceptron output is equal to the desired output. This weight updating is done based on the error ( $\epsilon$ ). Error is the difference between the output obtained in perceptron and the desired output ( $\epsilon = d(x) - f(x)$ ). If the error value is not zero, the output is incorrectly predicted and the weights must be updated. The learning process repeats several times and update

weights to find the optimize weights with zero or at least less error value. When the perceptron output is equal to the desired output, the perceptron finds the optimal weight and the repetition process stops. Pseudocode of the perceptron algorithm is given below in Code 1[36, p. 71].

*Perceptron algorithm*

*Assumption: the two desired output(classes),  $d(x) = 1$  and  $d(x) = 0$ , are linearly separable.*

1. *Initialize all weights,  $w_i$ , to small random numbers.*

*Choose an appropriate learning rate,  $\alpha \in (0,1]$ .*

2. *For each training example,  $x = (x_1, x_2, \dots, x_n)$  and  $x_0$ .  $w_0$  is bias, whose class is  $d(x)$ :*

2.1.1. *Let  $f(x) = 1$  if  $\sum_{i=0}^n w_i x_i > 0$ , and  $f(x) = 0$  otherwise.*

2.1.2. *Update each weight using formula,  $w_i = w_i + \alpha[d(x) - f(x)].x_i$*

3. *If  $d(x) = f(x)$  for all training examples, stop; otherwise, return to step 2.*

*End//perceptron*

*Code 1. Pseudocode of perceptron algorithm*

**MLP** utilizes a supervised learning method for training. It is an artificial Neural Network (ANN) with at least three layers: input, hidden, and output layers, shown in Figure 2.6. Single layer perceptron can learn only linearly separable patterns, while multilayer perceptron can learn patterns that are more complicated. MLP distinguish data that are not linearly separable.

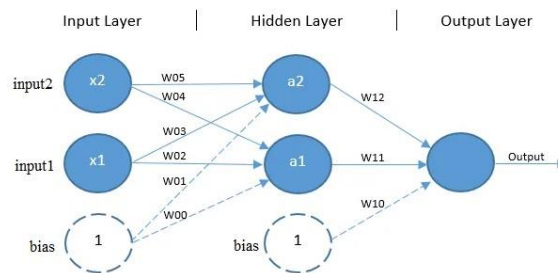


Figure 2.6: Multilayer perceptron (MLP)

In Figure 2.6, filled circles represent the nodes with real values. The dotted circles represent the bias with a constant value of one. Solid lines represent the connection



between nodes. Dotted lines connect bias value to the nodes. Nodes labeled by  $x_i$  for input values for input nodes in input layer and  $a_i$  for activation node in hidden layer where  $i$  is the number of a node. Arcs connect nodes from one layer to other and labelled by  $w_{ij}$ .  $w_{ij}$  is weight for each connection where  $i$  represents the layer number and  $j$  represents the line number. Output layer has one node with the output value.

MLP learning step starts with the input layer and forward data propagation to the output layer. This is “forward propagation” process. Calculate the error (the difference between the current and desired output) and need to minimize the error. The next step is to find derivative for each weight in the network, and update the model. Repeat these steps to learn ideal weights same as in single perceptron. Now model is created. Finally, to obtain the expected class labels for the test data, the output is taken through the model.

Forward Propagation: Each node-to-node link is associated with a weight. The weight of the link from the  $j$ -th hidden nodes to the  $i$ -th output nodes is denoted as  $w_{ji}$ , and the weight of the link from the  $k$ -th value of hidden nodes ( $a_k$ ) to the  $j$ -th hidden node is introduced as  $w_{kj}$ . The inputs multiplied by their respective weights are summed ( $\sum_k w_{kj}x_k$ ), and then mapped to the output via the transfer function. The transfer function can be one of several different functions below, depending on which interval we want to map, for example interval  $(-1,1]$ . The transfer functions for  $x$  value are listed below.

The sigmoid functions:

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \in (-1,1)$$

output is in range (-1,1). or

$$f(x) = (1 + e^{-x})^{-1} \in (0,1)$$

and ReLU function:

$$f(x) = x^+ = \max(0, x) \in [0, \infty)$$

where  $x$  is a real number, and  $e = 2.71828$  is Euler's number.

The  $i$ -th output node then receives the weighted sum of the values coming from the hidden nodes and returns to the activation function. This is how the  $i$ -th output is obtained. Two-layer perceptron calculates the following formula where  $f$  is the sigmoid transfer function,  $w_{kj}$  and  $w_{ji}$  are the weights of hidden layer and output layer and  $x_k$  are the input values.

$$y_i = f\left(\sum_j w_{ji} f\left(\sum_k w_{kj} \cdot x_k\right)\right)$$

Backpropagation: it is a short form for "backward propagation of errors." The principle of the backpropagation approach is to model a given function by modifying internal weightings of input signals to produce an expected output signal. The system is trained using a supervised learning method, where the error between the system's output and a known expected output is presented to the system and used to modify its internal state.

How MLP works?

1. Inputs  $x_i$ , arrive through the preconnected path
2. Input is modelled using real weights  $w_i$ . The weights are usually randomly selected.
3. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.

4. Calculate the error in the outputs

$$\text{Error}_B = \text{Actual Output } f(x) - \text{Desired Output } d(x)$$

5. Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased (Backpropagation).

Keep repeating the process until the desired output is achieved.

## 2.2 Single and multiple-node classifier ensembles

### 2.2.1 Single-node Bagging-like ensembles

In machine learning, the classifier ensemble learning [12] methods apply multiple learning algorithms to increase predictive performance.

Bootstrap aggregating (bagging) [14] is a simple ensemble of classifiers which combines predictions of multiple machine learning algorithms to make more accurate final result than any individual classifier. It executes on a single node. Bagging generates  $m$  new training datasets by sampling from training dataset. These new training datasets are named "bootstrap samples". Bootstrap samples are randomly selected with replacement, and the number of records in each bootstrap sample is equal to the amount of data in the training data set. A separate different training algorithm trains subsamples. After training,  $m$  models create. For illustration, let the training dataset have 10 records:  $TD = \{A, B, C, D, E, F, G, I, J, K\}$ . Bagging using random sampling with replacement creates 3 training datasets,  $TD1 = \{D, K, D, D, K, A, F, E, B, I\}$ ,  $TD2 = \{A, B, F, K, C, K, I, A, G, J\}$ ,  $TD3 = \{G, J, J, J, J, K, B, E, F, C\}$  to train 3 different classifiers in the ensemble. For example,  $TD1$  is fed to CART learning algorithm to train first model, and  $TD2$  and  $TD3$  are fed to MLP to train model2 and model3. And then the results from each trained model are aggregated in the form of voting (for classification) or averaging

(for regression). Aggregation for classification is made in the way that the vote-based ensemble is created with each classification in the ensemble to predict the class of new sample data, the class is selected by a majority of votes as the final ensemble prediction. For example, Figure 2.1 shows model1 predict label class cross for t1 and circle for t2 and cross for t3. Model2 predict cross for t1 and t2, circle for t3. Model3 predict cross for t1 and t3 and circle for t2. Therefore vote-based ensemble predict t1 region as cross because all the models predict it as cross and predict t2 region as circle because two models predict this region as circle and so on for region t3. Therefore, ensemble classifier predict t1 as cross, t2 as circle and finally t3 as cross.

Aggregation for regression, average of class labels is selected as the final prediction of ensemble. For example in figure 2.3, model1 predict  $\{(x < 2, t4), (2 < x < 4, t5), (4 < x < 6, t6), (x > 6, t7)\}$  and model2 predict  $\{(x < 3, t4), (3 < x < 4, t5), (4 < x < 7, t6), (x > 7, t7)\}$  and model3 predict  $\{(x < 3, t4), (3 < x < 6, t5), (6 < x < 7, t6), (x > 7, t7)\}$ . Ensemble classifier calculate average of x in each region and create final model as  $\{(x < 2.6, t4), (2.6 < x < 4.6, t5), (4.6 < x < 6.6, t6), (x > 6.6, t7)\}$ .

Figure 2.7 shows the steps of training input dataset by ensemble classifier method (bagging) in single node approach. First, the input dataset is partitioned into training and testing dataset. Bootstrap samples of records are selected from training dataset. In bagging ensemble method, there are several classifiers trained on the bootstrap samples separately to create model. A model is a file that has been trained to organize certain types of patterns to make prediction. The models are combined to create final ensemble classifier; this is “Aggregating” step. And finally in the testing step, final ensemble classifier or final model can predict the class label of the new

data and it can predict test data. The label of the region which the new data is placed is the prediction of new data.

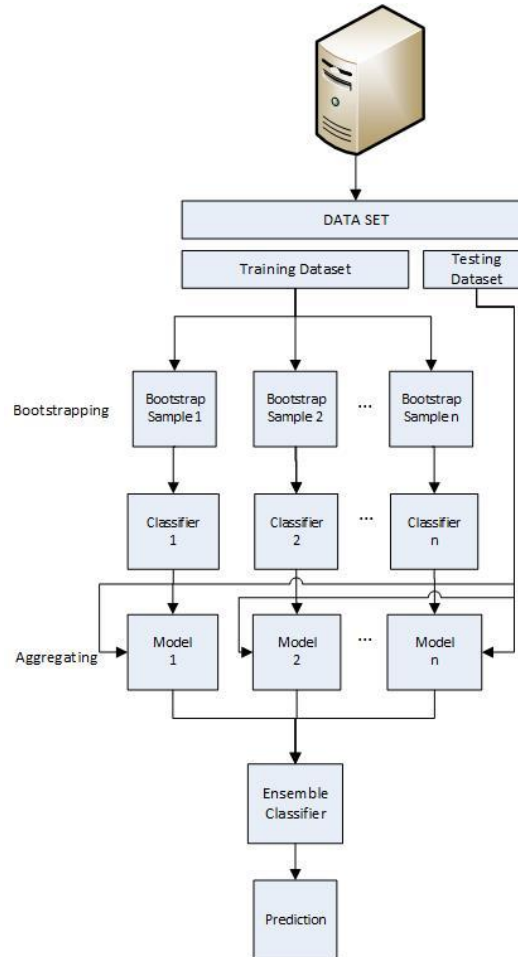


Figure 2.7: Single-node Bagging ensemble classifier schema

In the case working with big data, as the size of dataset is too large for the memory to match, it is impractical to train data using single-node method to create several bootstrap samples of sizes which are equal to the initial dataset. So the Bagging-like approach is suggested to solve this issue. In next section, we will examine methods of dataset partitioning and Bagging-like approach.

### 2.2.2 Multi-node Bagging-like ensembles

Bagging-like is an approach to creating a group of  $n$  classifiers from the partitioning of training dataset into  $n$  part. Each of these parts has size  $(1/n)$  which will fit to the memory, while in bagging approach each partition has size  $(n)$  equal to the original training dataset. Bagging-like has different methods for partitioning training dataset into disjoint subsets. These methods are disjoint partition (D), Small bags (SB), no replication Small bags (NRSB) and disjoint bags (DB) which will describe in next section in detail [Section 2.3].

Training on these disjoint partitions are done in distributed way in multiple node. In multi node classifier, training dataset is trained across many computing systems through network in parallel. Client-server architecture use to train multiple classifiers in multi systems.

In Figure 2.8, you can see the multi-node classifier ensemble schema. This schema has three steps:

Step 1. First step runs in server side and is involved partitioning dataset into training and testing dataset. The training dataset partitioned into smaller disjoint set by four different methods of Bagging-like partitioning (D/DB/SB/NRSB) [Section 2.3]. These disjoint datasets transfer to the client's side.

Step 2. Second step run on clients in parallel. Each disjoint dataset is settled in separate client and trained by distributed classifiers on any distributed frameworks (clients) for learning them in parallel way and extracting  $N$  models. Models will transfer to the server.

Step 3. In this step, all models combined and create final ensemble classifier, aggregating phase same as in normal bagging approach.

All these models will join to the server side for voting (classification) or averaging (regression) purpose. The ensemble's final prediction use to predict new data classes.

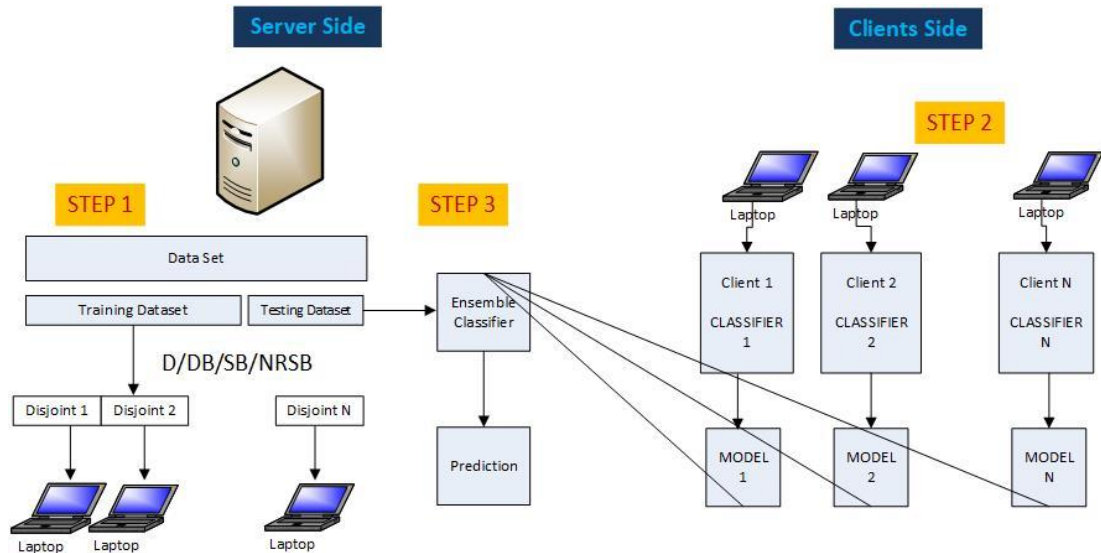


Figure 2.8: Multi-node Bagging-like classifier ensemble schema

### 2.3 Partitioning methods for Bagging-like ensembles

Chawla et al [2] suggested four Bagging-like ways of dataset partitioning.

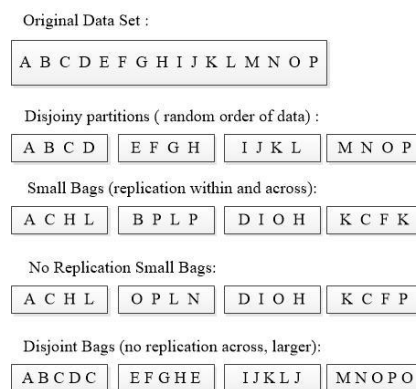


Figure 2.9: Four Bagging-like strategies: D, SB, NRSB, DB [2]

Figure 2.9 illustrates four Bagging-like partitioning methods [2]. They are:

- Disjoint partitioning (D): partition data into parts (bags) with random selection of data. The bags have not repeated records within or across the bags.
- Small bags (SB): select data randomly with replacement, so then there may be repeated records within and across the bags.
- In no replication Small bags (NRSB), record selection within bags is random without replacement, but across bags allows replacement.
- The last approach is disjoint bags (DB) choosing data across bags with replacement, and it has extra records so bags are larger than for three other methods, D, SB, NRSB

Details of the methods are considered below

### 2.3.1 Disjoint partitioning

original data set:

A B C D E F G H I J K L M N O P

Disjoint partitions (random order of data)

A B C D E F G H I J K L M N O P D

Figure 2.10: Disjoint partitioning method D [2]

Disjoint partitioning is a method to partition input dataset into N bags, and records are randomly selected without replacement.

Having a database, DB, with number of records, R, and given the number of partitions, N. Records of DB are read one by one and will be distributed in disjoint partitions one by one in sequence, so the union of disjoint sets (bags) is equal the



input dataset. Each letter in the bags refers to class labels. The approach D is illustrated by Figure 2.10.

Two sets are disjoint if they have no common elements. equivalently; two disjoint sets (bags) are sets whose intersection is empty. It is necessary splitting  $S$  into  $N$  partitions,  $P_i$ , such that  $\bigcup_{i=1}^N P_i = S$  and  $P_i \cap P_j = \emptyset$ . For example in Fig 2.2,  $N=4$  bags created are:  $S=\{A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P\}$ ,

$P_1= \{A, B, C, D\}$ ,  $P_2= \{E, F, G, H\}$ ,  $P_3= \{I, J, K, L\}$ , and  $P_4= \{M, N, O, P\}$ . The partitions obtained are expected having approximately the same number of records.

For disjoint partitions with random arrangement of data, first, disorder the DB by using random number generator. Then, the records are read one by one and added to the disjoint sets (bags). It should be emphasized that if the number of records in input file is not divisible by number of partitions, the number of records in the bags will not be equal. A pseudocode of D bagging is given below:

*D bagging{*

**Inputs:** *DB array, data file;*

*N integer, number of partitions (bags);*

**Outputs:** *P [N] array of partitions*

*L=len(DB)*

*/\* length size of input dataset*

*randomdf=reorder\_randomly(DB)*

*/\* generate input file with random*

*/\* records*

*while not end of randomdf do*

*For i=1 to N do*

*rec= read (randomdf)/\* read one record from random datafile*

*P[i]= P[i] + rec /\* add record to each P[i] bag*

*endfor*

*endwhile*

*}*

The sample of input and output of the program is given below:

**Sample of Program:**

```

DB = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q'] /* input data
L = 17, N=4
Randomdf = ['b', 'h', 'f', 'd', 'o', 'p', 'g', 'j', 'i', 'c', 'a', 'm', 'e', 'k', 'l', 'n', 'q']

i = 1 → P[1] = ['b'], i = 2 → P[2]=['h'], i = 3 → P[3]=['f'], i = 4 → P[4]=['d']

i = 1 → P[1] = ['b', 'o'], i = 2 → P[2]=['h', 'p'], i = 3 → P[3]=['f', 'g'], i = 4 → P[4]=['d', 'j']

i = 1 → P[1] = ['b', 'o', 'i'], i = 2 → P[2]=['h', 'p', 'c'], i = 3 → P[3]=['f', 'g', 'a'], i = 4 → P[4]=['d', 'j', 'm']

i = 1 → P[1] = ['b', 'o', 'i', 'e'], i = 2 → P[2]=['h', 'p', 'c', 'k'], i = 3 → P[3]=['f', 'g', 'a', 'l'],
i = 4 → P[4]=['d', 'j', 'm', 'n']

i = 1 → P[1] = ['b', 'o', 'i', 'e', 'q']

p= [['b', 'o', 'i', 'e', 'q'], ['h', 'p', 'c', 'k'], ['f', 'g', 'a', 'l'], ['d', 'j', 'm', 'n']] /* output lists

```

**2.3.2 Small bags partitioning**

small bags (SB) partitioning from [2, p. 458] is same as disjoint partition except that records are randomly selected with replacement so replicated records are within and across the dataset partitions. The union of these datasets is not equal to the input dataset.

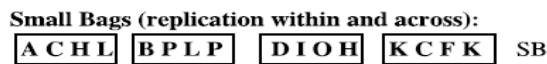


Figure 2.11: An illustration of small bags, SB [2]

For achieving partitions in random order of data with replication within and across the partitions, first check if the number of records is divisible by N, or not. If it is divisible, random records select as the number of integer value of quotient. If it is not divisible, integer value of quotient plus one record is selected. The relevant pseudocode is given below. Ceil function maps x to the smallest number greater than

or equal to  $x$ . In the case of divisibility, all partitions have the same number of records.

```

SB bagging {
Inputs: DB array, data file;
           N integer, number of partitions;
Outputs: P [N] array of partitions;

L = len(DB)
ceil(x)={x, if x integer, the minimal integer not less than x}
k= ceil(L/N)
For i=1 to N do
    randomdf = select_randomly_with_replacement(DB,k)
                /*select random samples in size of k with replacement
    P[i]= randomdf
endfor
} //end of SB bagging

```

Example is given below:

```
DB = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r'] /* input data
```

N = 4 and L= 18, k=5

```
i=1 → p[1]:['a', 'n', 'i', 'j', 'q'] /* output lists
```

```
i=2 → p[2]:['r', 'f', 'g', 'g', 'l']
```

```
i=3 → p[3]:['l', 'j', 'a', 'm', 'b']
```

```
i=4 → p[4]:['e', 'h', 'c', 'g', 'd']
```

Above sample shows, letter 'a' is repeated across partitions(1,3) and letter 'g' is repeated within and across partitions 2 and 4.

### 2.3.3 No-replication small bags partitioning

No Replication small bags (NRSB), is like small bags, but each partition is created by random sampling without replacement, so records are not duplicated in partitions, but may repeat across partitions. So, intersection of partition sets may not be empty,

$P_i \cap P_j \neq \emptyset$ .

original data set:

A B C D E F G H I J K L M N O P

No Replication Small Bags:

A C H L O P L N D I O H K C F P NRSB

Figure 2.12: An illustration of No-replication small bags, NRSB [2]

For NRSB in figure 2.12, first check the divisibility of input record numbers (L) to the number of disjoint partition (N) same as SB. The algorithm selects random records in the length of k without replacement in each partition subset separately. The pseudocode follows.

*NRSB bagging{*

*Inputs: DB: the dataset for partitioning;*

*N: integer number of subsets;*

*Output: P [N] array of partitions*

*L = len(DB)*

*ceil(x)={x, if x integer, the minimal integer not less than x}*

*k= ceil(L / N)*

*For i=1 to N do*

*randomdf = select\_randomly\_without\_replacement(DB,k)*

*/\*select random samples in size of k without replacement*

*P[i]= randomdf*

*endfor*

*//end of NRSB bagging*

In the following you can see the sample of input and output of the program.

### Sample of Program:

*DB = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q'] /\* input data*

*L=17, N=4, k=5*

*i=1 → p[1]: ['a', 'f', 'n', 'k', 'g'] /\* output lists*

*i=2 → p[2]: ['f', 'e', 'd', 'c', 'i']*

*i=3 → p[3]: ['h', 'm', 'f', 'e', 'i']*

*i=4 → p[4]: ['b', 'l', 'j', 'e', 'q']*

Above sample shows, letter 'f' is repeated across partitions(1,2,3) and not repeated within partitions.

### 2.3.4 Disjoint bags partitioning

The fourth approach is disjoint bags (DB) illustrated by Figure. 2.13. Each partition is created by random with replacement but in larger size than the first three approaches so the union of bags is all of the original data, and some data is replicated.

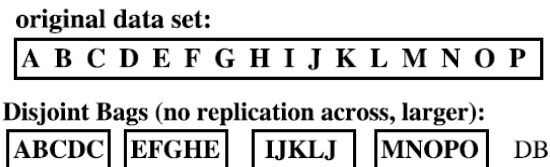


Figure 2.13: An illustration of disjoint bags, DB [2]

For this approach, firstly, change the order of records by creating random file. Then, create the random partition set without replacement. Then, choose an element in each partition set as a candidate for appending to the same partition set. In this case, each partition is larger than before with repeated element. The relevant pseudocode is as follows.

```

DB bagging{
  Inputs: DB array, data file; N integer, number of partitions;
  Outputs: P [N] array of partitions;
  L=len(DB)                                     /* length size of input dataset
  randomdf = reorder_randomly(DB)
                                                /*select random sample without replacement from DB

  while not end of randomdf do
    for i=1 to N do
      rec= read(randomdf) /* read one record from random datafile
      P[i]= P[i] + rec    /* add record to each P[i] bag
    endfor
  endwhile

```

```

for i=1 to N do
    S = select_randomly(P[i],1)      /*select randomly one record from P[i]
    P[i]= P[i] + S
endfor
} //end of DB bagging

```

Example of DB work is given below:

### Sample Output of Program:

```

DB = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q']      /* input data
L=17, N=4

```

```

Randomdf = ['b', 'h', 'f', 'd', 'o', 'p', 'g', 'j', 'i', 'c', 'q', 'a', 'm', 'e', 'k', 'l', 'n']

```

```

i = 1 → P[1] = ['b'], i = 2 → P[2]=['h'], i = 3 → P[3]=['f'], i = 4 → P[4]=['d']

```

```

i = 1 → P[1] = ['b', 'o'], i = 2 → P[2]=['h', 'p'], i = 3 → P[3]=['f', 'g'], i = 4 → P[4]=['d', 'j']

```

```

i = 1 → P[1] = ['b', 'o', 'i'], i = 2 → P[2]=['h', 'p', 'c'], i = 3 → P[3]=['f', 'g', 'q'], i = 4 → P[4]=['d', 'j', 'a']

```

```

i = 1 → P[1] = ['b', 'o', 'i', 'm'], i = 2 → P[2]=['h', 'p', 'c', 'e'], i = 3 → P[3]=['f', 'g', 'q', 'k'],

```

```

    i = 4 → P[4]=['d', 'j', 'a', 'l']

```

```

i = 1 → P[1] = ['b', 'o', 'i', 'm', 'n']

```

```

i = 1 → S='m', P[1] = ['b', 'o', 'i', 'm', 'n', 'm']

```

```

i = 2 → S='h', P[2] = ['h', 'p', 'c', 'e', 'h']

```

```

i = 3 → S='q', P[3] = ['f', 'g', 'q', 'k', 'q']

```

```

i = 4 → S='l', P[4] = ['d', 'j', 'a', 'l', 'l']

```

## 2.4 The label-aware distributed ensemble learning (LADEL)

Khalifa et al. [3] work on the other model to prepare training dataset, named “LADEL”. Label\_aware distributed ensemble learning (LADEL) is a method to distribute data in the distributed systems [5] parallel which increase accuracy in big training dataset. The strong point of this new technique is capability of working in any classification algorithms, in any platform without any changing in program and faster training.

Generally, there are two types of perspectives to solve machine learning problems. One is to focus on speed and gain faster training time, and the other is to gain better accuracy. For this purpose, the following steps have been taken to obtain the results for decision making. First approach is distributed sequential single-node algorithms. Second, Distributed Stratified Sampling (DSS) algorithm is used on large dataset.

DSS is a technique that input data is broken down into small groups (stratum), and then random sampling is done from each of this stratum independently. These random records gather in a file used as an input for classification algorithm (Figure 2.14). Big Data should be stratified and distributed to shared-nothing [13] architecture systems. Standard Stratified sampling work on broken down training dataset into small size dataset without any special policy on selecting samples. In DSS, stratum is created based on the class labels. Therefore, random sampling forced strata to train in all training dataset class labels. Finally, LADEL model is used to train data in any framework.

LADEL needs three inputs: -the input dataset, -the classification method and - the number of desired parallel dataset partitions to construct a trained classifier.

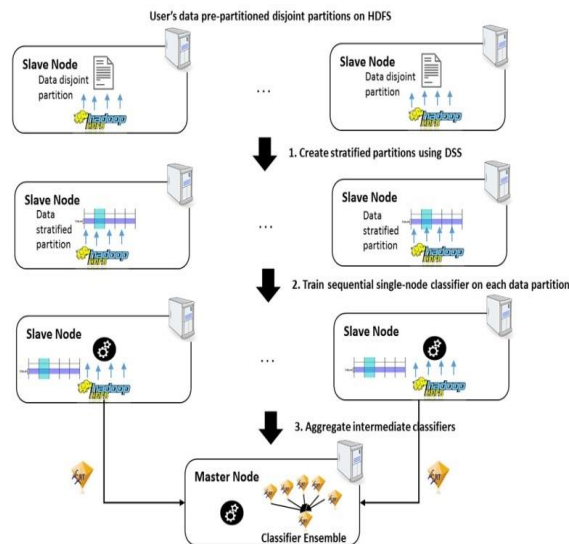


Figure 2.14: LADEL execution overview [3]

Figure 2.15 shows three steps of LADEL execution model:

- 1- Create stratified partitions using Distributed Stratified sampling (DSS).
- 2- Train sequential single-node classifiers on each data partition.

3- Aggregate classifiers by using majority vote method.

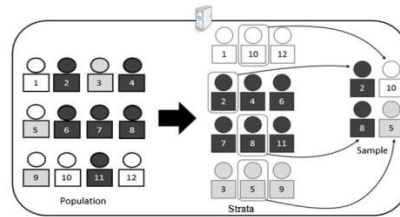


Figure 2.15: single-node stratified sampling [3]

The labels and number of records in each class is distributed balanced. Figure 2.15 displays two steps for making stratified in sequential single node.

Step 1 - Create groups of records in separate strata by class label.

Step 2 - Select samples from each stratum randomly.

Created categories include all the class labels.

In the single-node memory, big data does not fit. Simple random sample selection is used to create distributed filesystem and transfer them to distributed servers. These disjoint partitions does not include all class labels and poorly trained ensemble classifier. Figure 2.16 displays distributed stratified sampling (DSS) model to create a disjoint partitions with all class labels included. DSS has two phases, Local and Global.

Local phase is executed in each individual slave node separately in parallel. It consists of two steps;

Step 1- Create groups of records in separate strata by class label. Therefore, the number of strata is equal to the number of class labels.

Step 2- Round-robin selector runs in parallel on each system separately on each stratum (including one class label). In each round-robin cycle, a record is selected



and stored in the current slave node and the second selected record is sent to second slave node and so to the end. Thus, all the class labels distribute in slave nodes.

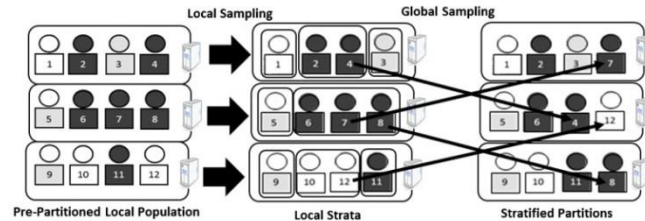


Figure 2.16: Distributed stratified sampling (DSS) [3]

For example, as you can see in Figure 2.16, local sampling creates local strata on each node. The upper node has three class labels (white, grey, black), so it has three local strata. White and grey strata each have one record and black strata have two records. Such grouping will be done on each node in parallel. In the next step, Round-robin selector runs in each node separately in parallel. In the upper node, record labeled number 1 remains in the same node. This stratum has not any other record, so selector goes to the next strata. In next stratum, record label number 2 remains in the same node, record number 4 will be sent to the next node. In the next stratum, record number 3, remains in the same node. As you can see, in the middle node, there is a black stratum with three records numbers by 6, 7, and 8. Record number 6 remains in the same node, number 7 is sent to the node above, and record number 8 is sent to the node below. At the end, you can see each node has records including all class labels.

Global phase runs in each individual slave node after the local phase. It integrates the records sent from local phase with its own records. These new strata have all class labels of original big dataset inside and are ready to be used by a classifier.

By increasing number of strata and having parallel process, execution time will be declined. Practical experiments show that using sequential single-node stratified sampling needs more execution time than DSS to aggregate the distributed partitions.

## 2.5 LADEL evaluation

In [3], three approaches are used to compare the training time and accuracy on decision tree, neural network with or without LADEL technique. First, it uses LADEL on distributed nodes and a sequential single-node data mining library. Second, it uses voted-based distributed ensemble learning on distributed sequential single node classification algorithms, and third, it uses original sequential single-node classification algorithm on a single machine trained on the whole training data. The past empirical experiments show that neural network algorithms have better training time and accuracy on single-node training in comparison to other solutions. Multilayer perceptron has been chosen for testing the distributed systems. The second algorithm chosen from decision tree methods is Hoeffding Tree. The HIGGS dataset with almost 7 million training records and 2 class labels is used.

Figure 2.14 shows the effect of increasing the number of classifiers on the amount of training time in distributed ensemble compared to a sequential single node. A distributed ensemble is modelled by training classifiers (*Hoeffding Tree, multilayer perceptron*) based on disjoint partitions data divided into separate partitions in multiple slave nodes. The trained models are sent to the master node to participate in the vote-based classification. Figure 2.17 demonstrates that training time is decreased by increasing number of slave-nodes and this result is independent of the training algorithm. In the best case with 100 nodes, almost 5 % and in the worst case with 10

nodes, almost 29% of the time required to train dataset in a single-node classifier for total records in data set.

Figure 2.18 displays the effect of increasing number of classifiers on the scoring time. It shows that scoring time increases when using ensembles in comparison of using single classifier on single node. So by adding the number of classifiers may sacrifice training time to better predict accuracy.

Figure 2.19 displays the effect of distributing the scoring time operation across 10 nodes. It shows reduction on scoring time than that of a single classifier running on a single machine. Scoring time in distributed systems on 10 machines is 18 times faster than single node.

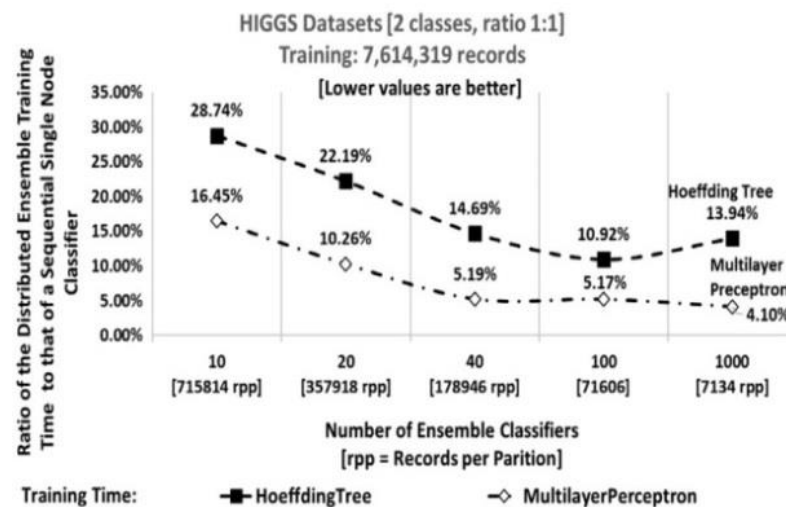


Figure 2.17: Ratio of the training time of distributed ensemble to a sequential single-node classifier [3]

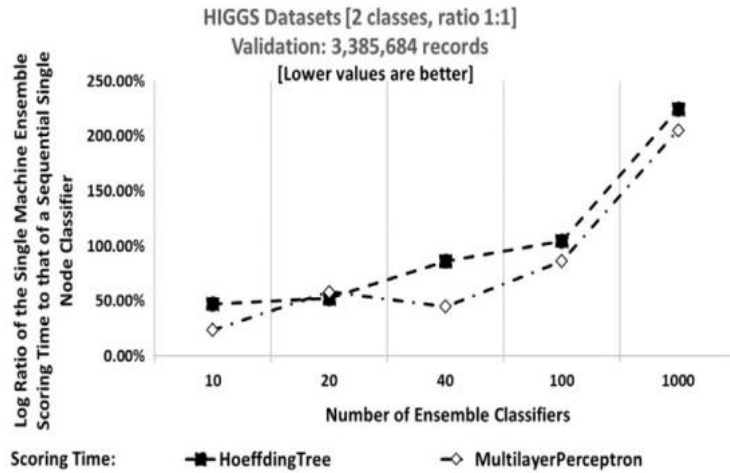


Figure 2.18: Ratio of the scoring time of single machine ensemble to a sequential single-node classifier [3]

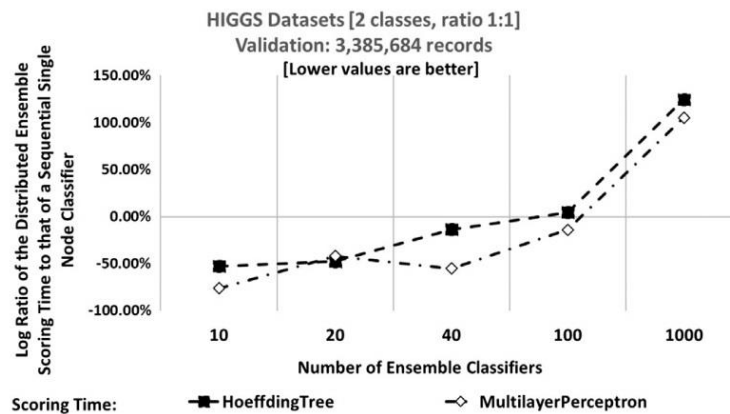


Figure 2.19: Ratio of distributed scoring time on 10 machines of an ensemble to that of a single classifier on a single machine [3]

In addition to timing score analysis, they also focused on prediction accuracy analysis by using LADEL ensembles. LADEL gains better precision than single-node classifier trained on entire training dataset.

## 2.6 Parallel computing

Traditionally, software has been written for *serial* computation. It means, a problem is broken into a series of instructions, and instructions are executed sequentially one after another on a single processor, and, of course, only one instruction executes at any moment in time. We usually need a more powerful processor and big size of

memory to process the Big Data. Mostly the size of a dataset does not fit to memory, so that is impractical to run program just on a single system. Sometimes the problem is not memory but processor. Even if the program can run in single node system, it will be very time consuming. Using *parallel computing* seems to be the right solution.

In the simplest sense, parallel computing is the simultaneous use of multiple computing resources to solve a computational problem easier and faster. Using parallel computers through networks connects multiple stand-alone computers to make larger parallel computer clusters to solve larger complex problems and provide concurrency.

Parallel computers can be classified according to the number of instruction streams and number of data streams into four categories. Single-instruction single-data (SISD), single-Instruction multiple-data (SIMD), multiple-instruction single-data (MISD) and multiple-instruction single-data (MIMD). Various parallel solutions make three different types of parallelism: Shared memory systems, Distributed systems. Shared memory systems with multiple processing units are attached to a single memory (Figure 2.19). Distributed systems consist of many computer units (Figure 2.20), each with its own processing units and physical memory that are connected to fast connection networks. Regarding the three types of parallelism, there are three different approaches for parallel programming: threads model for shared memory systems, message passing model for distributed memory system. In this project we used distributed memory topology to use memory resources in all shared nodes to increase scale of memory and ability of fast computing during long lasting process on big data set. Besides, using the processing power of several

machines simultaneously increases the overall processing power and reduces the processing time. These are the biggest advantages of using parallel processing systems that we use in this project.

In Ubuntu operating system NFS (Network File Sharing) service, all the clients can exchange and access the input data and source code of program to execute it in their own system in parallel. For example, if we have two laptops, connection between these two systems is possible through the connection to the modem-router. Now it is possible to share storage of one laptop (master node) with other laptop (client node) by NFS service. For this purpose, cloud folder is created in master node and is mounted in both laptops to access files through network and it consider as a shared folder. This folder physically used storage of master node but it is logically accessible by clients.

#### NFS Configuration:

On master node:

```
Sudo apt-get install nfs-kernel-server /*NFS server package installation  
Mkdir cloud /*create cloud folder  
Sudo vi /etc/exports /*cloud folder permanently shared  
/home/azi/cloud * (rw, sync, no_root_squash, no_subtree_check)  
Sudo exportfs -a  
Sudo service nfs-kernel-server restart
```

On client nodes:

```
sudo apt-get install nfs-common /* NFS client package installation  
mkdir cloud /* create cloud folder  
  
sudo mount -t nfs master:/home/azi/cloud ~/cloud  
/*mount cloud folder of master on cloud folder in client  
sudo service nfs-kernel-server restart /*restart nfs service
```

## 2.7 Client-server architecture

Client-server architecture [16] is a network computing model where one of the systems has a role of server and the rest systems connected to a central server over network. The server is responsible to produce services for clients through network. These services can include application access, storage; file sharing, processors and memory access. The clients' requests specific services or resources and the server can manage several clients' requests simultaneously and respond to their needs by preparing requested resources.

Client-server configuration needs to prepare network connection between at least three systems. Each of these servers are dedicated to static IP address and should be identified to all systems on the network by adding IP address and alias names to the *hosts* file.

```
Sudo apt install net-tools          /* installation of network tools

Sudo vi /etc/hosts                 /* assign alias names to the IPs
192.168.0.10 master asus
192.168.0.20 client sony
```

FTP Configuration:

```
sudo apt install vsftpd           /*FTP package installation
vi /etc/ftp
sudo vi /etc/vsftpd.conf          /*set FTP configuration parameters
anonymous_enable=NO
local_enable=YES
write_enable=YES
sudo service vsftpd restart      /* restart FTP service
```

## 2.8 Message passing interface (MPI)

The message passing interface (MPI) [17] is a library used to convey messages in parallel programming. Data transfer from the address space of one process to the other process through cooperative operations on each process. As it is known, processors can be local, that means they are housed in a single system or they operate in distributed systems. MPI is designed to be used in distributed systems. Regarding the usage of memory, MPI runs on different hardware platforms, including shared memory, distributed memory and hybrid memory.

In Figure 2.20, Distributed memory [18] is assigned to the system, which has multiprocessor, and each processor has his own memory separately.

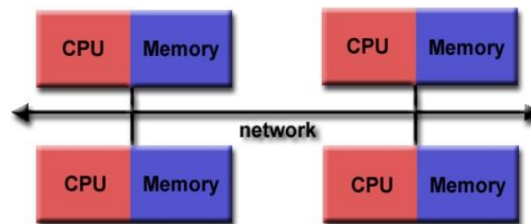


Figure 2.20: Distributed memory model [17]

Figure 2.21 shows shared memory [19] model. In this model, memory may be accessed by multiple program simultaneously and use by multi-processors.

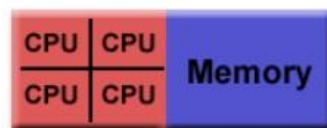


Figure 2.21: Shared memory model [19]



Hybrid memory model is a multiple processors use multiple memories. As you can see in Figure 2.22 four processors use each memory.

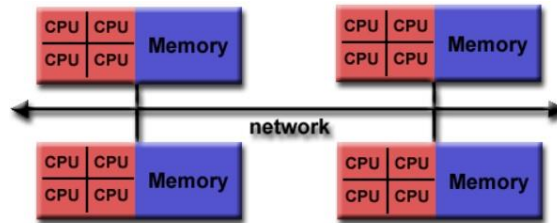


Figure 2.22: Hybrid memory model [17]

One of the major differences between these models are the data transfer cost. In shared memory systems, the time required to transfer data is the strong point of this method, as the inter-connection is much faster than distributed memory. However, the number of processors that can be used in parallel in distributed memory is much higher (more than eight) than the previous topology. Therefore, we used the second method of distributed memory in our project. This structure can reduce execution time. The communication speed in distributed memory depends on the internal network topology and the geographical distance of the communicated systems. The total execution time is the sum of computation and communication time. MPI is a middleware tool which runs tasks on separate processors using local memory and communicating with each other over a network. It helps us to run tasks in parallel, using more resources in parallel and save execution time.

## 2.9 MPI in Python

Python is a popular language for programming and excellent tool for developing parallel code today. Anaconda Distribution is an open source Python programming language distribution that can be used for large-scale data processing and has many tools and packages for data mining programming and machine learning purposes.

Parallel programming on distributed memory machines are using MPI in Python by *mpi4py* package. Being array-based in Python is an excellent advantage for developing parallel codes in small and medium size programs besides it is largely object oriented [20]. MS-MPI SDK is installed and configured in Ubuntu operating system in both laptops. For simplicity of later usage, it is recommended to add the execution path of MPI to the PATH environment variable in all nodes so easily can run *mpirun* command from any path you are in Ubuntu console command prompt. For simply run an MPI application, you can create the execution file or the Python version of your program and run *mpirun* command through console.it will run n copies of execution file in the run-time environment in a round-robin fashion by CPU slot. There are two purposes of running this command. One is running N copies of program in the current run-time environment and distributes them on the processors in the same system and the other way is to run N copies on different processors from other systems.

#### MPI Configuration:

Gcc (the C compiler on Ubuntu 18.04) installation prerequisite for openmpi:

<https://linuxconfig.org/how-to-install-gcc-the-c-compiler-on-ubuntu-18-04-bionic-beaver-linux>

```
sudo apt install gcc           /* gcc package installation
sudo apt install build-essential /* package installation
gcc --version                 /* gcc version checking
```

libopenmpi2:

*/\* openmpi packages installation*

```
sudo apt-get install openmpi-bin openmpi-common openssh-client openssh-
server libopenmpi2 libopenmpi-dev
```

mpi4py package installation in Anaconda Python V3 & PIP:

```
conda install mpi             /* mpi package installation
conda install -c anaconda mpi4py /* mpi4py package installation
sudo apt install python-pip   /* package installation
pip install mpi4py
conda install --name py3 pandas
```

*Mpi4py* allow programmers to run multiple processor computing systems. *Mpirun* command specifies the number of cores available and use in particular cluster. This library assigns unique rank to each processor and manages tasks by their ranks. Point to point communication prepares transmission process through processors by sending message from one side and receiving from the other processor. Each of the sending messages is labeled by tags, so we can trace them by these tags. MPI allow controlling in overlapping on communications by using non-blocking technique. Non-blocking always tests by completion function to be sure the requested operation has been completed. Using *isend* and *ireceive* command helps to implement non-blocking method.

For running n copies of program in multi-processor in parallel on single-node just need to run *mpirun* command like below format. As an example it will assign 8 copy of *hello.py* program through current path to 8 processors. (as we use Python source code of our application in Python version 3, it should be coded in the executable command otherwise parser is searching on Python version 2)

```
mpirun -np 8 python3 ./hello.py
```

The following command can be used to run the same program on parallel servers through network. The main server runs on 6 processors and the client runs on two processors, for a total of 8 copies of the program on 8 processors concurrently.

```
mpirun -n 8 -H master:6,client:2 python3 ./hello.py
```

## **2.10 Problem definition**

The problems of the study are as follows:

1. Design DELS using:

- 1.1. Partition methods: disjoint partitions (D), disjoint Bags (DB), small bags (SB), no-replication small bags (NRSB) and label-aware distributed ensemble learning (LADEL);
- 1.2. Classification and regression tree (CART) and multilayer perceptron (MLP) classifiers
- 1.3. Message Processing Interface (MPI) and the *mpi4py* package in Python V3 in Ubuntu TLS in single-node (not shared structure) and we implemented multi-node (client-server architecture) using Secure Shell (SSH) for secure data transmission over the LAN.
2. Test DELS on a network of two laptops with four and eight processors, with network connection prepared through the TP-Link wireless modem router.
3. Conduct experiments on DELS to evaluate the accuracy of various methods of data partitioning dependence on a single-node or multi-node systems, as well as the required runtime to implement them. The accuracy dependence on two classifiers CART and MLP.
4. For experiments on DELS, use KDD Cup 99 dataset [23] for comparison on Bagging-like methods and HIGGS dataset [15] for training and scoring time experiments.

## Chapter 3

# DESIGN, IMPLEMENTATION AND EXAMINATION OF DELS

In this chapter, we describe the design, implementation, and testing of DELS using partitioning methods: disjoint partitions (D), disjoint Bags (DB), small bags (SB), No Replication small bags (NRSB), and label-aware distributed ensemble learning (LADEL). The input dataset, the required number of partitions, and classification algorithm are three inputs to DELS. Two algorithms of classification, tree classification and regression (CART) and multilayer perceptron (MLP) are used in the system. DELS distributes the partitioned data sets and trains them in parallel to create models, and the trained models are eventually gathered to produce the final model. Distributed learning method allows Big Data management. The final model maintains the prediction accuracy and predicts the class label of new data through single-node or multi-node structure.

In this chapter, DELS design and architecture (Section 3.1), design and implementation of Input/ Output subsystem (Section 3.2), partitioning subsystem including LADEL model (Section 3.3), training subsystem (Section 3.4), testing subsystem (Section 3.5) and testing DELS in single-node and multi-node classifier (Section 3.6) are considered. In testing, first, focus is made on checking accuracy of partitioning methods (D/DB/SB/NRSB) and comparing these methods accuracy with CART classification algorithm and multi-layer perceptron, MLP classification

algorithm. These tests are made in single-node and multi-node with different number of partitions. Second, checking the prediction test-score of partitioning methods (D/DB/SB/NRSB) on multi-node with MLP algorithm. Finally, checking the accuracy and prediction test-score with LADEL in multi-node and compare the test-score and execution time on MLP and CART.

### 3.1. DELS design and architecture

DELS system architecture is represented in Figure 3.1. It has the following parts: Input/output Subsystem (IOS), Task partitioning, partitioning Subsystem, Training Subsystems, and Testing Subsystem.

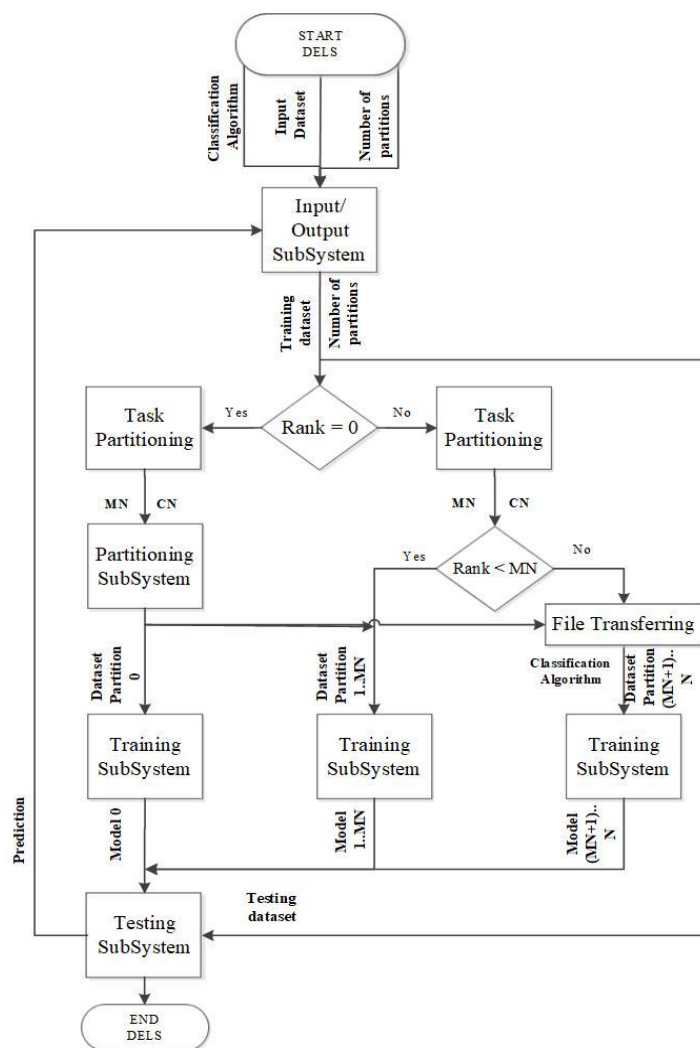


Figure 3.1: DELS architecture

DELS has three inputs, the input dataset, number of partitions and the classification algorithm. It works on input dataset which is in big size to divide data into smaller sub partitions by partitioning methods such as disjoint partitions (D), disjoint Bags (DB), small bags (SB), No Replication small bags (NRSB) and Label\_aware distributed ensemble learning (LADEL). The number of data partitions is determined by number of partitions. Parallel tasks are generated, distributed, and executed by MPI to perform parallel processing on each of these data sub partitions. Parallel tasks implement in single-node (not shared structure) and multi-node (Client-server architecture). In this experiment, CART or MLP classification algorithms are used. In DELS, learning process is performed on each of these separate tasks by CART and MLP classifiers. Input dataset is introduced in filename variable [Appendix A.3.1, Line 55]. The format of input dataset is comma separated text file which can be one of .csv or .txt format. A dataset consists of a series of records with multivariate features. Each record is defined as a series of attributes with comma separators, and the attributes value are integers. The KDD Cup 1999 is selected dataset for testing Bagging-like methods and HIGGS dataset is selected for testing training and scoring time in DELS. The KDD Cup dataset has a collection of records with 44 columns, and 44th column is considered as class label (full scheme is given in Appendix A.1.1, line 62-70, “names” array). In HIGGS dataset, the first 21 features (columns 2-22) are kinematic properties measured by the particle detectors in the accelerator and The first column is class column(full scheme is given in Appendix A.3.1, line 57-75, “names” array). Since the program is done to compare accuracy in the number of data partitions, the number of data partitions is stored in the array and processed in parallel. The number of partitions is kept in Partno array [Appendix A.3.1, Line 56].

The IOS subsystem [Appendix A.3.2], prepares training dataset and testing dataset from input dataset as an output of IOS. Training dataset is used in partitioning subsystem, and testing dataset is used in testing subsystem. These datasets preparation will be discussed in Subsection 3.2 in detail.

After preparing the training and testing datasets, the training datasets is used for learning. As mentioned in chapter 2, training to use a single big size dataset is impractical, as it does not fit in memory, so using Bagging-like approach to train the ensemble classifiers on disjoint partitions separate from the training dataset that can fit in memory is practical. Therefore if these partitions combined, will have the size of the original training dataset. This way all data is used in training to yield a better accuracy than if sampled. To address this challenge, we implement the Bagging-like approach to deliver distributed training to their classifier ensembles. Training data split into disjoint partitions by Bagging-like methods and then distributed and executed across several tasks, which is implemented by MPI. MPI is responsible for the simultaneous execution of tasks and distributing them across system(s) processors. Tasks can be distributed in a local system with multiple processors, single nodes, or in multiple processors belonging to multiple systems, multiple nodes.

In this experiment for distributing tasks through multi-node architecture, due to the lack of laboratory systems, we used two laptops with 8 and 4 processors. Since one of the systems has higher processing power, so we put two thirds of the parallel processing on the processors of this system and the other third is processed by another system. MPI assigns a unique rank to each of these tasks to distribute these tasks in parallel. We consider rank = 0 as “server task” and assign this task to the first



power laptop called “master”. In addition, master server executes two-thirds of the parallel tasks same as client systems tasks. The second laptop, called “client”, is responsible for performing one-third of the remaining tasks. For single-node architecture, all tasks will distribute in the processors of one laptop (usually master).

In Task partitioning [Appendix A.3.3, A.3.8], the number of master and client system tasks is calculated. MN shows number of master tasks and CN shows number of client tasks. As you can consider in the Figure 3.1, it has three branches. “Server task” is on the left, “master task” is in middle and “client task” is on the right.

“Server task” (rank = 0) is included:

- Data partitioning- partition the training dataset
- transfer the CN number of dataset partition(DP) to the client laptop
- train the first dataset partition(DP0), create model 0, waiting to receive all models prepared in other parallel tasks
- and finally, make prediction on testing dataset

“Master task” ( $0 < \text{rank} < \text{MN}$ ) is included:

- train the dataset partitions with the rank number of 1 to MN ( DP1.. MN )

“Client task” ( $\text{MN}+1 < \text{rank} < \text{N}$ ) is included:

- train the DP with the rank number of MN+1 to N ( DPMN+1 .. N ).

Partitioning subsystem [Appendix A.3.4] are used two inputs: training dataset and the number of partitions. Training dataset is divided into Partno smaller subset partitions. Each dataset partition (DP) is sent to a different task prepared by MPI for parallel processing. This stage of DELS is called “preprocessing”. Detail will be discussed in Subsection 3.3. In Training Subsystem [Appendix A.3.5, A.3.9, A.3.11],

a sequential classification algorithm is executed on each DP as training dataset in parallel to produce mathematical models, or classifiers. The number of classifiers is equal to the number of partitions determined in Partno array. In Testing Subsystem [Appendix A.3.6], the testing dataset, prepared in IOS and the prediction models from training subsystem are used as inputs. In this Subsystem, results of the models are joined by majority vote to select the final prediction. In the following, the implementation of DELS subsystems is described separately.

### **3.2 Design and implementation of IOS**

Figure 3.2 shows IOS starting with three inputs: data set, classification algorithm and desired parallel level or number of partitions. We consider the parallel level is in 2, 4, 6, 8, 10 and 12 processes. In machine learning techniques, input data needs to be split into two sections. One as a training part (usually 75%) and the second part is considered as testing part (usually 25%). Records are selected randomly. We consider two approaches to handle input data. One is consider as Bagging-Like methods (D, DB, SB and NRSB [Appendix A.3.2, Lines 76-94] and the other in LADEL format [Appendix A.4.2, Lines 81-137].

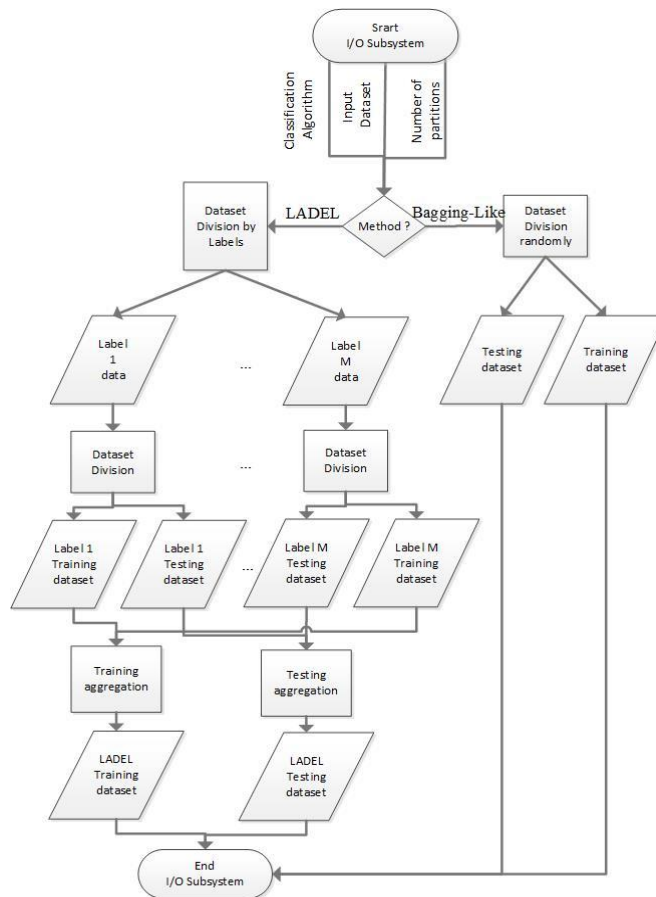


Figure 3.2: Input/output subsystem (IOS)

LADEL suggested us: all the labels should be in training dataset. To do this, records with the same class label are grouped and then stored in separate CSV files [Appendix A.4.2, Lines 81-97]. As the input dataset is in large size, it's ok to consider the 75% of input dataset records as training dataset and the rest 25% as testing dataset (training: testing ratio 75:25), but with all class labels included. We keep 75% of the records of each of these files in a separate file and the remaining 25% in another separate file (dataset division). Record selection is done randomly [Appendix A.4.2, Lines 106-123]. Next step, all files with 75% of label 1-M dataset are combined together for making LADEL training dataset (training aggregation) and all files with 25% of label 1-M dataset are combined together for making LADEL testing dataset (testing aggregation) [Appendix A.4.2, Lines 124-137]. Now we are

sure all the class labels are included in both data sets in the same ratio. LADEL training dataset is input for the next subsystem, partitioning subsystem and LADEL testing dataset is input for the testing subsystem.

In Bagging-like method, input dataset records are randomly divided into training dataset and testing dataset in a ratio of 75:25. [Appendix A.3.2, Lines 76-94].

### **3.3 Design and implementation of partitioning subsystem**

In this stage of DELS Process, the proper size inputs, which fit to the memory size of the system, is prepared. We use five different data division methods: 1-D as disjoint partition, 2-DB as disjoint bagging, 3-SB as small bags, and 4-NRSB as No Replication small bags and 5-LADEL as label\_aware distributed ensemble learning.

Figure 3.3 shows steps of partitioning subsystem for Bagging-like and LADEL. Training dataset is input of this subsystem. MPI is a utility that is responsible for performing simultaneous tasks. These tasks are distinguished by separate ranks. As it mentioned before, if the task rank is zero, it means the first executable task and it consider as the master task. This first task is responsible for data partitioning. The other ranks assign to the other parallel tasks for training purpose. In this subsystem (IOS), first should decide about dataset partitioning method between Bagging-like or LADEL methods.

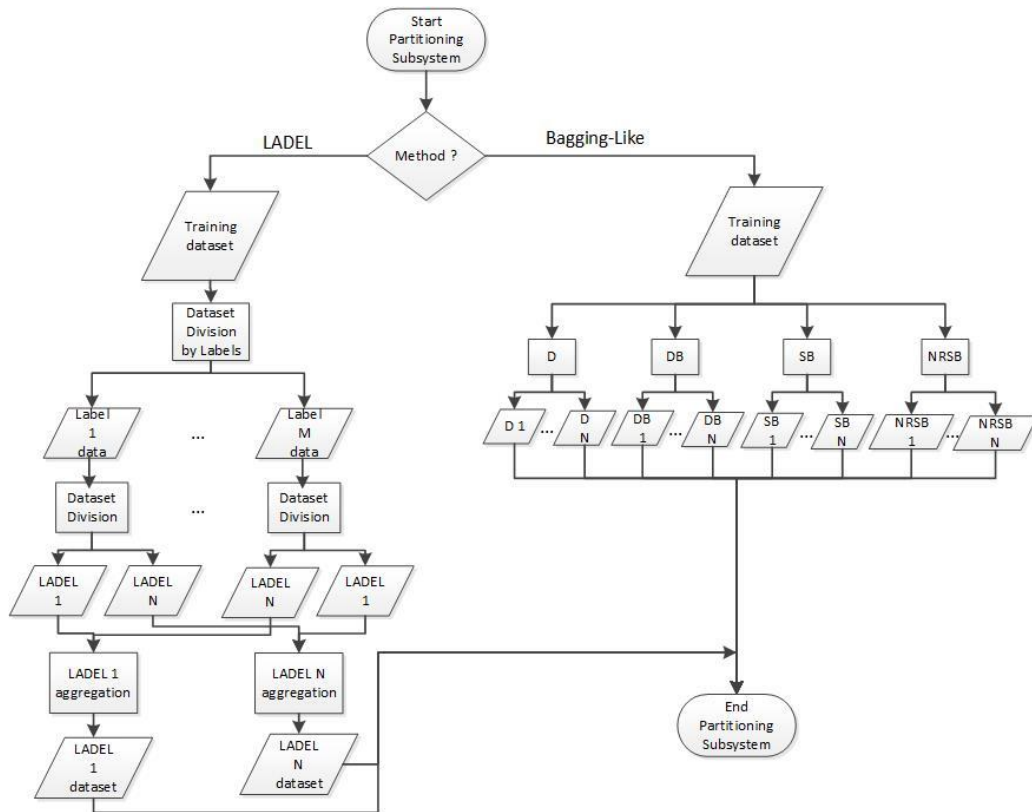


Figure 3.3: Partitioning subsystem

For Bagging-like approach, first main task open and read the training dataset [Appendix A.3.4]. Then partitioning in D/DB/SB/NRSB methods will do. Dataset segmentation is done in disjoint partitioning (D) [Appendix A.3.4, Lines 154-186], small bags (SB) [Appendix A.3.4, Lines 187-220], disjoint Bagging (DB) [Appendix A.3.4, Lines 221-251] and No replication small bags (NRSB) [Appendix A.3.4, Lines 252-289] in source code by using MLP classifier. All these process happened in rank=0 (first parallel task), other parallel tasks just wait until the first main task finished process. Number of segmentation is handled by the partno array. It has 2, 4, 6, 8, 10 and 12 value. After the data partitioning, a message is sent to the other tasks stating that the data is ready for further processing by other processors. The split and transfer time is recorded [Appendix A.3.4, Lines 290-311]. Details of these implementing data segmentation are described in chapter 2, section 2.3.

For LADEL, M is a number of labels in class column in input dataset and N is the number of data division, which keeps in the partno array. First, all the records of the same label in training dataset will separate and save in the CSV files [Appendix A.4.3, Lines 138-169] then each of these files divide into N and finally all the same division of label 1-M aggregate together [Appendix A.4.3, Lines 169-182]. Final datasets for sure, have all the labels in equal ratio.

### **3.4 Design and implementation of training subsystem**

In this subsystem, there are two classifier approaches for extracting model. One is CART and the other is MLP classifier in training subsystem, Figure 3.4. All segmented data begins to be learned in parallel by separate processors. As you know in this experiment, there are two approaches. First, accuracy checking and second, prediction checking.

Training section for accuracy checking is done in disjoint datasets (D) by MLP [Appendix A.1.3, Lines 238-275], in SB [Appendix A.1.3, Lines 276-310], in NRSB [Appendix A.1.3, Lines 311-346], in DB [Appendix A.1.3, Lines 347-380]. In this part, disjoint training dataset is read and using K-fold technique by 10-fold in MLP classifier. The mean accuracy of these 10 folds is extracted and saves in arrays (meanD, meanDB, meanSB and meanNRSB). Standard deviation and standard error of mean (SEM) are also save in a different arrays (stdD and semD) for future comparison [Appendix A.1.3, Lines 260-264]. First task(rank=0) waits to receive a signal from other tasks that the training has been completed, and then stores all the accuracy and error results in an array [Appendix A.1.4]. All the results will save in Excel file [Appendix A.1.5] and plot them [Appendix A.1.6]. In each matplots, you

can see the accuracy of each DSS method plus and minus standard error of mean (SEM) in each number of dataset partitions.

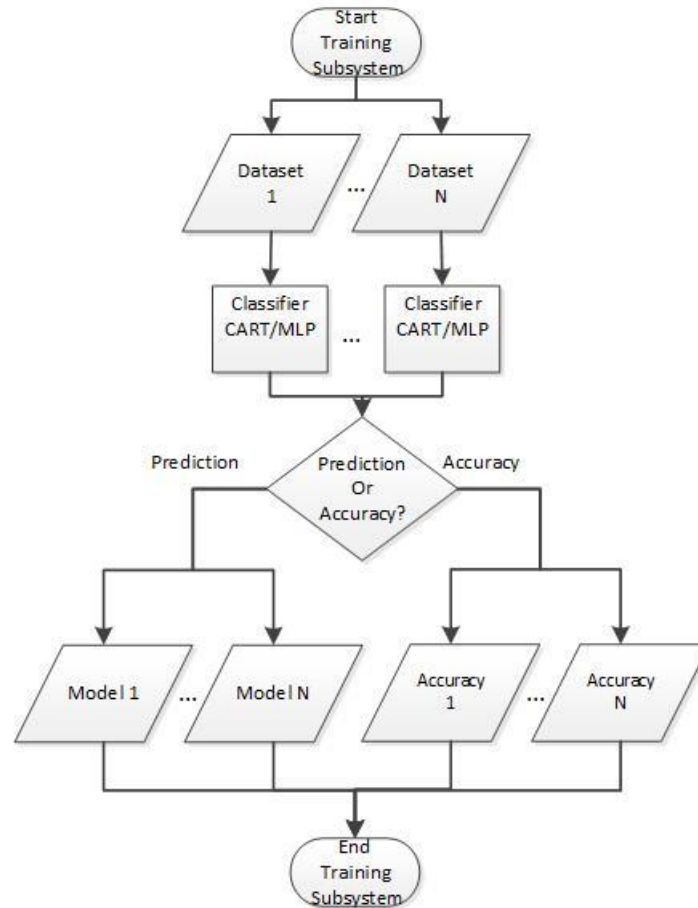


Figure 3.4: Training subsystem

Training section for prediction checking is done in [Appendix A.3.5, A.3.9, A.3.10], same as training section for accuracy checking, with the difference that the output of this subset is a learning model [Appendix A.3.5, Line 329].

Training dataset is done in 3 different rank type: First task by rank=0 is done in first laptop (ASUS VivoBook S14 with four cores, 1.8 GHz, 1992 MHz, 8 Logical Processor(s), 8 GB RAM, Page File Space 2.5 GB and 256 GB SSD Disk) as role of server [Appendix A.3.5]. Second, the two-third of total tasks in ASUS laptop as a

role of client [Appendix A.3.11]. The rest one-third of tasks in second laptop (Desktop SONY VAIO with two cores ,2.4 GHz, 2400 MHz, 4 Logical Processors, 4 GB RAM, Page File Space 1.88GB and 500 GB disk space) is coded [Appendix A.3.9]. Training subsystem for LADEL also has the same subsections in Appendix A.4.

### **3.5 Design and implementation of testing subsystem**

In this section of DELS architecture, the results of learning  $N$  input datasets are  $N$  models, Figure 3.5. The entire extracted model should gather in the server for majority voting decision-making. In majority voting stage, the prediction accuracy of the entire extracted models is compared and the highest rank will be selected for the final step in the application. As we use NFS for using shared storage, there is CLOUD folder as NFS file system. Both laptops can access to this area of storage in server. At the end of learning process, models will save in CLOUD folder, so in this phase no need to transfer back again the results by FTP to the server system. MPI is responsible to send a signal to the server after successful saving models in shared storage from all clients. The testing subsystem will do major voting and the results is one model. This model is used to test the testing dataset and extracting prediction test-score. Majority voting in multi-node by MLP for Bagging-like ensembles is in [Appendix A.3.6, Lines 420-480] and in LADEL [Appendix A.4.6, Lines 265-298].



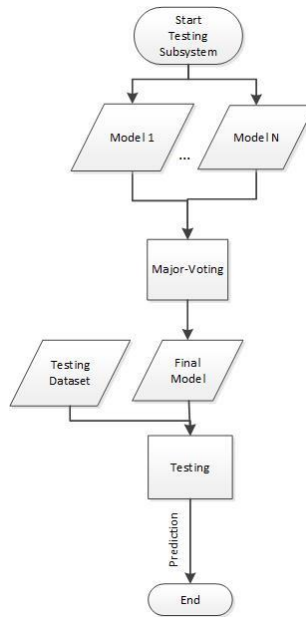


Figure 3.5: Testing subsystem

Next step is test the testing dataset by final model extracted in voting phase [Appendix A.3.6, Lines 482-551] and [Appendix A.4.6, Lines 299-325] for LADEL. Final step is to save some values in an Excel format file [Appendix A.3.7, A.4.7]. These values are: 1- order of dataset, which can be random for Bagging-like ensembles and sorted in LADEL format for labels, ordered. 2- number of node usage, which is equal to the number of disjoint

Partitions, parallelization level and used processors. 3- Partition number. 4- Bagging-like division method. 5- used classifier (MLP/CART). 6- Cross validation score [21] in by final model. 7- Test-score in testing dataset by final model. 8- Run-time for disjoint partition. 9- Run-time for transferring data through server to client. 10- Training time. 11- Voting time. You can see the results in excel output files in [Appendix A.I-L].

## **3.6 Testing DELS**

For testing the DELS system, first, we focus on comparison accuracy on Bagging-like methods by CART and MLP classifier in single and multi-node architecture, and then we focus on comparison test-score and execution time on Bagging-like methods and LADEL in multi-node architecture. Therefore, in general, the program runs in two ways, single-node and multi-node.

### **3.6.1 Accuracy comparison for Bagging-like partitioning method**

For testing this system, first we evaluate the accuracy of the Bagging-like methods (D/DB/SB/NRSB) work in distributed single-node classification algorithms and second we compare the accuracy of the same approach in distributed multi-node classification algorithms. Each of these tests is experienced with one of the decision tree and neural network classification algorithms. CART is selected from decision tree algorithm, and MLP is selected from neural network algorithm. You can find Python source code to compare the accuracy by MLP for Bagging-like methods in Appendix A.1 and by the CART algorithm in Appendix A.2. Both of these codes are executed in single and multiple systems as follows.

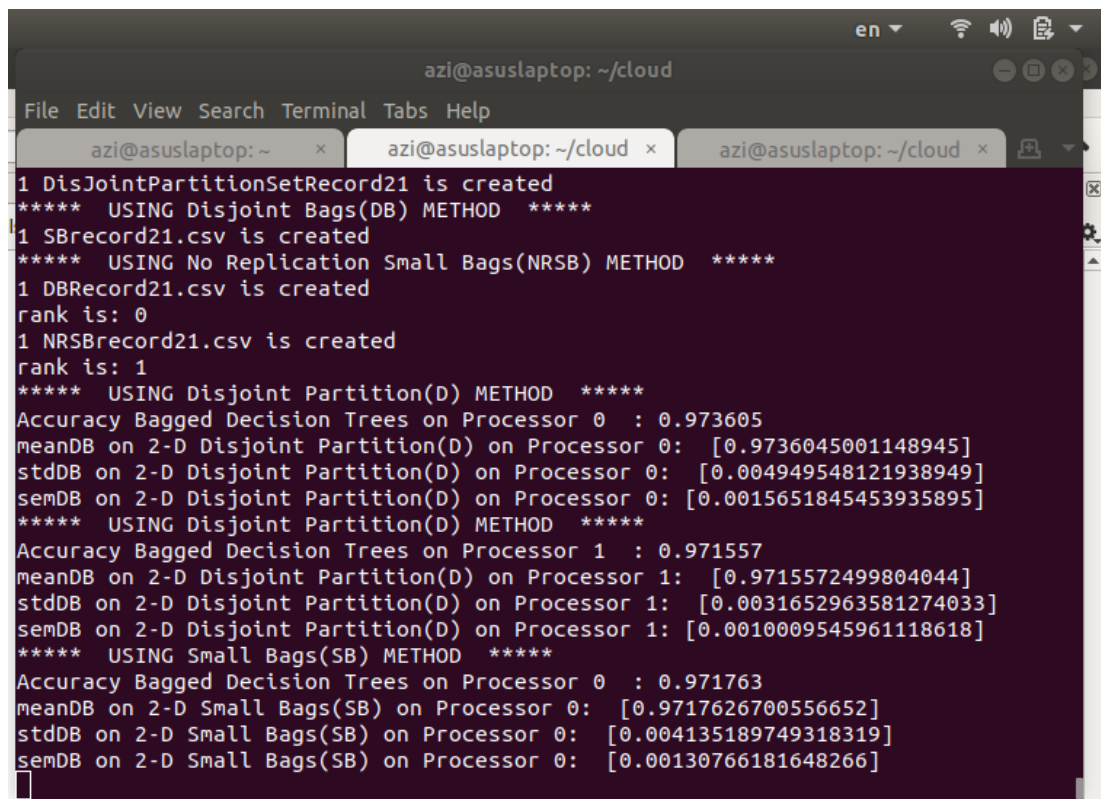
#### **3.6.1.1 Accuracy in single-node**

First approach of implementation is to execute program in single node architecture. The program run in one system and distribute the parallel tasks in the resources of one node. As a proof-of-concept distributed tasks prepared by MPI. MPI manage the task distribution and the processors each task needed. We can specify which system processors each task will use and how many processors each system will start with. First, we set the value in partno to the desired number of dataset partition (DP). Since each DP is assigned a task for further processes, so the partno value and the number of processors we use in the MPI command should be equal. We set the

partno into 2, 4, 6, 8, 10 and 12. And the input dataset is KDD Cup 99 dataset which set in the filename variable [Appendix A.1, Line 51]. For example, we start test for the partno equal to two. Using the following command, the data is divided into two parts and assigned to two separate tasks with two processors in parallel.

```
mpirun -np 2 python3 source-code.py
```

MPI assigns a separate rank to each of these tasks, and each rank considers its own dataset partition. All tasks start running simultaneously and MPI assign separate resources for each of them in same system. Figure 3.6 shows the program execution in the terminal and program outputs.



```
azi@asuslaptop: ~/cloud
File Edit View Search Terminal Tabs Help
azi@asuslaptop: ~ x azi@asuslaptop: ~/cloud x azi@asuslaptop: ~/cloud x
1 DisJointPartitionSetRecord21 is created
**** USING Disjoint Bags(DB) METHOD ****
1 SBRecord21.csv is created
**** USING No Replication Small Bags(NRSB) METHOD ****
1 DBRecord21.csv is created
rank is: 0
1 NRSBRecord21.csv is created
rank is: 1
**** USING Disjoint Partition(D) METHOD ****
Accuracy Bagged Decision Trees on Processor 0 : 0.973605
meanDB on 2-D Disjoint Partition(D) on Processor 0: [0.9736045001148945]
stdDB on 2-D Disjoint Partition(D) on Processor 0: [0.004949548121938949]
semDB on 2-D Disjoint Partition(D) on Processor 0: [0.0015651845453935895]
**** USING Disjoint Partition(D) METHOD ****
Accuracy Bagged Decision Trees on Processor 1 : 0.971557
meanDB on 2-D Disjoint Partition(D) on Processor 1: [0.9715572499804044]
stdDB on 2-D Disjoint Partition(D) on Processor 1: [0.0031652963581274033]
semDB on 2-D Disjoint Partition(D) on Processor 1: [0.0010009545961118618]
**** USING Small Bags(SB) METHOD ****
Accuracy Bagged Decision Trees on Processor 0 : 0.971763
meanDB on 2-D Small Bags(SB) on Processor 0: [0.9717626700556652]
stdDB on 2-D Small Bags(SB) on Processor 0: [0.004135189749318319]
semDB on 2-D Small Bags(SB) on Processor 0: [0.00130766181648266]
```

Figure 3.6: Program execution in terminal, single-node

The printed output in terminal is included:

- 1- the name of each disjoint partition dataset

- 2- rank number of each task
- 3- mean accuracy (meanD/ meanDB/ meanSB/ meanNRSB), standard deviation(stdD/ stdDB/ stdSB/ stdNRSB) and standard error of mean (semD/ semDB/ semSB/ semNRSB) for each of methods.

As shown in the Figure 3.6, each of these results is executed and printed from separate processors.

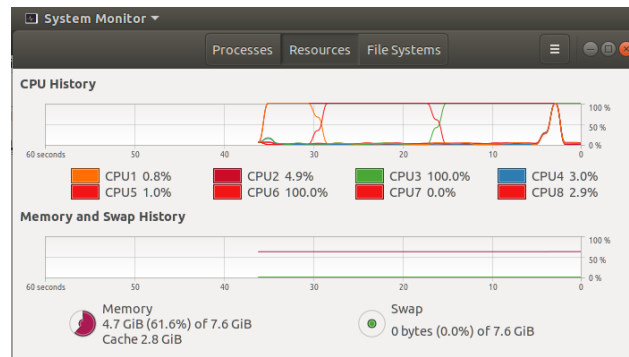


Figure 3.7: Resource monitoring in master system with 8 processors

Figure 3.7 illustrates the resource monitoring during program execution. It shows CPU3 and CPU6 working, and each of these resources is assigned to one task. The testing of other number of partitions is done as in the above method, with the difference that when executing command *mpirun*, we use *np* parameter (number of processors), which is the same as PARTNO value in programs. These tests are repeated for values 2, 4, 6, 8, 10, 12 as well as Appendix A.2 for CART classification algorithm. The tabular output results of these tests are shown in Appendices B and C. Appendices F and G contain the plot output of these tests.

### 3.6.1.2 Accuracy in multi-node architecture

Second approach of implementation is to execute program in multi node architecture. The program runs in more than one system and distributes the parallel tasks in the resources of multiple nodes. We first make sure that the physical network connections are established. We test the setting of IPs and netmask interface on the systems by *ifconfig* command to be sure about LAN connections are established and have correct configuration. Command *ping* is used to test the reachability of a host on an IP network. Shared storage should check on both systems. The contents of the CLOUD and CLOUDC1 folders on both systems must be visible. Input dataset and application source file are stored in the CLOUD folder. This folder also stores partitioned data sets, final model files, Excel output files, and plots. CLOUDC1 folder stores models generated by tasks other than the “Master task”. Both of these folders are shared in all nodes. For checking validity of these folders in shared architecture, mount command can be used (Section 2.6).

Now we can run program in multi-node by using the following command.

```
mpirun -n 2 -H master:1,client:1 python3 source-code.py
```

MPI assigns a separate rank to each of these tasks same as single-node way, but running each task in different system and using resources of multi systems in parallel. *master* is the alias name of ASUS laptop and *client* is the alias name of the SONY laptop and the number in front of these aliases is the number of processor(s) intend to use. Figure 3.8, illustrate the execution of above command in terminal and the outputs of each tasks in two nodes. This command run program by 2 processors, which each one of them executes in one node. One task in master node and the other

one in client node. The tabular output results of these tests are shown in Appendices D and E. Appendices H and I contain the plot output of these tests.

```

azl@asuslaptop:~/cloud$ mpirun -n 2 -H master:1,client:1 python3 ./mpi-singlenode-classifier-v5.py
partition number is : 2
1 DataSet(s) are used in Client and 1 in Master Node
1589906921.486529
***** USING Disjoint Partitioning(D) METHOD For 2 partitions *****
*****
Totaly 0 seconds for Disjoint Partitioning of data in Master node
***** USING Small Bags(SB) METHOD For 2 partitions *****
*****
partition number is : 2
*****
Totaly 0 second for Small Bags data in Master node
***** USING Disjoint Bags(DB) METHOD For 2 partitions *****
*****
Totaly 0 second for Small Bags data in Master node
***** USING No Replication Small Bags(NRSB) METHOD For 2 partitions *****
*****
Totaly 0 second for Small Bags data in Master node
Total Time For Partitioning is : 1.2565994262695312
Bagging Files are ready to transfer to Client(s) ...
bagging file is ready to transfer
Bagging Transformation from processor 1 is started.
File is Transferred
Model is received from clients
Mean Results of voting in D is : 0.9442431326709526
ensembleD is : [0.92402104 0.87960257 0.95908825 0.94418469 0.75131502 0.99941555
1. 0.99649328 0.98831093]
Mean Results of voting in DB is : 0.9442431326709526
ensembleDB is : [0.92402104 0.87960257 0.95908825 0.94418469 0.75131502 0.99941555
1. 0.99649328 0.98831093]
Mean Results of voting in SB is : 0.9442431326709526
ensembleSB is : [0.92402104 0.87960257 0.95908825 0.94418469 0.75131502 0.99941555
1. 0.99649328 0.98831093]
Mean Results of voting in NRSB is : 0.9442431326709526
ensembleNRSB is : [0.92402104 0.87960257 0.95908825 0.94418469 0.75131502 0.99941555
1. 0.99649328 0.98831093]
TbeginTESTD is: 1589907887.9560523
The Accuracy of Disjoint partitioning(D) is : 0.9804703338288958
The Predicted result of D method is produced.
The Accuracy of Small Bages(SB) is : 0.9804703338288958
The Predicted result of SB method is produced.
The Accuracy of No Replication Small Bages(NRSB) is : 0.9804703338288958
The Predicted result of NRSB method is produced.
The Accuracy of Disjoint Bages(DB) is : 0.9804703338288958
The Predicted result of DB method is produced.
TFTPdc0A [['Node0', 0], ('Node1', 4.558545172214508)]
TmdLdc0A [['Node0', 0.6186752319335938], ('Node1', 0.865351676940918)]
i is : 0
TFTPdc0A[i] ('Node0', 0)
TmdLdc0A[i] ('Node0', 0.6186752319335938)
i is : 1
TFTPdc0A[i] ('Node1', 4.558545172214508)
TmdLdc0A[i] ('Node1', 0.865351676940918)
azl@asuslaptop:~/cloud$ mpirun -n 4 -H master:3,client:1 python3 ./mpi-singlenode-classifier-v5.py

```

Figure 3.8: Program execution in terminal, multi-node

Figure 3.9 shows the resource monitoring in second node when we use command `mpirun -n 12 -H master:8,client:4 python3 source-code.py`. In this command, MPI starts 12 concurrent tasks, with using all the processors in first and second laptop. As you can see in Figure 3.11, all four CPUs in second node and local network are busy.

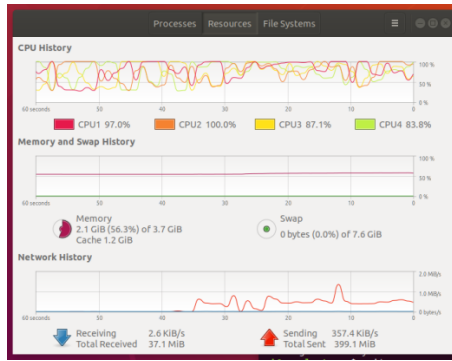


Figure 3.9: Resource monitoring in client system with 4 processors

### 3.6.2 Prediction and test-score and runtime comparison for Bagging-like partitioning method and LADEL algorithm

First, to test the DELS system in Bagging-like methods by MLP work on distributing multi-node classification algorithms, we execute given in the Appendix A.3 program using the *mpirun* command, same as previous program in multi-node system. The tabular output results are shown in Appendix J.

Second we evaluate the test-score and runtime of the LADEL method work on distributing multi-node classification algorithms by MLP [Appendix A.4] and CART [Appendix A.5] and the results of that program execution are in Appendix K and L.

The summary of achievements on this experiment are, in Bagging-like partitioning method as the number of data partitioning increases, the degree of accuracy is slowly declining and the error rate increases in both single and multi-node systems. In LADEL model, prediction of results in MLP is fluctuating, but the trend of results becomes closer to each other with increasing number of partitions in both classifiers MLP and CART. For training time in ensemble classification, despite the increase in the number of nodes, the time to learn and build the model should be distributed in the systems and therefore have a downward trend. However, this time has an upward trend. This trend has improved from Bagging-like method to LADEL dramatically.

## Chapter 4

### EXPERIMENTS ON DELS

In this segment, we analyze three strategies for training classifier. The types of grouping and stratified the data that have been introduced as Bagging-Like methods in previous articles have been applied to input data and then a comparison between two different types of learning algorithms, one of the well-known Decision Tree algorithms called CART and other neural network type called multi-layer perceptron or MLP is done.

We focus on evaluation the prediction accuracy and the standard deviation and standard error of mean (SEM) of classifiers using a variety of approaches. The duration of the training is also checked. Finally, the order of the input data is changed using the LADEL method and the accuracy and error rate and learning time duration are evaluated using Algorithms MLP and CART.

The first approach uses four disjoint partition methods to stratify large input dataset into n smaller one and distribute them into different processors in single node as parallel tasks. Implementation on this section is by Python-V3.7.4 (64-bit) on Spyder-V3.3.6 and Microsoft MPI v9 with random data in single node. This dataset is tested by two CART and MLP classifiers. The second approach uses an implementation of the same tools but in network area on distributing data through client-server topology. In our experimental test, we just use two laptops one has a role of master and client both and the other as a client. The “client” is the name of



machine you would like to do computation with and “master” is the name of main computer, which expected to distribute the files to the other client systems. We use a 300Mb/s TP-Link wireless modem router for creating network between two laptops through local area network (LAN). Both systems are run in Ubuntu 18.04 LTS with up to 300 Mb/s network bandwidth connections through wired LAN. Again, these tests are used random dataset records trained by CART and MLP classifiers. The third approach uses the sorted dataset by classifications, which is included by all the class labels in training dataset produced by LADEL method. This approach is trained on distributed servers with both decision tree and neural network algorithms. We choose classification and regression tree and multilayer perceptron technique as they represent LADEL’s worst and best cases in terms of training time and predictive accuracy in previous scientist’s researches.

In all machine learning techniques the entire of training data should be loaded into ram memory to do the training, thus in large datasets which the size of in-memory are not large enough to fit the data satisfactorily, dividing data and distributing them into several machine and do learning in parallel is suggested and implemented in all above introduced approaches.

## **4.1 Experimental setup**

### **4.1.1 Environment setup**

We use the laptop ASUS VivoBook S14, Ubuntu 18.04 LTS instances with four cores (1.8 GHz, 1992 MHz, 8 Logical Processor(s)), 8 GB RAM, Page File Space 2.5 GB and 256 GB SSD Disk. Our second node is Desktop SONY VAIO, Ubuntu 18.04 LTS instances with two cores (2.4 GHz, 2400 MHz, 4 Logical Processors), 4 GB RAM, Page File Space 1.88GB and 500 GB disk space.

### 4.1.2 Datasets

KDD Cup 99 dataset is used for the KDD Cup competition 1999. This is an annual Data Mining and Knowledge Discovery Competition organized by ACM group. This database contains data to be distinguished between the different kinds of connections happening over the military network, whether is good (approved connection) or is a bad (unwanted, intrusive connection). Extracting well model to predict the connection type in the future communication is requested. The original dataset is about 4GB of compressed data from seven weeks of network traffic. This was processed into about five million connection records. Similarly, the two weeks of test data yielded around two million connection records.

HIGGS dataset has been produced using Monte Carlo simulations. The first 21 features (columns 2-22) are kinematic properties measured by the particle detectors in the accelerator. The last seven features are functions of the first 21 features; these are high-level features derived by physicists to help discriminate between the two classes. The first column is class column, containing labels 0 and 1. The original dataset is about 4GB of compressed data. This was processed into about eleven million records. In our network laboratory facility, we just used partial of original dataset as bellow in Table 4.1. KDD Cup 1999 data set for Bagging-like method [Appendix A.1.1-A.3.1] and HIGGS dataset for LADEL method [Appendix A.4.1-A.5.1].

Table 4. 1: Input data set attributes

Usage method	Data Set	#number of records	#Attributes	#Class labels	Size (MB)
Bagging-like	KDD Cup 1999[24]	48,871	44	23	6.2
LADEL	HIGGS[15]	1468	28	2	300KB

## 4.2 Accuracy comparison for single-node classifier ensemble

In this analysis, we investigate the predictive accuracy and standard error on creating model in four stratify types of input medium-size dataset into smaller size which fit the memory in local computer and single-node architecture. Based on the tabular results obtained in [Appendixes B-C] and plot results obtained in [Appendixes F-G], a comparison can be made between the value of accuracy, standard deviation and standard error of mean rate (SEM) in a medium-sized input dataset obtained (Table 4.1). This experimental comparison of the different approaches on two, four, six, eight, ten and twelve disjoint partitions (D) and classifier groups formed using the other three approaches (DB, SB, NRSB). We have chosen two classifiers on the category of decision tree and neural network machine learning algorithms for training these divided datasets to extract the models and accuracy of them. Codes implementing these experiments are available in [Appendices A.1] for CART and [Appendices A] for MLP.

As the first practical experience in Figure 4.1, the first data point represent the Mean accuracy with tolerance of standard error of mean when the data is divided into 2 sections by disjoint partition method (D) and using MLP classification algorithm. The rest of data points in plot show the same size division but in sequence DB, SB and NRSB Bagging-like data division methods. Each of these datasets as a separate task is assigned to the separate processors in standalone computer. You can consider other plot results for 4, 6, 8, 10 and 12 dataset divisions in Plot.G.2-6. In addition, Tabular results in Table.C.2-6.

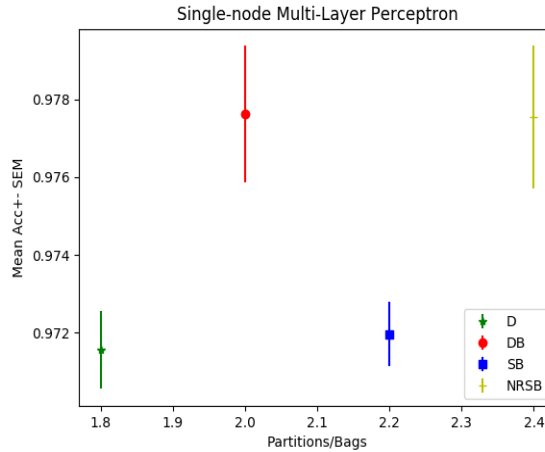


Figure 4.1: Accuracy comparison on Bagging-like ensembles by MLP for 2 disjoint partitions [Appendix G]

According to the plots in [Appendix G] using MLP, disjoint Bags (DB) is more accurate than other dataset partitioning methods. In this sample of experiments, the possibility of increased accuracy due to frequent records in each bag, as well as increasing the number of records increases. Duplicate records are available in each bag. AS in disjoint partition (D) method, there are no duplication records and only records are randomly entered and tested in the input file, so it is less accurate than the others are.

The same testing process is repeated by classification and regression tree (CART) classifier and the tabular results are given in Appendix B and the comparison plots in Appendix F. According to the plots in [Appendix F] using CART, Small Bag (SB) is more accurate than most other methods. In this sample of experiments, the probability of increasing accuracy increases due to the presence of duplicate records. Duplicate records are available in each bag and throughout the bag, so assuming that there is a possibility of increasing the number of records in a classification, it can be claimed that this method is more accurate than other methods. And in method

disjoint partition (D), there is no duplication in classifications and only records are randomly entered and tested in the input file, so it is less accurate than others.

It is clear from the statistics, the overall efficiency of data partitioning in all four methods decreases by increasing number of data divisions and processors. As the data loaded into memory is incomplete, the partial data is allocated to parallel tasks and processors, which reduces performance accuracy. In the same way, standard error of mean (SEM) has been increased slightly.

In another point of view, the CART classifier from the decision tree is more efficient than multilayer perceptron in all data division methods. Extracted accuracy in using CART classifier is almost 0.998 but in MLP classifier is between 0.96-0.97. Also, if we compare the amount of standard deviation obtained using these two classifiers with the number of the same processors and different classifications, we conclude that, by using MLP classifier, the standard deviation is less than the CART and the standard error of mean (SEM) is a little bit higher than using CART classifier. These changes are very small, about one thousandth or ten thousandths. Fortunately, as the number of data partitions increases and the number of processors increases, CART classification gives us even better accuracy. The standard error rate and the standard deviation are less than MLP.

We did comparison of the time required to run the program in various data partitioning by MLP or CART. Table 4.2 illustrates the time required to obtain accuracy for Bagging-like methods by decision tree and neural network classifier in different number of partitioning separately.

Table 4.2: Run-time in single-node mode

<b>Partitions? #</b>	<b>MLP Run-time (sec)</b>	<b>CART Run-time (sec)</b>
<b>2</b>	4080	360
<b>4</b>	2520	300
<b>6</b>	1440	240
<b>8</b>	1140	180
<b>10</b>	840	120
<b>12</b>	780	120

We measure run-time by T0 variable [Appendix A.1.2- Lines 220]. To measure the duration of the program (T0), we subtract the start time of the program (Tbegin0) [Appendix A.1.1- line 22] from the time of the end of the program (Tend0) [Appendix A.1.5- lines 450-452]. You can find raw data output in Appendix [B-C]. Table 4.2 illustrates neural network algorithm requires more time to learn, while the decision tree returns the result faster. As you can see, the CART algorithm is about 6 to 11 times faster than the MLP algorithm for Bagging-like methods in single-node.

Since different parts perform learning operations in parallel, increasing the number of partitions reduces the training time. This conclusion is true in both learning algorithms.

### **4.3 Accuracy comparison for multi-node classifier ensemble**

In this part of experiment, we run the program [Appendix A.1 and A.2] in multi node. In this method, exactly same as single-node by increasing the number of partitions, the accuracy rate is slowly reduced and the error rate and standard deviation are increased. In all data partitioning methods, we see a decrease in accuracy. You can refer to [Appendixes D-E] for tabular detailed results and [Appendixes H-I] comparison plots.

In multi-node system method, the input data partition into the small size datasets and then distributed to different processors in various nodes through the network. The results show that the accuracy reduces compared to the single-node method. This reduction in accuracy is negligible for data tested (Table 4.1). We may also see a greater decrease in predictable accuracy as the size of the input data and the number of network nodes increase.

Comparing the dataset partitioning methods (Bagging-like) and the degree of accuracy [Appendixes H-I], it can be said that Small Bag (SB) method still has the highest level of accuracy in multi-node. After the SB method, the disjoint Bags (DB-No replication across, Larger) method achieved the highest level of accuracy.

In terms of runtime, the same source code used for single-node use for multi-node, except that mpirun command uses the client-server format. Thus, we measure runtime by  $T_0$  variable [Appendix A.1.2- Lines 220]. To measure the duration of the program ( $T_0$ ), we subtract the start time of the program ( $T_{begin0}$ ) from the time of the end of the program ( $T_{end0}$ ). You can find raw data output for CART in Appendix [H] and for MLP in [Appendix I]. As you illustrate in Table 4.3, in addition to the fact that MLP algorithm is slower than the CART learning algorithm, it saves time by increasing the number of disjoint partitions, but the runtime in multi-node method is longer than in the single-node method. The time of distribution and transfer of data from the server to the clients, add to the execution time of other parts, so this slowness is understandable. The main result is by increasing the number of partition datasets, the runtime decrease.

Processor # Master: client column in Table 4.3 shows the number of processor we assigned to the DELS system by *mpirun*. *Mpirun* command has a parameter to assign the number of resources we need from master and clients. Since the ASUS, laptop has a better processor with faster speeds in running the program and the number of processors is twice of SONY laptop, so we assign 2/3 of the processor required by ASUS and 1/3 of the SONY to the program.

Table 4.3: Run-time in multi-node mode

Partitions #	Processor #	MLP	CART
	Master: Client	Runtime (sec)	Runtime (sec)
2	1:1	6180	540
4	3:1	2280	240
6	4:2	1560	180
8	6:2	960	180
10	7:3	780	180
12	8:4	960	180

#### 4.3.1 Prediction comparison for Bagging-like partitioning method

In machine learning a common problem is the tendency to memorize the data classifiers have been trained on [24]. It can make the model looks great on the training data but in actuality it has no ability to generalize on new data that is given. To gain perspective into how the model is actually doing, we use a train/test split (75%-25%). After data training and get the model, the model is tested on the test data split to realise its performance in comparison with the training data and obtain the test score. Cross-validation is any of various similar model validation techniques for assessing how the results of a statistical analysis will generalize to test split of data set. It is used for prediction, and we want to estimate how accurately a predictive model will perform in practice.



In this experiment, training dataset divide by Bagging-like dataset partitioning methods in multi node systems to get the models. The test split of data is tested by these models and we compare the test score of them [Appendix A.3]. The remarkable point in this part of the test is that despite using four different methods of data partitioning (D, DB, SB and NRSB) and observing a small amount of difference in accuracy of them, by adding major-voting part in the server side, the final cross-validation in all these methods is the same [Table 4.4] and we have a good score of 0.829 in cross-validation to evaluate the extraction model and a very high score of 0.955 for test-score to evaluate model on testing data. You can see the detail results of the studies in Appendix J, Tables J.1-6.

Table 4.4: Accuracy of model in training dataset by Cross-validation value & accuracy of model in testing dataset by test scores in Bagging-like methods by MLP

<b>Partitions #</b>	<b>Cross-validation</b>	<b>Test-score</b>
<b>2</b>	0.829	0.955
<b>4</b>	0.829	0.955
<b>6</b>	0.829	0.955
<b>8</b>	0.829	0.955
<b>10</b>	0.829	0.955
<b>12</b>	0.829	0.955

To compare the runtime of the DELS system by MLP, the value of T0 [Appendix A.3.6, line 553] is measured. As you see in Table 4.5, the training time decreases as the number of disjoint partitions increases, while the scoring time increases. There are two reasons for the decrease in training time with the increase in the number of partitions: 1) Fewer records are available in each partition to learn; 2) Parallel processing is done on the parts. The reason for this increase in scoring time is related to that as the number of models obtained increases, the duration of majority-voting increases.

Table 4.5: Training and scoring time in Bagging-like methods by MLP in multi-node mode

<b>Partitions #</b>	<b>Processor # Master: Client</b>	<b>Training time (sec)</b>	<b>Scoring time (sec)</b>
<b>2</b>	1:1	17.92	1901
<b>4</b>	3:1	12.55	4099
<b>6</b>	4:2	12.22	5708
<b>8</b>	6:2	11.99	8279
<b>10</b>	7:3	11.90	10083
<b>12</b>	8:4	11.4	11709

#### 4.4 Training time and scoring time comparison for LADEL

In this part of the experimental test, we review and compare training time and scoring time for label-aware distributed ensemble learning (LADEL) method with Khalifa’s article. The data set used in this part of the experimental test is HIGGS with the specifications mentioned in Table 4.1 of the second row. The MLP implementation code is available in [Appendix A.4] and the raw data outputs of this test implementation code are available in [Table K]. The CART classification algorithm repeats the same experimental test. The code is in [Appendix A.5] and output tabular results in [Appendix L].

In this analyze, we investigate the effect of increasing number of dataset partitioning in training time and scoring time for producing training data set in the format of LADEL with all class labels include in each dataset partition, in the ratio of [1:1]. Ratio of [1:1] means, original balanced on HIGGS dataset. It has two class labels with an equal records number belonging to each class. Disjoint partitioned datasets train in distributed process on single node or multiple nodes. The training classifiers gather in vote-based ensemble on the master node.

We analyze the training time in 2-12 number of disjoint partitions in parallel processing. As you can see in figure 4.2, adding number of training dataset leads to increasing the training parallelization, which leads to a shorter training time. By increasing number of parallel process, the graph has a downward trend, but having 12 ensemble classifiers graph shows slightly increase on the training time. The results illustrates that the training time in CART is at most 3 times the MLP algorithm, so using MLP is faster than CART.

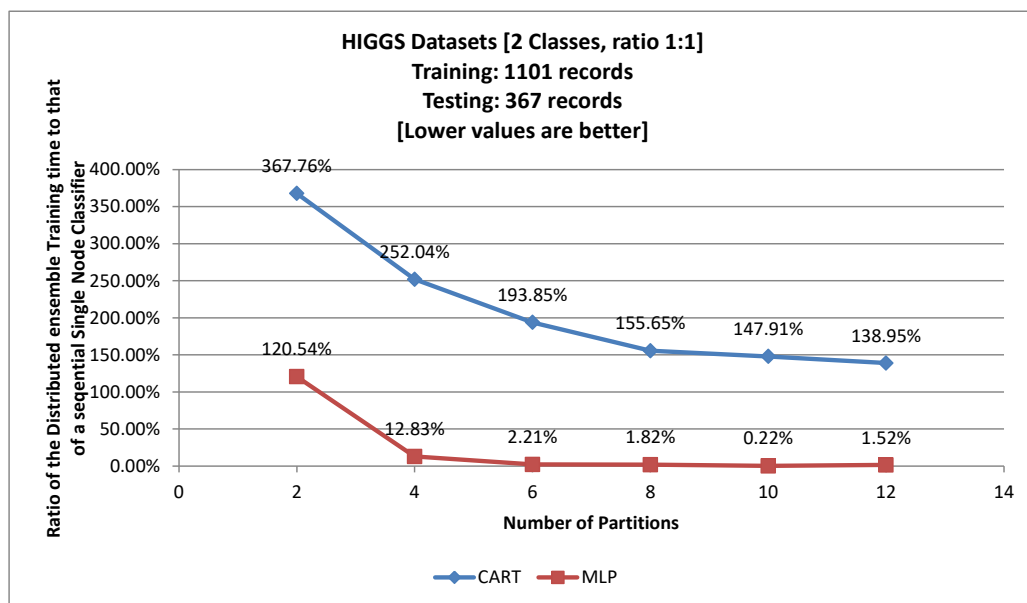


Figure 4.2: Ratio of the distributed ensemble training time to that of single-node classifier

The results obtained in Khalifa’s article [3] presented in Figure 2.17 are in line with the results of this study for training time. Since the hardware and software specifications of the tools used are not the same, and the number of data samples used in HIGGS dataset is not the same, we will leave the statistical comparison of the results to future studies.

In terms of scoring time, as the number of ensemble classifiers increases, the more time require to calculate the score the new data because the number of classifiers

increases. In this test, testing data has 367 records. We analyze scoring time in distributed classifiers in single-node (Figure 4.3) and in multi-node (Figure 4.4). Both approaches show mostly an upward trend.

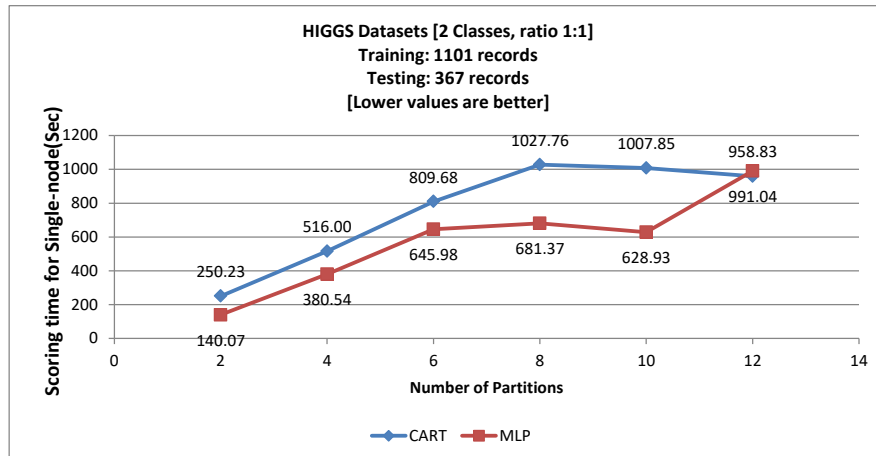


Figure 4.3: Scoring time for single-node mode

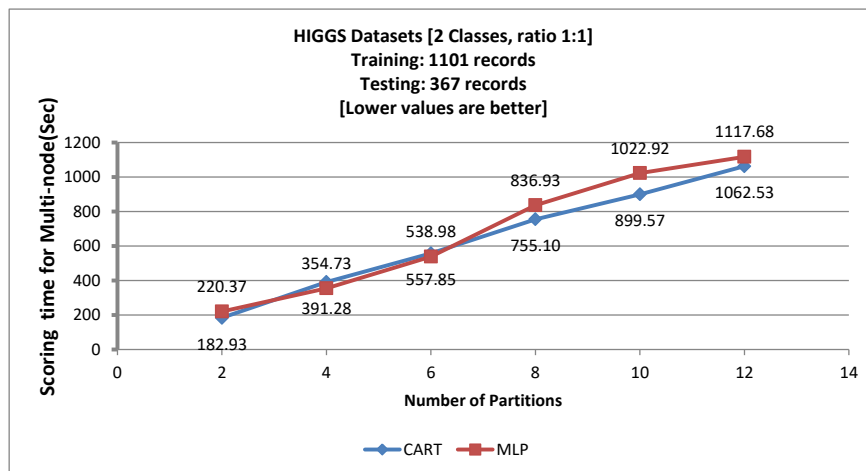


Figure 4.4: Scoring time for multi-node mode

Comparison of the results obtained versus known from the Khalifa's article [3, Fig 10], the results in Figure 2.17 extracted 4.1% in 1000 nodes faster in distributed ensemble classifier than single classifier by MLP in almost 7 million training records in 2 classes, ratio 1:1 and our results are in maximum 12 processors in almost 1

thousand training records in 2 classes by ratio 1:1 and it shows faster training time in MLP than CART classifier.

In summery our experimental findings are in DELS for Bagging-like methods show that as the number of data divisions increases the level of learning accuracy decreases and Error rate increases in both single and multi-node structure with parallel processing. Using CART, SB has almost better accuracy and D has almost the less accuracy. With MLP, DB has almost better accuracy and D has less accuracy. CART classification provides slightly better accuracy in single-node and multi-node experiments than MLP. The main purpose of these practical experiments is to reduce the training time. Training time decreases with increasing number of training datasets.

## Chapter 5

### CONCLUSION

This thesis considers a problem of minimizing training time of classifiers by distributing data across distributed nodes. To study the problem, DELS with several partitioning methods and classifiers was implemented and experiments were conducted. Partitioning methods used are D, DB, SB, NRSB and DSS in LADEL algorithm. The classifiers used are CART and MLP. The single-node and multi-node architecture are used to distribute parallel tasks. MPI is used for distributing disjoint partitions in parallel tasks.

DELS is implemented as a system with Input/ Output subsystem (IOS), partitioning subsystem including LADEL model, training subsystem and testing subsystem. The tools were used are Ubuntu 18.04 LTS, Python-V3.7.4 (64bit) on Spyder-V3.3.6 and Microsoft MPI v9. We just use two laptops one has a role of master and client both and the other as a client. TP-Link wireless modem router with 300Mb/s speed are used for creating network between two laptops. The KDD Cup 99 dataset for the KDD Cup competition 1999 and HIGGS dataset that has been produced using Monte Carlo simulations are used in DELS system.

We followed mainly [3] where LADEL system was proposed and [2] where Bagging-like methods was proposed. Our experimental results obtained for DELS using datasets, 1- KDD Cup 99 dataset for Bagging-like partitioning dataset methods and, 2- HIGGS dataset that has been produced using Monte Carlo simulations for

LADEL algorithm. The results obtained in [3], Figure 2.17 are in line with the results of Figure 4.2 for training time. Both results show, adding number of training dataset leads to increasing the training parallelization, which leads to a shorter training time. Also the results of Figures 2.18 and 2.19 about scoring-time comply with the results we obtained by DELS in Figures 4.3 4.4. As the number of ensemble classifiers increases, the more time require to calculate the score the new data because the number of classifiers increases.

To obtain experimental results, first, the implementation of Bagging-like methods, D, DB, SB, and NRSB, respectively investigated in single-node with parallel processors in the same system and then multi-node with parallel tasks in separate systems with two learning algorithms CART and MLP. From the results obtained in this experimental set, we concluded that in Bagging-like methods, the level of learning accuracy decreases and the error rate increases by increasing number of data divisions. These results are true for both single-node and multi-node methods with parallel processing. SB and D are the best and worst dataset partitioning methods by CART, respectively. In addition, for MLP classification algorithms, DB and D are the best and worst dataset partitioning methods. Simultaneous execution of tasks in distributed systems greatly reduces the run-time of the learning training data.

Data distribution by LADEL algorithm has been the next part of our experiment, which groups' data based on the uniform distribution of classifications in all files. Although data preparation takes time, it can be neglected in terms of better performance. The results of this section show that by an increase in the number of distributed nodes, the learning and modeling time is distributed in the systems, so less time is needed to get the results, but at the same time the scoring time increases.

In general, the CART classifier from the decision tree is more efficient than MLP in Bagging-like methods training split in single-node with accuracy of 0.998 but in MLP, the accuracy is 0.96-0.97 and SB method has higher accuracy than other methods. Cross validation in Bagging-like methods is 0.829 and test-score is 0.955.

The training time in a single-node with Bagging-like method and classification by MLP is 4080 and 780 seconds for 2 and 12 partitions, respectively. A similar experiment was performed by the CART algorithm with 360 and 120 seconds for 2 and 12 partitions. Runtime in multi-node by MLP is 6180 and 960 seconds for 2 and 12 disjoint partitions while by CART is 540 and 180 seconds for 2 and 12 disjoint partitions.

Using LADEL algorithm in DELS significantly reduces training time. The experimental results show that in both single-node and multi-node, MLP learns faster than CART. Best training time occurred in 12 distributed nodes with 0.46 milliseconds by MLP, while learn training dataset in 12 nodes by CART takes 21.39 seconds. Totally, results show that training time decreases in distributed systems while scoring time increases. To reduce the scoring time, the ensemble classifiers can be copied to the distributed nodes, and just as the training data is divided into number of nodes, the testing dataset can be divided and sent to the distributed nodes, and the scoring operation can be performed in parallel. This part of experiment is left for the future work.



## REFERENCES

- [1] Christopher, C. (2010). *Encyclopaedia britannica: Definition of data mining*. Retrieved from <https://www.britannica.com/technology/data-mining>
- [2] Chawla, N. V., Moore, T. E., Hall, L. O., Bowyer, K. W., Kegelmeyer, W. P., & Springer, C. (2003). Distributed learning with bagging-like performance. *Pattern recognition letters*, 24(1-3), 455-471.
- [3] Khalifa, S., Martin, P., & Young, R. (2019). Label-aware distributed ensemble learning: A simplified distributed classifier training model for big data. *Big Data Research*, 15, 1-11.
- [4] Bhavsar, P., Safro, I., Bouaynaya, N., Polikar, R., & Dera, D. (2017). Machine learning in transportation data analytics. In *Data analytics for intelligent transportation systems* (pp. 283-307). Elsevier.
- [5] Van Steen, M., & Tanenbaum, A. S. (2017). *Distributed systems*. Leiden, The Netherlands: Maarten van Steen.
- [6] Finlay, S. (2014). *Predictive analytics, data mining and big data: Myths, misconceptions and methods*. Springer.
- [7] Brownlee, J. (2017). What is the difference between test and validation datasets. *Machine Learning Mastery*, 14.

- [8] Gupta, P. (2017). *Decision trees in machine learning*. Retrieved from <https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052>
- [9] Steinberg, D., & Colla, P. (2009). CART: classification and regression trees. *The top ten algorithms in data mining*, 9, 179.
- [10] Nicholson, C. (2019). A beginner's guide to multilayer perceptrons (MLP).
- [11] Zhou, V. (2019). *Machine Learning for Beginners: An Introduction to Neural Networks*. *Towards Data Science*.
- [12] Opitz, D., & Maclin, R. (1999). Popular ensemble methods: An empirical study. *Journal of artificial intelligence research*, 11, 169-198.
- [13] Stonebraker, M. (1986). The case for shared nothing. *IEEE Database Eng. Bull.*, 9(1), 4-9.
- [14] Bauer, E., & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning*, 36(1), 105-139.
- [15] HIGGS UCI data set. (2014). Retrieved from <https://archive.ics.uci.edu/ml/datasets/HIGGS>
- [16] Buyya, R., Vecchiola, C., & Selvi, S. T. (2013). *Mastering Cloud Computing: Introduction 2. Principles of Parallel and Distributed Computing 3. Virtualization*

4. *Cloud Computing Architecture* 5. *Aneka: Cloud application platform* 6. *Concurrent Computing: Thread programming* 7. *High-Throughput Computing: Task Programming* 8. *Data Intensive Computing: Map-Reduce Programming* 9. *Cloud Platforms in Industry* 10. *Cloud Applications* 11. *Advanced Topics in Cloud Computing*. Morgan Kaufmann.

[17] Snir, M., Gropp, W., Otto, S., Huss-Lederman, S., Dongarra, J., & Walker, D. (1998). *MPI--the Complete Reference: the MPI core* (Vol. 1). MIT press.

[18] Barlas, G. (2014). *Multicore and GPU Programming: An integrated approach*. Elsevier.

[19] Dagum, L., & Menon, R. (1998). OpenMP: an industry standard API for shared-memory programming. *IEEE computational science and engineering*, 5(1), 46-55.

[20] Bejarano, J. (2013). An Introduction to parallel programming with MPI and python. Retrived from <https://materials.jeremybejarano.com/MPIwithPython/>

[21] Browne, M. W. (2000). Cross-validation methods. *Journal of mathematical psychology*, 44(1), 108-132.

[22] Kubat, M. (2017). *An introduction to machine learning*. Springer International Publishing AG.

[23] KDD Cup 1999 UCI Data Set. (1999). Reftived from <http://archive.ics.uci.edu/ml/datasets/kdd+cup+1999+data>

## **APPENDICES**

# Appendix A: Software source code- implementation of DELS

## 1) Accuracy for Bagging-like methods by MLP ( Python source code )

**Inputs:** filename, data file; partno[], array of partitions; base\_estimator, classification algorithm;

**Outputs:** meanD [] array for mean of accuracy by “disjoint partitioning”; meanDB [] array for mean of accuracy by “disjoint bags”; meanSB [] array for mean of accuracy by “small bags”; meanNRSB [] array for mean of accuracy by “no replication small bags”; stdD [], stdDB [], stdSB [], stdNRSB [] standard deviation for D/DB/SB/NRSB; semD [],semDB [],semSB [],semNRSB [] standard error of mean for D/DB/SB/NRSB;

### 1. Definitions and inputs:

```
1          # -*- coding: utf-8 -*-
2          """
3          Created on Sat Aug 10 17:45:38 2019
4
5          @author: Azi
6          """
7
8          import random
9          import math
10         import csv
11         import os
12         import numpy as np
13         from mpi4py import MPI
14         from pandas import read_csv
15         # Importing Bagged Decision Trees
16         from sklearn.model_selection import KFold
17         from sklearn.model_selection import cross_val_score
18         from sklearn.ensemble import BaggingClassifier
19         #from sklearn.tree import DecisionTreeClassifier
20         from sklearn.linear_model import Perceptron
21
22         import xlswriter
23         Tbegin0 = time.time()
24         comm = MPI.COMM_WORLD
25         rank = comm.rank
26         size = comm.size
27         name = MPI.Get_processor_name()
28         # MEAN and STD(standard deviation) and SEM(standard error of the Mean) arrays
29         meanD = []
30         meanSB = []
31         meanNRSB = []
32         meanDB = []
33         stdD = []
34         stdSB = []
35         stdNRSB = []
36         stdDB = []
37         semD = []
38         semSB = []
39         semNRSB = []
40         semDB = []
41
42         #####
43         ##          DELS inputs are          ##
44         ##          1-input dataset((kddcup 1999)) in filename variable          ##
45         ##          2-number of partitionsets in partno array          ##
46         ##          3-classification algorithm in base-estimator variable          ##
47         ##          (Perceptron for MLP & DecisionTreeClassifier for CART)          ##
48         #####
49
50         sharedfolder = '/home/azi/cloud/'
51         partno = [2,4,6,8,10,12] # Array, number of partition sets
52         filename = sharedfolder+ 'kddcup.data.corrected-sample.csv'
53         base_estimator=Perceptron
54
55         #####
56         ## dataset columns name. Last column(class) is the labeling criteria.          ##
57         ## class type: back,buffer_overflow,ftp_write,guess_passwd,imap,          ##
58         ## ipsweep,land,loadmodule,multihop,neptune,nmap,normal_perl_phf,          ##
59         ## pod_portsweep,rootkit,satan,smurf,spy,teardrop,warezclient,          ##
60         ## warezmaster.          ##
61         #####
62
63         names = ['rows','num','duration','protocol_type','service','flag','src_bytes','dst_bytes','land',
64                 'wrong_fragment','urgent','hot','num_failed_logins','logged_in','num_compromised',
```

64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148

```
'root_shell','su_attempted','num_root','num_file_creations','num_shells','num_access_files',  
'num_outbound_cmds','is_host_login','is_guest_login','count','srv_count','error_rate',  
'srv_error_rate','error_rate','srv_error_rate','same_srv_rate','diff_srv_rate',  
'srv_diff_host_rate','dst_host_count','dst_host_srv_count','dst_host_same_srv_rate',  
'dst_host_diff_srv_rate','dst_host_same_src_port_rate','dst_host_srv_diff_host_rate',  
'dst_host_error_rate','dst_host_srv_error_rate','dst_host_error_rate',  
'dst_host_srv_error_rate','class']
```

## 2. Partitioning Subsystem for Bagging-like methods:

```
if rank == 0: # Master process  
    for n in partno:  
  
        #####  
        ##          Read file          ##  
        #####  
  
        raw_data = open(filename, 'rt')  
        reader = csv.reader(raw_data, delimiter=',', quoting=csv.QUOTE_NONE)  
        x = list(reader)  
        data = np.array(x).astype('str')  
        #read no of records group by classes name  
        maindata = read_csv(filename,names= names)  
        class_counts = maindata.groupby('class').size()  
        #print(Name and Number of Classes in master node is : ',class_counts)  
        #print(Total number of records are : ',sum(class_counts))  
  
        #####  
        ##          partitioning in          ##  
        ##          disjoint Partition(D) Algorithm          ##  
        #####  
  
        print("***** USING disjoint Partition(D) METHOD *****")  
        print('n is : ',n)  
  
        #create random file  
        randomdf = random.sample(x,np.size(data,0)) #without replacement  
        p=[randomdf[i::n] for i in range(n)]  
  
        for i in range(0,n):  
            in_file = open(sharedfolder+'tmp%d%i.csv' %(n,i),'w')  
            with in_file:  
                writer = csv.writer(in_file)  
                writer.writerow(p[i])  
  
            # close files  
            in_file.close()  
  
            in_file = open(sharedfolder+'tmp%d%i.csv' %(n,i),'r')  
            with open(sharedfolder+'DisJointPartitionSetRecord%d%i.csv' %(n,i), "w") as out_file:  
                for line in in_file:  
                    if line.strip():  
                        out_file.write(line)  
            in_file.close()  
            out_file.close()  
            os.remove(sharedfolder+'tmp%d%i.csv' %(n,i))  
            if i != 0:  
                req = comm.isend('DisJointPartitionSetRecord%d%i is created' %(n,i), dest=i, tag=211)  
                req.wait()  
  
            #####  
            ##          partitioning in          ##  
            ##          small bags(SB) Algorithm          ##  
            #####  
  
            print("***** USING small bags(SB) METHOD *****")  
            #create random file  
            r = np.size(data,0) % n  
            for i in range(0,n):  
                if r == 0:  
                    m = 0  
                elif r > 0:  
                    m = 1  
                randomdf = random.choices(x,weights=None,k=int(np.size(data,0)/n+m)) #with replacement  
                infile = open(sharedfolder+'tmp%d%i.csv' %(n,i),'w')  
                with infile:  
                    writer = csv.writer(infile)  
                    writer.writerow(randomdf)  
                infile.close()  
                infile = open(sharedfolder+'tmp%d%i.csv' %(n,i),'r')  
                with open(sharedfolder+'SBrecord%d%i.csv' %(n,i), "w") as out_file:  
                    for line in infile:
```

140  
150  
151  
152  
153  
154  
155  
156  
157  
158  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234

```

        if line.strip():
            out_file.write(line)
        out_file.close()
    infile.close()
    os.remove(sharedfolder+'tmp%d%i.csv' %(n,i))
    out_file.close()
    if i != 0:
        req = comm.isend('SBRecord%d%i.csv is created' %(n,i), dest=i, tag=212)
        req.wait()

#####
##           partitioning in           ##
##           disjoint Bags(DB) Algorithm ##
#####

print("***** USING disjoint Bags(DB) METHOD *****")
#create random file
randommf = random.sample(x,np.size(data,0)) #without replacement
myarray1=np.array([randommf[i:n] for i in range(n)])
for i in range(0,n):
    myarray2 =np.append(myarray1[i], (random.choices(myarray1[i], weights=None, k=int(np.size(myarray1[i])))),
axis = 0)
#print(myarray2)
infile = open(sharedfolder+'tmp%d%i.csv' %(n,i),'w')
with infile:
    writer = csv.writer(infile)
    writer.writerow(myarray2)
    infile.close()
    infile = open(sharedfolder+'tmp%d%i.csv' %(n,i),'r')
    with open(sharedfolder+'DBRecord%d%i.csv' %(n,i), "w") as out_file:
        for line in infile:
            if line.strip():
                out_file.write(line)
    infile.close()
os.remove(sharedfolder+'tmp%d%i.csv' %(n,i))
out_file.close()
if i != 0:
    req = comm.isend('DBRecord%d%i.csv is created' %(n,i), dest=i, tag=213)
    req.wait()

#####
##           partitioning in           ##
##           No Replication small bags(NRSB) Algorithm ##
#####

print("***** USING No Replication small bags(NRSB) METHOD *****")
#create random file
r = np.size(data,0) % n
for i in range(0,n):
    if r == 0:
        k = 0
    elif r > 0:
        k = 1
    randommf = random.sample(x,int(np.size(data,0)/n+k)) #with replacement
    infile = open(sharedfolder+'tmp%d%i.csv' %(n,i),'w')
    with infile:
        writer = csv.writer(infile)
        writer.writerow(randommf)
        infile.close()
        infile = open(sharedfolder+'tmp%d%i.csv' %(n,i),'r')
        with open(sharedfolder+'NRSBRecord%d%i.csv' %(n,i), "w") as out_file:
            for line in infile:
                if line.strip():
                    out_file.write(line)
        infile.close()
    os.remove(sharedfolder+'tmp%d%i.csv' %(n,i))
    out_file.close()
    if i != 0:
        req = comm.isend('NRSBRecord%d%i.csv is created' %(n,i), dest=i, tag=214)
        req.wait()

#print('I am master%d'%rank)
#input("Press Enter to continue...")
Tend0 = time.time()
T0 = Tend0 - Tbegingc0
Print ("Run-time is :",T0)

else :

req = comm.irecv(source=0, tag=211)
data = req.wait()
print(rank,data)
req = comm.irecv(source=0, tag=212)
data = req.wait()
print(rank,data)
req = comm.irecv(source=0, tag=213)
data = req.wait()
print(rank,data)

```

235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320

```
req = comm irecv(source=0, tag=214)
data = req.wait()
print(rank,data)
for n in partno:
```

### 3. Training subsystem :

```
#####
##           Training in           ##
##           disjoint Partition(D) Algorithm           ##
#####
```

```
# Bagging Algorithms
# load data
print('rank is:'.rank)
dataframe = read_csv(sharedfolder+'DisJointPartitionSetRecord%d%i.csv' %(n_rank),names= names)
array = dataframe.values
X = array[:,7:43]
Y = array[:,44]
seed = 7
num_trees = 100
max_features = 3
kfold = KFold(n_splits=10, random_state=seed)
# Bagged Decision Trees model
#cart = DecisionTreeClassifier()
model = BaggingClassifier(base_estimator=Perceptron(max_iter = 5000), n_estimators=num_trees)
resultsD = cross_val_score ( model, X, Y, cv=kfold,scoring='accuracy' )
meanD.append ( resultsD.mean() )
stdD.append(resultsD.std())
semD.append(resultsD.std()/math.sqrt(10))
print('***** USING disjoint Partition(D) METHOD *****')
print("Accuracy Bagged Decision Trees on Processor %i : %f " %(rank,resultsD.mean()))
print('meanDB on %d-D disjoint Partition(D) on Processor %i: %o(n_rank),meanD)
print('stdDB on %d-D disjoint Partition(D) on Processor %i: %o(n_rank),stdD)
print('semDB on %d-D disjoint Partition(D) on Processor %i: %o(n_rank),semD)
if rank!=0:
    req = comm isend(meanD, dest=0, tag=int(str(n) + str(rank)+str(5)))
    req.wait()
    req = comm isend(stdD, dest=0, tag=int(str(n) + str(rank)+str(6)))
    req.wait()
    req = comm isend(semD, dest=0, tag=int(str(n) + str(rank)+str(7)))
    req.wait()
```

```
#####
##           Training in           ##
##           small bags(SB) Algorithm           ##
#####
```

```
# Bagging Algorithms
# load data
dataframe = read_csv(sharedfolder+'SBRecord%d%i.csv' %(n_rank),names= names)
array = dataframe.values
X = array[:,7:43]
Y = array[:,44]
seed = 7
num_trees = 100
max_features = 3
kfold = KFold(n_splits=10, random_state=seed)
# Bagged Decision Trees Model
#cart = DecisionTreeClassifier()
model = BaggingClassifier(base_estimator=Perceptron(max_iter = 5000), n_estimators=num_trees)
resultsSB = cross_val_score(model, X, Y, cv=kfold, scoring='accuracy')
meanSB.append(resultsSB.mean())
stdSB.append ( resultsSB.std() )
semSB.append ( resultsSB.std()/math.sqrt(10))
print('***** USING small bags(SB) METHOD *****')
print("Accuracy Bagged Decision Trees on Processor %i : %f " %(rank,resultsSB.mean()))
print('meanDB on %d-D small bags(SB) on Processor %i: %o(n_rank),meanSB)
print('stdDB on %d-D small bags(SB) on Processor %i: %o(n_rank),stdSB)
print('semDB on %d-D small bags(SB) on Processor %i: %o(n_rank),semSB)
if rank!=0:
    req = comm isend(meanSB, dest=0, tag=int(str(n) + str(rank)+str(8)))
    req.wait()
    req = comm isend(stdSB, dest=0, tag=int(str(n) + str(rank)+str(9)))
    req.wait()
    req = comm isend(semSB, dest=0, tag=int(str(n) + str(rank)+str(10)))
    req.wait()
```

```
#####
##           Training in           ##
##           No Replication small bags(NRSB) Algorithm           ##
#####
```

```
# Bagging Algorithms
# load data
```



321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406

```

dataframe = read_csv(sharedfolder+NRSBRecord%d%d.csv' %(n,rank) , names = names)
array = dataframe.values
X = array[:,7:43]
Y = array[:,44]
seed = 7
num_trees = 100
max_features = 3
kfold = KFold(n_splits=10, random_state=seed)
# Bagged Decision Trees model
#cart = DecisionTreeClassifier()
model = BaggingClassifier(base_estimator=Perceptron(max_iter = 5000), n_estimators=num_trees)
resultsNRSB = cross_val_score(model, X, Y, cv=kfold, scoring='accuracy')
meanNRSB.append(resultsNRSB.mean())
stdNRSB.append(resultsNRSB.std())
semNRSB.append(resultsNRSB.std()/math.sqrt(10))
print("***** USING No Replication small bags(NRSB) METHOD *****")
print("Accuracy Bagged Decision Trees on Processor %i : %f" %(rank,resultsNRSB.mean()))
print('meanDB on %d-D No Replication small bags(NRSB) on Processor %i: %%(n,rank),meanNRSB)
print('stdDB on %d-D No Replication small bags(NRSB) on Processor %i: %%(n,rank),stdNRSB)
print('semDB on %d-D No Replication small bags(NRSB) on Processor %i: %%(n,rank),semNRSB)
if rank!=0:
    req = comm.isend(meanNRSB, dest=0, tag=int(str(n) + str(rank)+str(11)))
    req.wait()
    req = comm.isend(stdNRSB, dest=0, tag=int(str(n) + str(rank)+str(12)))
    req.wait()
    req = comm.isend(semNRSB, dest=0, tag=int(str(n) + str(rank)+str(13)))
    req.wait()

```

```

#####
##           Training in           ##
##           disjoint Bags(DB) Algorithm           ##
#####

```

```

# Bagging Algorithms
# load data
dataframe = read_csv(sharedfolder+DBRecord%d%d.csv' %(n,rank),names= names)
array = dataframe.values
X = array[:,7:43]
Y = array[:,44]
seed = 7
num_trees = 100
max_features = 3
kfold = KFold(n_splits=10, random_state=seed)
# Bagged Decision Trees model
#cart = DecisionTreeClassifier()
model = BaggingClassifier(base_estimator=Perceptron(max_iter = 5000), n_estimators=num_trees)
resultsDB = cross_val_score(model, X, Y, cv=kfold, scoring='accuracy')
meanDB.append(resultsDB.mean())
stdDB.append(resultsDB.std())
semDB.append(resultsDB.std()/math.sqrt(10))
print("***** USING disjoint Bags(DB) METHOD *****")
print("Accuracy Bagged Decision Trees on Processor %i : %f" %(rank,resultsDB.mean()))
print('meanDB on %d-D disjoint Bags(DB) on Processor %i: %%(n,rank),meanDB)
print('stdDB on %d-D disjoint Bags(DB) on Processor %i: %%(n,rank),stdDB)
print('semDB on %d-D disjoint Bags(DB) on Processor %i: %%(n,rank),semDB)
if rank!=0:
    req = comm.isend(meanDB, dest=0, tag=int(str(n) + str(rank)+str(14)))
    req.wait()
    req = comm.isend(stdDB, dest=0, tag=int(str(n) + str(rank)+str(15)))
    req.wait()
    req = comm.isend(semDB, dest=0, tag=int(str(n) + str(rank)+str(16)))
    req.wait()

```

#### 4. Generate outputs:

```

if rank== 0 :
#####
##           Receive all outputs           ##
##           (meanD,meanDB,meanSB,meanNRSB)           ##
##           (stdD,stdDB,stdSB,stdNRSB)           ##
##           (semD,semDB,semSB,semNRSB)           ##
##           from tasks > 0           ##
#####

for i in range(1,n):
    req = comm.irecv(source=i, tag=int(str(n) + str(i)+str(5)))
    datameanD = req.wait()
    meanD.append(datameanD)
    print("datameanD : ",rank,datameanD)
    req = comm.irecv(source=i, tag=int(str(n) + str(i)+str(6)))
    datastdD = req.wait()
    stdD.append(datastdD)
    print("datastdD : ",rank,datastdD)
    req = comm.irecv(source=i, tag=int(str(n) + str(i)+str(7)))
    datasemD = req.wait()
    semD.append(datasemD)

```

```

407     print("dataseM : ",rank_dataseM)
408     print("*****")
409     #*****
410     req = comm irecv(source=i, tag=int(str(n) + str(i)+str(8)))
411     datameanSB = req.wait()
412     meanSB.append(datameanSB)
413     print("datameanSB : ",rank_datameanSB)
414     req = comm irecv(source=i, tag=int(str(n) + str(i)+str(9)))
415     datastdSB = req.wait()
416     stdSB.append(datastdSB)
417     print("datastdSB : ",rank_datastdSB)
418     req = comm irecv(source=i, tag=int(str(n) + str(i)+str(10)))
419     dataseM = req.wait()
420     semSB.append(dataseM)
421     print("dataseM : ",rank_dataseM)
422     print("*****")
423     #*****
424     req = comm irecv(source=i, tag=int(str(n) + str(i)+str(11)))
425     datameanNRSB = req.wait()
426     meanNRSB.append(datameanNRSB)
427     print("datameanNRSB : ",rank_datameanNRSB)
428     req = comm irecv(source=i, tag=int(str(n) + str(i)+str(12)))
429     datastdNRSB = req.wait()
430     stdNRSB.append(datastdNRSB)
431     print("datastdNRSB : ",rank_datastdNRSB)
432     req = comm irecv(source=i, tag=int(str(n) + str(i)+str(13)))
433     dataseMNRSB = req.wait()
434     semNRSB.append(dataseMNRSB)
435     print("dataseMNRSB : ",rank_dataseMNRSB)
436     print("*****")
437     #*****
438     req = comm irecv(source=i, tag=int(str(n) + str(i)+str(14)))
439     datameanDB = req.wait()
440     meanDB.append(datameanDB)
441     print("datameanDB : ",rank_datameanDB)
442     req = comm irecv(source=i, tag=int(str(n) + str(i)+str(15)))
443     datastdDB = req.wait()
444     stdDB.append(datastdDB)
445     print("datastdDB : ",rank_datastdDB)
446     req = comm irecv(source=i, tag=int(str(n) + str(i)+str(16)))
447     dataseMDB = req.wait()
448     semDB.append(dataseMDB)
449     print("dataseMDB : ",rank_dataseMDB)
450     print("*****")
451     Tend0 = time.time()
452     T0 = Tend0 - Tbegin0
453     Print ("Run-time is :",T0)
454

```

## 5. Save outputs in Excel file:

```

455
456
457     #####
458     ##      save outputs in      ##
459     ##      EXCEL file          ##
460     #####
461
462     workbook = xlswriter.Workbook('/home/azi/cloud/NN-Perceptron/Comapre methods by NN-Perceptron-Multinode-
463     random_%i.xlsx' %n)
464     worksheetn = workbook.add_worksheet()
465     #currency_format = workbook.add_format({'num_format': '$#,##0'})
466     # Some sample data for the table.
467     data = []
468     for i in range(n):
469
470         data.append(['RANDOM', n, i, 'D', 'MLP',str(meanD[i]), str(stdD[i]), str(semD[i]) ])
471         data.append(['RANDOM', n, i, 'DB', 'MLP',str(meanDB[i]), str(stdDB[i]), str(semDB[i]) ])
472         data.append(['RANDOM', n, i, 'SB', 'MLP',str(meanSB[i]), str(stdSB[i]), str(semSB[i]) ])
473         data.append(['RANDOM', n, i, 'NRSB', 'MLP',str(meanNRSB[i]),str(stdNRSB[i]), str(semNRSB[i]) ])
474
475     caption = 'single-node Ensemble and Classifier.'
476     worksheetn.set_column('B:K', 12)
477     worksheetn.write('B1', caption)
478     worksheetn.add_table('B3:I60', {'data': data,
479     'style': 'Table Style Light 11',
480     'columns': [{'header': 'ORDER'},
481     {'header': 'NODE #'},
482     {'header': 'Part #'},
483     {'header': 'partitioning Method'},
484     {'header': 'Classifier'},
485     {'header': 'Accuracy'},
486     {'header': 'Standard Deviation'},
487     {'header': 'Standard Error of Mean(SEM)},
488     ]})
489
490     workbook.close()
491
492

```

493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513

## 6. Plot drawing:

```
#####
##      plot drawing      ##
#####

import matplotlib.pyplot as plt
print('datameanD is ',datameanD)
fig = plt.figure()
ax = fig.add_subplot(111)
plt.xlabel('Partitions/Bags')
plt.ylabel('Mean Acc+- SEM')
plt.title('single-node multilayer perceptron')
plt.errorbar(n-0.2,datameanD,datasemD,marker='*',linestyle=None,color='g',label='D')
plt.errorbar(partno,datameanDB,datasemDB,marker='o',linestyle=None,color='r',label='DB')
plt.errorbar(np.asarray(partno)+0.2,datameanSB,datasemSB,marker='s',linestyle=None,color='b',label='SB')

plt.errorbar(np.asarray(partno)+0.4,datameanNRSB,datasemNRSB,marker='_',linestyle=None,color='y',label='NRSB')
plt.legend(loc='lower right')
plt.savefig('/home/azi/cloud/NN-Perceptron/Total%imeanComparison.png%n')
plt.show()
```

## 2) Accuracy for Bagging-like methods by CART ( Python source code )

All sections are the same with the MLP classification. The differences are coded in the following lines.

### 1. Definitions and inputs:

1  
2  
3  
4  
5  
6  
7

```
from sklearn.tree import DecisionTreeClassifier
```

### 2. Training subsystem:

```
CART1 = DecisionTreeClassifier()
modelCART = BaggingClassifier ( base_estimator = CART1, n_estimators = num_trees, random_state = seed)
```

## 3) DELS system for Bagging-like methods by MLP:

### 1. Definitions and inputs:

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Oct 5 15:10:40 2019

@author: azi
"""

#####
##### Multi Node Ensemble of #####
##### Classifiers building or using #####
##### Multi-layer perceptron classification #####
##### and D, DB, SB, NRSB data devision method #####
#####

import random
import csv
import os
import time
import numpy as np
from mpi4py import MPI
from pandas import read_csv

# Importing Bagged Decision Trees
From sklearn.model_selection import KFold
From sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from joblib import dump
from joblib import load
from sklearn.ensemble import VotingClassifier
import xlswriter
Tbegin0 = time.time()
comm = MPI.COMM_WORLD
rank = comm.rank
size = comm.size
name = MPI.Get_processor_name()
A = 0.0
B = 0.0
```

39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123

```
C = 0.0
#####
##      DELS inputs are      ##
##      1-input dataset(kddcup 1999) in filename variable      ##
##      2-number of partitionsets in partno array      ##
##      3-classification algorithm is MLP set in model variable      ##
##      in subsection 3.training subsystem      ##
## (model = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 2), random_state=2)) ##
#####
sharedfolder = '/home/azi/cloud/singleNode/bagging/'
sharedfolderC = '/home/azi/cloudC1/singleNode/bagging/'
modelfolder = '/home/azi/cloud/singleNode/model/'
modelfolderC = '/home/azi/cloudC1/singleNode/model/'
foldername = '/home/azi/cloud/singleNode/data/'

filename = foldername+ 'kddcup.data.corrected-sample.csv'
partno= [2,4,6,8,10,12] # Array, number of partition sets
#####
## dataset columns name. Last column(class) is the labeling criteria.      ##
## class type: back,buffer_overflow,ftp_write,guess_passwd,imap,      ##
## ipsweep,land,loadmodule,multihop,neptune,nmap,normal,perl,phf,      ##
## pod,portsweep,rootkit,satan,smurf,spy,teardrop,warezclient,      ##
## warezmaster.      ##
#####

names = ['rows','num','duration','protocol_type','service','flag','src_bytes','dst_bytes','land',
'wrong_fragment','urgent','hot','num_failed_logins','logged_in','num_compromised',
'root_shell','su_attempted','num_root','num_file_creations','num_shells','num_access_files',
'num_outbound_cmds','is_host_login','is_guest_login','count','srv_count','serror_rate',
'srv_serror_rate','error_rate','srv_error_rate','same_srv_rate','diff_srv_rate',
'srv_diff_host_rate','dst_host_count','dst_host_srv_count','dst_host_same_srv_rate',
'dst_host_diff_srv_rate','dst_host_same_src_port_rate','dst_host_srv_diff_host_rate',
'dst_host_serror_rate','dst_host_srv_serror_rate','dst_host_error_rate',
'dst_host_srv_error_rate','class']
```

## 2. Input/Output Subsystem (IOS):

```
#####
###      Create training & testing dataset      ##
### input dataset (kddcup.data.corrected-sample)      ##
### is used to split into 75% for training & 25% for testing      ##
#####

#for line in filename:
# r = random.random()
# if r > 0.75:
# fTestRandom.write(line)
# else:
# fTrainRandom.write(line)
#
#fTrainRandom.close()
#fTestRandom.close()
fTrainRandom = foldername+'kddcup.data.corrected.ordered.Random75%.csv'
fTestRandom = foldername+'kddcup.data.corrected.ordered.Random25%.csv'
```

## 3. Task partitioning:

```
#####
###      The task partitioning subsystem Calculates the      ##
### processors needed to divide by partno on each laptop.      ##
### Two-thirds of the tasks are allocated to the first laptop      ##
### and one-third to the second laptop.      ##
### MN is number of master node processors      ##
### CN is number of client node processors      ##
### Tasks are distributed across the processors by MPI.      ##
#####

if rank == 0: # this is master processor
for n in partno:

TmdlDc0A = []
TmdlDbc0A = []
TmdlSbc0A = []
TmdlNRSbc0A = []
TFTPDc0A = []
TFTPDbc0A = []
TFTPSbc0A = []
TFTPNRSbc0A = []

filename = fTrainRandom
print("partition number is :",n)
CN,r = divmod(n,3)
if (CN < 1 and n != 1):
MN = 1
```

124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209

```

    CN = 1
elif n == 1:
    MN = 1
    CN = 0
else:
    MN = n - CN
print('%i DataSet(s) are used in Client and %i in Master Node' %(CN,MN))

```

#### 4. Partitioning subsystem:

```

#####
###      The Training dataset is partitioned in number of  ##
###      partno in several sequential splits by Bagging-like mthds ##
###      and distributed across the processors (MPI tasks) in the ##
###      number of MN and CN. ##
###      Reading Training data for partitioning ##
###      in 4 different techniques ##
###      (partitioning/Bagging) ##
###      1-disjoint Partition(D) ##
###      2-small bags(SB) ##
###      3-disjoint Bags(DB) ##
###      4-No Replication small bags(NRSB) ##
#####

Tbegin0 = time.time()
print(Tbegin0)

raw_data = open(filename, 'r')
reader = csv.reader(raw_data, delimiter=',', quoting=csv.QUOTE_NONE)
x = list(reader)
data = np.array(x).astype('str')

#####
##      partitioning in ##
##      disjoint Partition(D) Algorithm ##
#####

print('***** USING disjoint Partition(D) METHOD For %i partitions *****\n')
#create random file
randomdf = random.sample(x,np.size(data,0)) # without replacement
p=[randomdf[i::n] for i in range(n)]

for j in range(0,n):

    in_file = open(sharedfolder+'tmp%d%i.csv' %(n,j),'w')
    with in_file:
        writer = csv.writer(in_file)
        writer.writerows(p[j])
        in_file.close()

    in_file = open(sharedfolder+'tmp%d%i.csv' %(n,j),'r')
    with open(sharedfolder+'DisJointPartitionSetRecord%d%i.csv' %(n,j), "w") as out_file:
        for line in in_file:
            if line.strip():
                out_file.write(line)
        in_file.close()
        out_file.close()
    os.remove(sharedfolder+'tmp%d%i.csv' %(n,j))

TendDc0 = time.time()
TDc0 = TendDc0 - Tbegin0

print('*****')
print('      Totaly %i seconds for disjoint partitioning of data in Master node ' %TDc0)

#####
##      partitioning in ##
##      small bags(SB) Algorithm ##
#####

print('***** USING small bags(SB) METHOD For %i partitions *****\n')
TbeginSBc0 = time.time()
#create random file
r = np.size(data,0) % n
for j in range(0,n):
    if r == 0:
        m = 0
    elif r > 0:
        m = 1
    randomdf = random.choices(x,weights=None,k=int(np.size(data,0)/n+m)) #with replacement
    infile = open(sharedfolder+'tmp%d%i.csv' %(n,j),'w')
    with infile:
        writer = csv.writer(infile)
        writer.writerows(randomdf)
        infile.close()
    infile = open(sharedfolder+'tmp%d%i.csv' %(n,j),'r')
    with open(sharedfolder+'SBRecord%d%i.csv' %(n,j), "w") as out_file:
        for line in infile:

```

```

210         if line.strip():
211             out_file.write(line)
212     infile.close()
213     os.remove(sharedfolder+'tmp%d%i.csv' %(n,j))
214     out_file.close()
215
216     TendSBc0 = time.time()
217     TSBc0 = TendSBc0 - TbeginSBc0
218     print('*****')
219     print('        Totally %i second for small bags data in Master node ' %TSBc0)
220
221     #####
222     ##          partitioning in          ##
223     ##          disjoint Bags(DB) Algorithm      ##
224     #####
225
226     print('***** USING disjoint Bags(DB) METHOD For %i partitions *****\n')
227     TbeginDBc0 = time.time()
228     #create random file
229     randomdf = random.sample(x,np.size(data,0)) #without replacement
230     myarray1=np.array([randomdf[1:n] for l in range(n)])
231     for j in range(0,n):
232         myarray2 =np.append(myarray1[j], (random.choices(myarray1[j], weights=None, k=100)), axis = 0)
233         #print(myarray2)
234         infile = open(sharedfolder+'tmp%d%i.csv' %(n,j),'w')
235         with infile:
236             writer = csv.writer(infile)
237             writer.writerow(myarray2)
238             infile.close()
239         infile = open(sharedfolder+'tmp%d%i.csv' %(n,j),'r')
240         with open(sharedfolder+'DBRecord%d%i.csv' %(n,j), "w") as out_file:
241             for line in infile:
242                 if line.strip():
243                     out_file.write(line)
244             infile.close()
245         os.remove(sharedfolder+'tmp%d%i.csv' %(n,j))
246         out_file.close()
247
248     TendDBc0 = time.time()
249     TDBc0 = TendDBc0 - TbeginDBc0
250     print('*****')
251     print('        Totally %i second for small bags data in Master node ' %TDBc0)
252
253     #####
254     ##          partitioning in          ##
255     ##          No Replication small bags(NRSB) Algorithm      ##
256     #####
257
258     print('***** USING No Replication small bags(NRSB) METHOD For %i partitions *****\n')
259     TbeginNRSBc0 = time.time()
260     #create random file
261     r = np.size(data,0) % n
262     for j in range(0,n):
263         if r == 0:
264             k = 0
265         elif r > 0:
266             k = 1
267         randomdf = random.sample(x,int(np.size(data,0)/n+k)) #with replacement
268         infile = open(sharedfolder+'tmp%d%i.csv' %(n,j),'w')
269         with infile:
270             writer = csv.writer(infile)
271             writer.writerow(randomdf)
272             infile.close()
273         infile = open(sharedfolder+'tmp%d%i.csv' %(n,j),'r')
274         with open(sharedfolder+'NRSBRecord%d%i.csv' %(n,j), "w") as out_file:
275             for line in infile:
276                 if line.strip():
277                     out_file.write(line)
278             infile.close()
279         os.remove(sharedfolder+'tmp%d%i.csv' %(n,j))
280         out_file.close()
281
282     TendNRSBc0 = time.time()
283     TNRSBc0 = TendNRSBc0 - TbeginNRSBc0
284     print('*****')
285     print('        Totally %i second for small bags data in Master node ' %TNRSBc0)
286
287     print('Total Time for partitioning is : ',TDC0+TSBc0+TDBc0+TNRSBc0)
288     print('Bagging Files are ready to transfer to Client(s) ... ')
289
290     #####
291     ###          send message to other tasks that files are ready for next processes          ###
292     ###          save the time of file transfering in variables          ###
293     #####
294
295     for t in range (1,n):

```

296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381

```

print('t is: ', t)
fpreq = comm.isend('Files are ready to transfer to Client(s)', dest=t, tag=1357)
fpreq.wait()
print('bagging file is ready to transfer')
TFTPDc0A.append(('Node0',0))
TFTPDc0A.append(('Node0',0))
TFTPSBc0A.append(('Node0',0))
TFTPNRSBc0A.append(('Node0',0))
for i in range(MN,n):
    reqTFTPc0 = comm.recv(source=i, tag= 4044)
    FTPT = reqTFTPc0 / 4
    TFTPDc0A.append(('Node%i'%i,FTPT))
    TFTPDc0A.append(('Node%i'%i,FTPT))
    TFTPSBc0A.append(('Node%i'%i,FTPT))
    TFTPNRSBc0A.append(('Node%i'%i,FTPT))

```

## 5. Training subsystem ( in master task ) :

```

#####
##      Training in master,      ##
##      disjoint Partition(D) Algorithm      ##
#####

```

```

TbeginDc0 = time.time()
fbagname = sharedfolder+'DisJointPartitionSetRecord%d%i.csv' %(n,rank)

dataframeD = read_csv(fbagname, names=names)
arrayD = dataframeD.values
XD = arrayD[:,7:43]
Y = arrayD[:,44]
YD = Y.astype('str')
X_train, X_test, Y_train, Y_test = train_test_split(XD, YD, test_size=0.01, random_state=9)
# Fit the model on 1%
modelD = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 2), random_state=2)
modelD.fit(X_train, Y_train)
# save the model to disk
fbagname = 'MLPjoblib_model_D%d%i.sav' %(n,rank)
dump(modelD, modelfolder+fbagname)
TendDc0 = time.time()
TmdlDc0 = TendDc0 - TbeginDc0
TmdlDc0A.append(('Node%i'%rank,TmdlDc0))

```

```

#####
##      Training in master,      ##
##      small bags(SB) Algorithm      ##
#####

```

```

TbeginSBc0 = time.time()
fbagname = sharedfolder+'SBRecord%d%i.csv' %(n,rank)

dataframeSB = read_csv(fbagname, names=names)
arraySB = dataframeSB.values
XSB = arraySB[:,7:43]
Y = arraySB[:,44]
YSB = Y.astype('str')
X_train, X_test, Y_train, Y_test = train_test_split(XSB, YSB, test_size=0.01, random_state=9)
# Fit the model on 1%
modelSB = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 2), random_state=2)
modelSB.fit(X_train, Y_train)
# save the model to disk
fbagname = 'MLPjoblib_model_SB%d%i.sav' %(n,rank)
dump(modelSB, modelfolder+fbagname)
TendSBc0 = time.time()
TmdlSBc0 = TendSBc0 - TbeginSBc0
TmdlSBc0A.append(('Node%i'%rank,TmdlSBc0))

```

```

#####
##      Training in master,      ##
##      No Replication small bags(NRSB) Algorithm      ##
#####

```

```

TbeginNRSBc0 = time.time()
fbagname = sharedfolder+'NRSBRecord%d%i.csv' %(n,rank)

dataframeNRSB = read_csv(fbagname, names = names)
arrayNRSB = dataframeNRSB.values
XNRSB = arrayNRSB[:,7:43]
Y = arrayNRSB[:,44]
YNRSB = YNRSB.astype('str')
X_train, X_test, Y_train, Y_test = train_test_split(XNRSB, YNRSB, test_size=0.01, random_state=9)
# Fit the model on 1%
modelNRSB = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 2), random_state=2)
modelNRSB.fit(X_train, Y_train)
# save the model to disk
fbagname = 'MLPjoblib_model_NRSB%d%i.sav' %(n,rank)
dump(modelNRSB, modelfolder+fbagname)
TendNRSBc0 = time.time()

```

```

382 TmdlNRSBc0 = TendNRSBc0 - TbeginNRSBc0
383 TmdlNRSBc0A.append(('Node%i'%rank,TmdlNRSBc0))
384
385 #####
386 ## Training in master, ##
387 ## disjoint Bags(DB) Algorithm ##
388 #####
389
390 TbeginDBc0 = time.time()
391 fbagname = sharedfolder+'DBRecord%d%i.csv' %(n,rank)
392
393 dataframeDB = read_csv ( fbagname, names = n_ames)
394 arrayDB = dataframeDB.values
395 XDB = arrayDB[:,7:43]
396 Y = arrayDB[:,44]
397 YDB = YDB.astype('str')
398 X_train, X_test, Y_train, Y_test = train_test_split ( XDB, YDB , test_size=0.01, random_state=9)
399 # Fit the model on 1%
400 modelDB = MIPClassifier ( solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 2), random_state=2)
401 modelDB.fit ( X_train, Y_train)
402 # save the model to disk
403 fbagname = 'MLPjoblib_model_DB%d%i.sav' %(n,rank)
404 dump(model, modelfolder+fbagname)
405 TendSBc0 = time.time()
406 TmdlDBc0 = TendSBc0 - TbeginDBc0
407 TmdlDBc0A.append(('Node%i'%rank,TmdlDBc0))
408
409 for i in range(1,n):
410 reqmodM = comm.irecv(source=i, tag= 1346)
411 data1 = reqmodM.wait()
412 reqTmdlDc0 = comm.recv(source=i, tag= 4040)
413 reqTmdlSBc0 = comm.recv(source=i, tag= 4041)
414 reqTmdlNRSBc0 = comm.recv(source=i, tag= 4042)
415 reqTmdlDBc0 = comm.recv(source=i, tag= 4043)
416 TmdlDc0A.append(('Node%i'%i,reqTmdlDc0))
417 TmdlDBc0A.append(('Node%i'%i,reqTmdlDBc0))
418 TmdlSBc0A.append(('Node%i'%i,reqTmdlSBc0))
419 TmdlNRSBc0A.append(('Node%i'%i,reqTmdlNRSBc0))

```

## 6. Testing subsystem:

```

420 #####
421 ##### Vote-based Ensemble Step (Major-Voting) : In Master Task #####
422 #####
423
424
425
426
427
428 print('Model is received from clients')
429 TbeginENSM = time.time()
430
431 dataframeV = read_csv ( fTrainRandom, names = names)
432 arrayV = dataframeV.values
433 XV = arrayV[:,7:43]
434 Y = arrayV[:,44]
435 YV = YV.astype('str')
436 kfold = Kfold ( n_splits =10, random_state=7 )
437 # create the sub models for vote-based ensemble
438
439 estimatorsD = []
440 estimatorsDB = []
441 estimatorsSB = []
442 estimatorsNRSB = []
443
444 for j in range(0,n):
445 print('j is:',j)
446 mfD = modelfolder+'MLPjoblib_model_D%d%i.sav' %(n,j)
447 mfDB = modelfolder+'MLPjoblib_model_DB%d%i.sav' %(n,j)
448 mfSB = modelfolder+'MLPjoblib_model_SB%d%i.sav' %(n,j)
449 mfNRSB = modelfolder+'MLPjoblib_model_NRSB%d%i.sav' %(n,j)
450
451 modelD = load(mfD)
452 modelDB = load(mfDB)
453 modelSB = load(mfSB)
454 modelNRSB = load(mfNRSB)
455
456 estimatorsD.append(('modelD%i' %j, modelD) )
457 estimatorsDB.append(('modelDB%i'%j, modelDB))
458 estimatorsSB.append(('modelSB%i'%j, modelSB))
459 estimatorsNRSB.append(('modelNRSB%i' %j, modelNRSB))
460
461 # create the ensemble model
462
463 ensembleD = VotingClassifier(estimatorsD)
464 ensembleDB = VotingClassifier(estimatorsDB)
465 ensembleSB = VotingClassifier(estimatorsSB)
466 ensembleNRSB = VotingClassifier(estimatorsNRSB)
467
468 resultsD = cross_val_score(ensembleD, X, Y, cv=kfold)
469 resultsDB = cross_val_score(ensembleDB, X, Y, cv=kfold)

```



```

468 resultsSB = cross_val_score(ensembleSB, X, Y, cv=kfold)
469 resultsNRSB = cross_val_score(ensembleNRSB, X, Y, cv=kfold)
470 print("Mean Results of voting in D is :",resultsD.mean())
471 print("ensembleD is :",resultsD)
472 print("Mean Results of voting in DB is :",resultsDB.mean())
473 print("ensembleDB is :",resultsDB)
474 print("Mean Results of voting in SB is :",resultsSB.mean())
475 print("ensembleSB is :",resultsSB)
476 print("Mean Results of voting in NRSB is :",resultsNRSB.mean())
477 print("ensembleNRSB is :",resultsNRSB)
478
479 TendENSM = time.time()
480 TENSM = TendENSM - TbeginENSM
481
482 #####
483 ##           Testing data in master task           ##
484 #####
485
486 dataframeTest = read_csv ( f TestRandom, names = na mes)
487 arrayTest = dataframeTest.values
488 X_test = arrayTest[:,7:41]
489 Y_test = arrayTest[:,41]
490 Y_test = Y_test.astype('str')
491
492 #####
493 ##           Testing in master task           ##
494 ##           disjoint Partition(D) Algorithm           ##
495 #####
496
497 TbeginTESTD = time.time()
498 loaded_modelD = ensembleD.fit(X_test, Y_test)
499 print("TbeginTESTD is :", TbeginTESTD)
500 accuracyD = loaded_modelD.score(X_test, Y_test)
501 print("The Accuracy of disjoint partition(D) is :",accuracyD)
502 ynewD = loaded_modelD.predict(X_test)
503 #print(X_test,ynewD)
504 print("The Predicted result of D method is produced.")
505
506 TendTESTD = time.time()
507 TTESTD = TendTESTD - TbeginTESTD
508
509 #####
510 ##           Testing in master task           ##
511 ##           small bags(SB) Algorithm           ##
512 #####
513
514 loaded_modelSB = ensembleSB.fit(X_test, Y_test)
515 accuracySB = loaded_modelSB.score(X_test, Y_test)
516 print("The Accuracy of Small Bages(SB) is :",accuracySB)
517 ynewSB = loaded_modelSB.predict(X_test)
518 #print(X_test,ynewD)
519 print("The Predicted result of SB method is produced.")
520
521 TendTESTSB = time.time()
522 TTESTSB = TendTESTSB - TendTESTD
523
524 #####
525 ##           Testing in master task           ##
526 ##           No Replication small bags(NRSB) Algorithm           ##
527 #####
528
529 loaded_modelNRSB = ensembleNRSB.fit(X_test, Y_test)
530 accuracyNRSB = loaded_modelNRSB.score(X_test, Y_test)
531 print("The Accuracy of No Replication small bags(NRSB) is :",accuracyNRSB)
532 ynewNRSB = loaded_modelNRSB.predict(X_test)
533 #print(X_test,ynewD)
534 print("The Predicted result of NRSB method is produced.")
535
536 TendTESTNRSB = time.time()
537 TTESTNRSB = TendTESTNRSB - TendTESTSB
538
539 #####
540 ##           Testing in master task           ##
541 ##           disjoint Bages(DB) Algorithm           ##
542 #####
543
544 loaded_modelDB = ensembleDB.fit(X_test, Y_test)
545 accuracyDB = loaded_modelDB.score(X_test, Y_test)
546 print("The Accuracy of disjoint Bages(DB) is :",accuracyDB)
547 ynewDB = loaded_modelDB.predict(X_test)
548 #print(X_test,ynewD)
549 print("The Predicted result of DB method is produced.")
550
551 TendTESTDB = time.time()
552 TTESTDB = TendTESTDB - TendTESTNRSB
553 Tend0 = time.time()
554 T0 = Tend0 - Tbegin0

```

554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638

```
Print ("Run-time is :",T0)
```

## 7. Save outputs in Excel file:

```
#####
##      save outputs in      ##
##      EXCEL file          ##
#####

workbook = xlswriter.Workbook(foldername+ MultiNode-MLP-Random-TestScore-part%i.xlsx' %n)
worksheetn = workbook.add_worksheet()
#currency_format = workbook.add_format({'num_format': '$#,##0'})
# Some sample data for the table.
data = []
print('TFTPDC0A',TFTPDC0A)
print('TmdlDc0A',TmdlDc0A)
for i in range(n):
    #print('i is :',i)
    #print('TFTPDC0A[i]',str(TFTPDC0A[i]))
    #print('TmdlDc0A[i]',str(TmdlDc0A[i]))

    data.append(['RANDOM', n, i, 'D', 'MLP',resultsD.mean(), accuracyD, TDC0, str(TFTPDC0A[i]), str(TmdlDc0A[i]),
TENSMM ])
    data.append(['RANDOM', n, i, 'DB', 'MLP',resultsDB.mean(), accuracyDB, TDBc0, str(TFTPDBc0A[i]),
str(TmdlDBc0A[i]), TENSMM ])
    data.append(['RANDOM', n, i, 'SB', 'MLP',resultsSB.mean(), accuracySB, TSBc0, str(TFTPSC0A[i]),
str(TmdlSBc0A[i]), TENSMM ])
    data.append(['RANDOM', n, i, 'NRSB', 'MLP',resultsNRSB.mean(), accuracyNRSB,TNRSBc0, str(TFTPNSBc0A[i]),
str(TmdlNRSBc0A[i]), TENSMM ])

caption = 'single-node Ensemble and single Classifier.'
worksheetn.set_column('B:L', 12)
worksheetn.write('B1', caption)
worksheetn.add_table('B3:L52', {'data': data,
'style': 'Table Style Light 11',
'columns': [{'header': 'ORDER'},
{'header': 'NODE #'},
{'header': 'Part #'},
{'header': 'partitioning Method'},
{'header': 'Classifier'},
{'header': 'Cross-Val-Score'},
{'header': 'Test-Score'},
{'header': 'Time of partitioning'},
{'header': 'Time of Transferring'},
{'header': 'Time of Training'},
{'header': 'Time of Ensemble Classifier'},
]})

workbook.close()
```

## 8. Task partitioning in other tasks than master:

```
else: # rank is not 0
for n in partno:
print("partition number is :", n)
CN,r = divmod(n,3)
if CN < 1:
MN = 1
CN = 1
else:
MN = n - CN

if rank < MN : # rank is not 0 and is run on ASUS (one-third of tasks)
```

## 9. Training Subsystem in other tasks (one-third of tasks):

```
fpreq = comm.irecv(source=0, tag=1357)
text = fpreq.wait()
print('rank in ASUS is :', rank)

#####
##      Training in first laptop      ##
##      disjoint Partition(D) Algorithm      ##
#####

TbeginDc0 = time.time()
fbagname = sharedfolder+'DisJointPartitionSetRecord%d%i.csv' %(n,rank)

dataframeTrain = read_csv ( fbagname, names = name s )
arrayTrain = dataframeTrain.values
XTrain = arrayTrain[:,7:43]
Y = arrayTrain[:,44]
YTrain = Y.astype('str')
X_train, X_test, Y_train, Y_test = train_test_split ( XTrain, YTrain, test_size=0.33, r andom_state=9)
# Fitt the model on 33%
modelTrain = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), random_state=2)
```

639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724

```

modelTrain.fit(X_train, Y_train)
# save the model to disk
fbagname = 'MLPjoblib_model_D%d%i.sav' %(n,rank)
dump(modelTrain, modelfolder+fbagname)
TendDc0 = time.time()
TmdlDc0 = TendDc0 - TbeginDc0
reqTmdlDc0 = comm.send(TmdlDc0, dest=0, tag= 4040)

#####
## Training in first laptop, ##
## small bags(SB) Algorithm ##
#####

TbeginSBc0 = time.time()
fbagname = sharedfolder+'SBRecord%d%i.csv' %(n,rank)

dataframeSB = read_csv(fbagname, names = na mes)
arraySB = dataframeSB.values
XSB = arraySB[:,7:43]
YSB = arraySB[:,44]
YSB = YSB.astype('str')
X_train, X_test, Y_train, Y_test = train_test_split(XSB, YSB, test_size=0.33, random_state=9)
# Fit the modelSB on 33%
modelSB = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), random_state=2)
model.fit(X_train, Y_train)
# save the model to disk
fbagname = 'MLPjoblib_model_SB%d%i.sav' %(n,rank)
dump(model, modelfolder+ fbagname)
TendSBc0 = time.time()
TmdlSBc0 = TendSBc0 - TbeginSBc0
reqTmdlSBc0 = comm.send(TmdlSBc0, dest=0, tag=4041)

#####
## Training in first laptop, ##
## No Replication small bags(NRSB) Algorithm ##
#####

TbeginNRSBc0 = time.time()
fbagname = sharedfolder+'NRSBRecord%d%i.csv' %(n,rank)

dataframeNRSB = read_csv(fbagname, names = nam es)
arrayNRSB = dataframeNRSB.values
XNRSB = arrayNRSB[:,7:43]
YNRSB = arrayNRSB[:, 44]
YNRSB = YNRSB.astype('str')
X_train, X_test, Y_train, Y_test = train_test_split(XNRSB, YNRSB, test_size=0.33, random_state=9)
# Fit the model NRSB on 33%
modelNRSB = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), random_state=2)
modelNRSB.fit(X_train, Y_train)
# save the model to disk
fbagname = 'MLPjoblib_model_NRSB%d%i.sav' %(n,rank)
dump(modelNRSB, modelfolder+fbagname)
TendNRSBc0 = time.time()
TmdlNRSBc0 = TendNRSBc0 - TbeginNRSBc0
reqTmdlNRSBc0 = comm.send(TmdlNRSBc0, dest=0, tag= 4042)

#####
## Training in first laptop, ##
## disjoint Bags(DB) Algorithm ##
#####

TbeginDBc0 = time.time()
fbagname = sharedfolder+'DBRecord%d%i.csv' %(n,rank)

dataframe = read_csv(fbagname, names=names)
array = dataframe.values
X = array[:,7:43]
Y = array[:,44]
Y = Y.astype('str')
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=9)
# Fit the model on 33%
model = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), random_state=2)
model.fit(X_train, Y_train)
# save the model to disk
fbagname = 'MLPjoblib_model_DB%d%i.sav' %(n,rank)
dump(model, modelfolder+fbagname)
TendDBc0 = time.time()
TmdlDBc0 = TendDBc0 - TbeginDBc0
reqTmdlDBc0 = comm.send(TmdlDBc0, dest=0, tag= 4043)

reqmodM = comm.isend('models', dest=0, tag= 1346)
reqmodM.wait()

10. File transferring:

elif rank < n: # rank is not 0 and is run on SONY

fipreq = comm.irecv(source=0, tag= 1357)
text = fipreq.wait()
print('rank in SONY is :', rank)

```

725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810

```
#####
###      FTP Step : The Training dataats are      #####
###      distributed across the tasks and nodes      #####
#####

print('Bagging Transformation from processor %i is started.' %rank)

TbeginFTP = time.time()
from ftplib import FTP
#domain name or server ip
ftp = FTP('client')
ftp.login(user='azi', passwd = 'cmpe1234')
ftp.cwd('/home/azi/cloudC1/singleNode/bagging/')
os.chdir('/home/azi/cloud/singleNode/bagging/')

def placeFile():
    filename = 'DisJointPartitionSetRecord%d%i.csv' %(n,rank)
    ftp.storbinary('STOR '+filename, open(filename, 'rb'))
    filename = 'SBRecord%d%i.csv' %(n,rank)
    ftp.storbinary('STOR '+filename, open(filename, 'rb'))
    filename = 'DBRecord%d%i.csv' %(n,rank)
    ftp.storbinary('STOR '+filename, open(filename, 'rb'))
    filename = 'NRSBRecord%d%i.csv' %(n,rank)
    ftp.storbinary('STOR '+filename, open(filename, 'rb'))
    ftp.quit()

placeFile()
print('File is Transferred')
TendFTP = time.time()
TFTPc0 = TendFTP - TbeginFTP
reqTFTPc0 = comm.send(TFTPc0, dest=0, tag=4044)
```

### 11. Training Subsystem in other tasks (two-third of tasks):

```
#####
##      Training in second laptop      ##
##      disjoint Partition(D) Algorithm      ##
#####

TbeginDc0 = time.time()
filename = sharedfolderC+'DisJointPartitionSetRecord%d%i.csv' %(n,rank)

dataframeD1 = read_csv(filename, names=names)
array D1 = dataframe.D1.values
X D1 = array D1 [ :,7:43]
Y D1 = array D1 [ :,44]
Y D1 = Y D1.astype('str')
X_train, X_test, Y_train, Y_test = train_test_split(X D1, Y D1, test_size=0.01, ranom_state=9)
# Fit the model D1 on 1%
model D1 = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 2), random_state=2)
model D1.fit(X_train, Y_train)
# save the model D1 to disk
filename D1 = 'MLPjoblib_model_D%d%i.sav' %(n,rank)
dump(model D1, modelfolder+filename)
TendDc0 = time.time()
TmdlDc0 = TendDc0 - TbeginDc0
reqTmdlDc0 = comm.send(TmdlDc0, dest=0, tag= 4040)

#####
##      Training in second laptop,      ##
##      small bags(SB) Algorithm      ##
#####

TbeginSBc0 = time.time()
filename = sharedfolderC+'SBRecord%d%i.csv' %(n,rank)

dataframeSB1 = read_csv(filename, name s=names)
array SB1 = dataframe.SB1.values
X SB1 = array SB1 [ :,7:43]
Y SB1 = array SB1 [ :,44]
Y SB1 = Y SB1.astype('str')
X_train, X_test, Y_train, Y_test = train_test_split(X SB1, Y SB1, test_size=0.01, random_state=9)
# Fit the model SB1 on 1%
model SB1 = MLPClassifier ( s olver='lbfgs', alpha=1 e-5,hidden_layer_sizes=(5, 2), random_state=2)
model SB1.fit(X_train, Y_train)
# save the SB1 model to disk
filename = 'MLPjoblib_model_SB%d%i.sav' %(n,rank)
dump(model SB1 , modelfolder+ filename)
TendSBc0 = tim e.time()
TmdlSBc0 = TendSBc0 - TbeginSBc0
reqTmdlSBc0 = comm.send(TmdlSBc0, dest=0, tag= 4041)

#####
```

```

11                                     ##      Training in second laptop,      ##
12                                     ##      No Replication small bags(NRSB) Algorithm  ##
13                                     #####
14
15 TbeginNRSBc0 = time.time()
16 filename = sharedfolderC+NRSBrecrod%d%d.csv' %(n,rank)
17
18 dataframeNRSB1 = read_csv(filename, names=names)
19 arrayNRSB1 = dataframe NRSB1.values
20 X NRSB1 = array NRSB1[:,7:43]
21 Y NRSB1 = array NRSB1[:,44]
22 Y NRSB1 = Y NRSB1.astype('str')
23 X_train, X_test, Y_train, Y_test = train_test_split(X NRSB1, Y NRSB1, test_size=0.01, random_state=9)
24                                     # Fit the NRSB1 model on 1%
25 model NRSB1 = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), random_state=2)
26 model NRSB1.fit(X_train, Y_train)
27                                     # save the NRSB1 model to disk
28 filename = 'MLPjoblib_model_NRSB%d%i.sav' %(n,rank)
29 dump ( model NRSB1 , modelfolder+filename)
30 TendNRSBc0 = time.time()
31 TmdlNRSBc0 = TendNRSBc0 - TbeginNRSBc0
32 reqTmdlNRSBc0 = comm.send(TmdlNRSBc0, dest=0, tag= 4042)
33
34                                     #####
35                                     ##      Training in second laptop,      ##
36                                     ##      disjoint Bags(DB) Algorithm      ##
37                                     #####
38
39 TbeginDBc0 = time.time()
40 filename = sharedfolderC+DBRecord%d%d.csv' %(n,rank)
41
42 dataframeDB1 = read_csv (filename, names=names)
43 arrayDB1 = dataframe DB1.values
44 X DB1 = array DB1[:,7:43]
45 Y DB1 = array DB1[:,44]
46 Y DB1 = Y DB1.astype('str')
47 X_train, X_test, Y_train, Y_test = train_test_split ( X DB1 , Y DB1 , test_size=0.01, random_state=9)
48                                     # Fit the mode DB1 on 1%
49 model DB1 = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), random_state=2)
50 model DB1.fit ( X_train, Y_train)
51                                     # save the DB1 model to disk
52 Filename = 'MLPjoblib_model_DB%d%i.sav' %(n,rank)
53 dump (model DB1, modelfolder+filename)
54 TendDBc0 = Time.Time ()
55 TmdlDBc0 = TendDBc0 - TbeginDBc0
56 reqTmdlDBc0 = comm.send(TmdlDBc0, dest=0, tag= 4043)
57
58 reqmodM = comm.isend('models', dest=0, tag= 1346)
59 reqmodM.wait()

```

#### 4) DELS system for LADEL model by MLP (Python source code):

1. Definitions and inputs:

```

1
2
3
4
5
6
7
8
9
10 #####
11 ##### 1- divide input dataset into 75% as training & #####
12 #####      25% as testing dataset by LADEL #####
13 ##### 2- partitioning training dataset to number of partno #####
14 #####
15
16 #####
17 ##      LADEL inputs are      ##
18 ##      1-input dataset(HIGGS) in filename variable      ##
19 ##      2-number of partitionsets in partno array      ##
20 #####
21 #!/usr/bin/env python3
22 # -*- coding: utf-8 -*-
23 """
24 Created on Sat Oct 5 15:10:40 2019
25
26 @author: azi
27 """
28
29 import pandas as pd
30 import fnmatch
31 import os
32 import shutil
33 import time
34 import csv
35 import random
36 from mpi4py import MPI
37 from pandas import import read_csv##
38 from sklearn.model_selection import Kfold##
39 from sklearn.model_selection import cross_val_score##
40 from sklearn.model_selection import train_test_split##

```

34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118

```

from sklearn.neural_network import MLPClassifier##
from joblib import dump
from joblib import load
from sklearn.ensemble import VotingClassifier
import xlswriter

comm = MPI.COMM_WORLD
rank = comm.rank
size = comm.size
name = MPI.Get_processor_name()
A = 0.0
B = 0.0
C = 0.0

#import numpy as np
Tbegin = time.time()
SourceFile = '/home/azi/cloud/LADEL/'
LADELFile = '/home/azi/cloud/LADEL/data/'
LADELFileC = '/home/azi/cloudC1/LADEL/data/'
LADELmodel = '/home/azi/cloud/LADEL/model/'
tempfolder = '/home/azi/cloud/LADEL/temp/'
RPTfolder = '/home/azi/cloud/LADEL/report/'

folder25 = '/home/azi/cloud/LADEL/percentage25/'
folder75 = '/home/azi/cloud/LADEL/percentage75/'

filename = '/home/azi/cloud/singleNode/data/HIGGS.data.corrected.ordered.LADEL1%.csv'
partno= [12]# Array, number of partition sets
#partno= [2,4,6,8,10,12] # Array, number of partition sets

#####
## dataset columns name. Last column(class) is the labeling criteria. ##
## class type: back,buffer_overflow,ftp_write,guess_passwd,imap, ##
## ipsweep,land_loadmodule,multihop,neptune,nmap,normal_perl,phf, ##
## pod,portsweep,rootkit,satan,smurf,spy,teardrop,warezclient, ##
## warezmaster. ##
#####

names = ['rows','num','duration','protocol_type','service','flag','src_bytes','dst_bytes','land',
'wrong_fragment','urgent','hot','num_failed_logins','logged_in','num_compromised',
'root_shell','su_attempted','num_root','num_file_creations','num_shells','num_access_files',
'num_outbound_cmds','is_host_login','is_guest_login','count','srv_count','error_rate',
'srv_error_rate','error_rate','srv_error_rate','same_srv_rate','diff_srv_rate',
'srv_diff_host_rate','dst_host_count','dst_host_srv_count','dst_host_same_srv_rate',
'dst_host_diff_srv_rate','dst_host_same_src_port_rate','dst_host_srv_diff_host_rate',
'dst_host_error_rate','dst_host_srv_error_rate','dst_host_retor_rate',
'dst_host_srv_retor_rate','class']

```

## 2. Input/Output Subsystem (IOS) (LADEL):

```

#save start time
Tbeginclass = time.time()
df = pd.read_csv(filename,names= names , low_memory = False)
shutil.rmtree(tempfolder, ignore_errors=True)
os.mkdir(tempfolder)
df.dropna(inplace=True)#drop rows with missing value
df.drop_duplicates(inplace=True) # Drop duplicate values
df_by_class = df.groupby('class').count()
#print('data file by class: ',df_by_class)

#####
#### separate class labels in files #####
#####

for i, g in df.groupby('class'):
    g.to_csv('/home/azi/cloud/LADEL/{}csv'.format(i), header=False, index_label=False)
Tendclass = time.time()
Totalclass = Tendclass-Tbeginclass
print('*****')
print('Begin process time for separte classifications is :%i' %Tbeginclass)
print('End process time for separte classifications is :%i' %Tendclass)
print('*****')
print('        Totaly %i second ' %Totalclass)

for filename in os.listdir(SourceFile):
    if fnmatch.fnmatch(filename, '*.csv'):
        #print(filename)
        fin = open(SourceFile+filename, 'rb')
        f9out = open(folder75+'75'+filename, "wb")
        flout = open(folder25+'25'+filename, "wb")

        for line in fin:
            r = random.random()
            if r > 0.75:
                flout.write(line)
            else:
                f9out.write(line)
        flout.write(line)

```

120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204

```

f9out.write(line)
fin.close()
f9out.close()
fout.close()
list75 = os.listdir(folder75)
combined_csv75 = pd.concat ( [pd.read_csv(folder75+f, header=None, engine='python') for f in list75
],sort=False,axis=0)
combined_csv75.to_csv( LADELFile+'HIGGS.data.corrected.ordered.LADEL75%.csv', header=False)
list25 = os.listdir(folder25)
#print(list25)
combined_csv25 = pd.concat ( [pd.read_csv(folder25+f, header=None, engine='python') for f in list25
],sort=False,axis=0)
combined_csv25.to_csv( LADELFile+'HIGGS.data.corrected.ordered.LADEL25%.csv', header=False)

df = pd.read_csv(LADELFile+'HIGGS.data.corrected.ordered.LADEL75%.csv',names= names , low_memory = False)
for i, g in df.groupby('class'):
    g.to_csv('/home/azi/cloud/LADEL/{i}.csv'.format(i), header=False, index_label=False)

```

### 3. Partitioning subsystem (LADEL):

```

#####
###      The Training dataset is partitioned in number of  ##
###      partno in several sequential splits byLADEL mthds  ##
###      and distributed across the processors (MPI tasks) in the ##
###      number of MN and CN.                                ##
#####

Tbegin = time.time()

for filename in os.listdir(SourceFile):
if filename.endswith('*.csv'):
    fin = open(SourceFile+filename, 'r')
    file = list(csv.reader(fin))
    numline = len(file)
    length = int(numline / partno[0])
    #print('length:',length)
    folds = []
    for i in range(int(partno[0]-1)):
        folds += [file[i]*length: (i+1)*length]
        with open(tempfolder+filename+'%i%.csv' %(partno[0],i), 'w') as fout:
            wr = csv.writer(fout, dialect='excel')
            wr.writerows(folds[i])

    folds += [file[(partno[0]-1)*length: numline]]

    with open(tempfolder+filename+'%i%.csv' %(partno[0],i+1), 'w') as fout:
        wr = csv.writer(fout, dialect='excel')
        wr.writerows(folds[i+1])

for i in range(partno[0]):
    listdata = [f for f in os.listdir(tempfolder) if f.endswith('%i%.csv' %(partno[0],i)) and os.stat(tempfolder+f).st_size >
0]
    combined_csv = pd.concat ( [pd.read_csv(tempfolder+f, header=None, engine='python') for f in listdata
],sort=False,axis=0)
    combined_csv.to_csv( LADELFile+'HIGGS.data.corrected.ordered.LADEL.part%i%.csv'      %(partno[0],i),
header=False)
    Tend = time.time()
    Ttotal = Tend-Tbegin
    print('Begin process time for equal devision is :%i' %Tbegin)
    print('End process time for equal devision is :%i' %Tend)
    print('*****')
    print('      Totaly %i second' %Ttotal)

```

### 4. Task partitioning (LADEL):

```

fLADELSort75 = LADELFile + 'HIGGS.data.corrected.ordered.LADEL75%.csv'
fLADELSort25 = LADELFile + 'HIGGS.data.corrected.ordered.LADEL25%.csv'
fLADELSort1 = 'HIGGS.data.corrected.ordered.LADEL1%.csv'

#####
###      The task partitioning subsystem Calculates the  ##
###      processors needed to divide by partno on each laptop. ##
###      Two-thirds of the tasks are allocated to the first laptop ##
###      and one-third to the second laptop.                    ##
###      MN is number of master node processors                ##
###      CN is number of client node processors                 ##
###      Tasks are distributed across the processors by MPI.    ##
#####

if rank == 0: # this is master processor or node
for n in partno:

    TmdlLADELc0A = []
    TftPLADELc0A = []

```

205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289

```

filename = fLADELSort75
print("partition number is :", n)
CN,r = divmod(n,3)
if (CN < 1 and n != 1):
    MN = 1
    CN = 1
elif n == 1:
    MN = 1
    CN = 0
else:
    MN = n - CN
print("%i Data Set(s) are used in Client and %i in Master Node" %(CN,MN))

#####
###      send message to other tasks that files are ready for next processes      ###
###      save the time of file transferring in variables                          ###
#####

TbeginC0 = time.time()
print(TbeginC0)

for t in range (MN):
    FTPLADELc0A.append(0)
print("bagging file is ready to transfer")

for i in range(MN,n):
    reqTFTPc0 = comm.recv(source=i, tag= 4044)
    FTPT = reqTFTPc0
    FTPLADELc0A.append(FTPT)

```

### 5. Training subsystem in master task (LADEL):

```

#####
##      Training in master,      ##
##      LADEL Algorithm by MLP   ##
#####

TbeginLADELc0 = time.time()
fbagname = LADELFile + 'HIGGS.data.corrected.ordered.LADEL.part%i%i.csv' %(n,rank)

dataframeLADEL = read_csv ( fbagname, names= n a m e s , low_memory = False)
arrayLADEL = dataframeLADEL.values
XLADEL = arrayLADEL [ :,7:43]
YLADEL = arrayLADEL [ :,44]
YLADEL = YLADEL.astype('str')
X_train, X_test, Y_train, Y_test = train_test_split(XLADEL, YLADEL, test_size = 0.01, random_state=9)
# Fit the LADEL model on 1%
modelLADEL = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), random_state=2)
modelLADEL.fit ( X_train, Y_train)
# save the LADEL model to disk
fbagname = 'MLPjoblib_model_LADEL%d%i.sav' %(n,rank)
dump(modelLADEL , LADELmodel+fbagname)
TendLADELc0 = time.time()
TmdlLADELc0 = TendLADELc0 - TbeginLADELc0
TmdlLADELc0A.append((TmdlLADELc0))

for i in range(1,n):
    reqmodM = comm.irecv(source=i, tag= 1346)
    data1 = reqmodM.wait()
    reqTmdlLADELc0 = comm.recv(source=i, tag= 4040)

    TmdlLADELc0A.append((reqTmdlLADELc0))

```

### 6. Testing subsystem (LADEL):

```

#####
#####      Vote-based Ensemble Step (Major-Voting) : In Master Node      #####
#####

print("Model is received from clients")
TbeginENSM = time.time()
dataframeLADEL = read_csv ( fLADELSort75, names= names)
arrayLADEL = dataframeLADEL.values
XLADEL = array LADEL [ :,7:43]
YLADEL = array LADEL [ :,44]
YLADEL = YLADEL.astype('str')
kfold = Kfolds (n_splits=10, random_state=7)
# create the LADEL sub models

estimatorsLADEL = []

for j in range(0,n):

    mflADEL = LADELmodel+'MLPjoblib_model_LADEL%d%i.sav' %(n,j)
    modelLADEL = load(mflADEL)
    estimatorsLADEL.append(('modelLADEL%i' %j, modelLADEL))

# create the LADEL ensemble model

ensembleLADEL = VotingClassifier(estimatorsLADEL)

```



290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374

```

resultsLADEL = cross_val_score ( ensembleLADEL, X, Y, cv=kfold)

print ( 'Mean results of voting in LADEL is :',resultsLADEL.mean())
print(ensemble LADEL is :',resultsLADEL)

TendENSM = time.time()
TENSM = TendENSM - TbeginENSM

#####
###           Testing data in master task           ###
#####

dataframeLADEL = read_csv ( fLADELsort 25, nnames=names, low_memory = False)
arrayLADEL = dataframeLADEL .values
X_test = arrayLADEL [:,7:43]
Y_test = arrayLADEL [:,44]
Y_test = Y_test.astype('str')

#####
##           Testing in master task           ##
##           LADEL Algorithm                 ##
#####

TbeginTESTLADEL = time.time()
loaded_modelLADEL = ensembleLADEL.fit(X_test, Y_test)
print('TbeginTESTLADEL is :', TbeginTESTLADEL)
accuracyLADEL = loaded_modelLADEL.score(X_test, Y_test)
print('The Accuracy of disjoint partition(LADEL) is :',accuracyLADEL)
ynewLADEL = loaded_modelLADEL.predict(X_test)
#print(X_test,ynewLADEL)
print('The Predicted result of LADEL method is produced.')

TendTESTLADEL = time.time()
TTESTLADEL = TendTESTLADEL - TbeginTESTLADEL

7. Save outputs in Excel file:

#####
##           save outputs in                 ##
##           EXCEL file                     ##
#####

workbook = xlswriter.Workbook(RPTfolder+'MultiNode-LADEL-MLP-TestScore-part%i.xlsx' %n)
worksheetn = workbook.add_worksheet()
#currency_format = workbook.add_format({'num_format': '$#,##0'})
# Some sample data for the table.
data = []
print('TFTPLADELc0A',TFTPLADELc0A)
print('TmdlLADELc0A',TmdlLADELc0A)
for i in range(n):

    data.append(['SORTED',          n,          i,'LADEL','MLP',resultsLADEL.mean(),accuracyLADEL,8,
str(TFTPLADELc0A[i]),str(TmdlLADELc0A[i]),TENSM])

caption = 'MULTI-Node Ensemble and single Classifier.'
worksheetn.set_column('B:L', 12)
worksheetn.write('B1', caption)
worksheetn.add_table('B3:L52', {'data': data,
'style': 'Table Style Light 11',
'columns': [{'header': 'ORDER'},
{'header': 'NODE #'},
{'header': 'Part #'},
{'header': 'partitioning Method'},
{'header': 'Classifier'},
{'header': 'SRV Cross-Val-Score'},
{'header': 'SRV Test-Score'},
{'header': 'SRV Time of partitioning'},
{'header': 'Time of Transferring'},
{'header': 'Time of Training'},
{'header': ' Ensemble-Scoring Time'},
]})

workbook.close()

8. Task partitioning in other tasks than master:
else: # rank is not 0
for n in partno:

    CN,r = divmod(n,3)
    if CN < 1:
        MN = 1
        CN = 1
    else:
        MN = n - CN

    if rank < MN : # rank is not 0 and is run on ASUS (one-third of tasks)

```

## 9. Training Subsystem in other tasks (one-third of tasks):

```
#####  
##      Training in first laptop      ##  
##      LADEL Algorithm                ##  
#####  
  
TbeginLADELc0 = time.time()  
fbagname = LADELFile + 'HIGGS.data.corrected.ordered.LADEL.part%i.csv' %(n,rank)  
  
dataframeLADEL = read_csv(fbagname, names=names, low_memory = False)  
arrayLADEL = dataframe.LADEL.values  
XLADEL = arrayLADEL[:,7:43]  
YLADEL = arrayLADEL[:,44]  
YLADEL = YLADEL.astype('str')  
X_train, X_test, Y_train, Y_test = train_test_split(XLADEL, YLADEL, test_size=0.33, random_state=9)  
# Fit LADEL model on 33%  
model = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), random_state=2)  
model.fit(X_train, Y_train)  
# save the model to disk  
fbagname = 'MLPjoblib_model_LADEL%d%i.sav' %(n,rank)  
dump(model, LADELmodel+fbagname)  
TendLADELc0 = time.time()  
TmdlLADELc0 = TendLADELc0 - TbeginLADELc0  
reqTmdlLADELc0 = comm.send(TmdlLADELc0, dest=0, tag= 4040)  
  
reqmodM = comm.isend('models', dest=0, tag= 1346)  
reqmodM.wait()
```

## 10. File transferring:

```
elif rank < n: # rank is not 0 and is run on SONY
```

```
#####  
###      FTP Step : The Training datasets are      ###  
###      distributed across the tasks and nodes    ###  
#####  
  
print('Bagging Transformation from processor %i is started.' %rank)  
  
TbeginFTP = time.time()  
from ftplib import FTP  
#domain name or server ip  
ftp = FTP('client')  
ftp.login(user='azi', passwd = 'cmpe1234')  
ftp.cwd('/home/azi/cloudC1/LADEL/data/')  
os.chdir('/home/azi/cloud/LADEL/data/')  
  
def placeFile():  
  
    filename = 'HIGGS.data.corrected.ordered.LADEL.part%i.csv' %(n,rank)  
    ftp.storbinary("STOR "+filename, open(filename, 'rb'))  
    ftp.quit()  
  
placeFile()  
print('File is Transferred')  
TendFTP = time.time()  
TFTPc0 = TendFTP - TbeginFTP  
reqTFTPc0 = comm.send(TFTPc0, dest=0, tag=4044)
```

## 11. Training subsystem in other tasks (two-third of tasks):

```
#####  
##      Training in second laptop      ##  
##      LADEL Algorithm                ##  
#####  
  
TbeginLADELc0 = time.time()  
filename = LADELFileC + 'HIGGS.data.corrected.ordered.LADEL.part%i.csv' %(n,rank)  
  
dataframeTLADEL = read_csv(filename, names=names, low_memory = False)  
arrayTLADEL = dataframeTLADEL.values  
XTLADEL = arrayTLADEL[:,7:43]  
YTLADEL = arrayTLADEL[:,44]  
YTLADEL = YTLADEL.astype('str')  
X_train, X_test, Y_train, Y_test = train_test_split(XTLADEL, YTLADEL, test_size=0.01, random_state=9)  
# Fit TLADEL model on 1%  
modelTLADEL = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), random_state=2)  
modelTLADEL.fit(X_train, Y_train)  
# save TLADEL model to disk  
filenameTLADEL = 'MLPjoblib_model_LADEL%d%i.sav' %(n,rank)  
dump(modelTLADEL, LADELmodel+filenameTLADEL)  
TendLADELc0 = time.time()  
TmdlLADELc0 = TendLADELc0 - TbeginLADELc0  
reqTmdlLADELc0 = comm.send(TmdlLADELc0, dest=0, tag= 4040)  
  
reqmodM = comm.isend('models', dest=0, tag= 1346)  
reqmodM.wait()
```

459  
460  
461  
462

## 5) DELS system for LADEL model by CART (Python source code):

All sections are the same with the MLP classification. The differences are coded in the following lines.

### 1. Definitions and inputs:

```
from sklearn.tree import DecisionTreeClassifier
```

### 9 .Training subsystem in master and other tasks (LADEL):

```
fbagname = LADELFile + 'HIGGS.data.corrected.ordered.LADEL.part%i%i.csv' %(n,rank)

dataframeMLPLADEL = read_csv(fbagname, names=names, low_memory = False)
arrayMLPLADEL = dataframeMLPLADEL.values
XMLPLADEL = arrayMLPLADEL[:,7:43]
YMLPLADEL = arrayMLPLADEL[:,44]
YMLPLADEL = YMLPLADEL.astype('str')
seed = 7
num_trees = 100
max_features = 3
kfold = KFold(n_splits=10, random_state=seed)
# Bagged Decision Trees Model
X_train, X_test, Y_train, Y_test = train_test_split(XMLPLADEL, YMLPLADEL, test_size= 0.01, random_state =
9)
cart = DecisionTreeClassifier()
modelMLPLADEL = BaggingClassifier(base_estimator=cart, n_estimators=num_trees, random_state=seed)
modelMLPLADEL.fit(X_train, Y_train)
# save the model MLPLADEL to disk
fbagname = 'CARTjoblib_model_LADEL%d%i.sav' %(n,rank)
dump(modelMLPLADEL, LADELmodel+fbagname)
```

## Appendix B: Experimental results of DELS

Table B.1: Implementation of the subsystem of data partitioning by DSS. single-node, single-classifier (CART) for 2 disjoint partitions:

NODE #	Part #	Partition	Classifier	Accuracy	STD	SEM	Runtime(Sec)
2	0	D	Decision Tree-CART	0.999427	0.000584419	0.00018481	360
2	0	DB	Decision Tree-CART	0.999182	0.000484216	0.000153123	360
2	0	SB	Decision Tree-CART	0.999304	0.000550564	0.000174104	360
2	0	NRSB	Decision Tree-CART	0.999263	0.000703935	0.000222604	360
2	1	D	Decision Tree-CART	0.999304	0.000777647	0.000245914	360
2	1	DB	Decision Tree-CART	0.999468	0.00045016	0.000142353	360
2	1	SB	Decision Tree-CART	0.999509	0.000477211	0.000150907	360

**ORDER:** order of input dataset

**NODE:** number of processor on one or more systems

**PART#:** processor rank number

**PARTITION:** method of partitioning input dataset

**CLASSIFIER:** type of usage classifier

**ACCURACY:** Mean of cross validation score on 10 fold

**STD:** Standard Deviation

**SEM:** Standard Error of Mean

**CP (MASTER:CLIENT):** no of processors used in master and client machines controlled by MPI command

$$\text{Standard deviation } \varphi = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}} \quad \text{Variance} = \varphi^2$$

$$\text{Standard Error of Mean (SEM)} (\varphi_{\bar{x}}) = \frac{\varphi}{\sqrt{n}}$$

Where  $\bar{x}$  = the sample's mean & n = the sample size

Table B.2: Implementation of the subsystem of data partitioning by DSS. single-node, single-classifier (CART) for 4 disjoint partitions:

Ensemble and Classifier.

NODE #	Part #	Partition	Classifier	Accuracy	Standard Dev	Standard Error	Runtime(Sec)
4	0	D	Decision Tree-CART	0.998445	0.0008543	0.0002702	300
4	0	DB	Decision Tree-CART	0.999018	0.0008019	0.0002536	300
4	0	SB	Decision Tree-CART	0.9991	0.0008544	0.0002702	300
4	0	NRSB	Decision Tree-CART	0.998526	0.001409	0.0004456	300
4	1	D	Decision Tree-CART	0.998936	0.0007367	0.000233	300
4	1	DB	Decision Tree-CART	0.999182	0.0007319	0.0002315	300
4	1	SB	Decision Tree-CART	0.999182	0.0008964	0.0002835	300
4	1	NRSB	Decision Tree-CART	0.998854	0.001048	0.0003314	300
4	2	D	Decision Tree-CART	0.998527	0.0014553	0.0004602	300
4	2	DB	Decision Tree-CART	0.999182	0.0009683	0.0003062	300
4	2	SB	Decision Tree-CART	0.999345	0.0006124	0.0001937	300
4	2	NRSB	Decision Tree-CART	0.99869	0.001048	0.0003314	300
4	3	D	Decision Tree-CART	0.998854	0.0012247	0.0003873	300
4	3	DB	Decision Tree-CART	0.9991	0.0006797	0.000215	300
4	3	SB	Decision Tree-CART	0.998854	0.001473	0.0004658	300
4	3	NRSB	Decision Tree-CART	0.9991	0.0006797	0.000215	300

Table B.3: Implementation of the subsystem of data partitioning by DSS. single-node, single-classifier (CART) for 6 disjoint partitions:  
e Ensemble and Classifier.

NODE #	Part #	Partition	Classifier	Accuracy	Standard Deviat	Standard Error of IV	Runtime(Se
6	0	D	Decision Tree-CART	0.998771951	0.001648207	0.000521209	240
6	0	DB	Decision Tree-CART	0.998035905	0.001472946	0.000465786	240
6	0	SB	Decision Tree-CART	0.998772855	0.001344105	0.000425043	240
6	0	NRSB	Decision Tree-CART	0.999140803	0.000785801	0.000248492	240
6	1	D	Decision Tree-CART	0.998649553	0.001281525	0.000405254	240
6	1	DB	Decision Tree-CART	0.998772252	0.000776496	0.00024555	240
6	1	SB	Decision Tree-CART	0.999017802	0.0007369	0.000233028	240
6	1	NRSB	Decision Tree-CART	0.998403853	0.001105618	0.000349627	240
6	2	D	Decision Tree-CART	0.998772252	0.001345206	0.000425391	240
6	2	DB	Decision Tree-CART	0.998036056	0.001664265	0.000526287	240
6	2	SB	Decision Tree-CART	0.998649704	0.0011578	0.000366128	240
6	2	NRSB	Decision Tree-CART	0.998526552	0.00107081	0.00033862	240
6	3	D	Decision Tree-CART	0.998772102	0.001098468	0.000347366	240
6	3	DB	Decision Tree-CART	0.998895103	0.000859413	0.00027177	240
6	3	SB	Decision Tree-CART	0.998649553	0.000859219	0.000271709	240
6	3	NRSB	Decision Tree-CART	0.998035905	0.001664865	0.000526477	240
6	4	D	Decision Tree-CART	0.998035905	0.000982048	0.000310551	240
6	4	DB	Decision Tree-CART	0.998895253	0.001157592	0.000366063	240
6	4	SB	Decision Tree-CART	0.998158756	0.001669946	0.000528083	240
6	4	NRSB	Decision Tree-CART	0.998158756	0.000951204	0.000300797	240
6	5	D	Decision Tree-CART	0.998158756	0.000859628	0.000271838	240
6	5	DB	Decision Tree-CART	0.998158756	0.001203496	0.000380579	240
6	5	SB	Decision Tree-CART	0.998158756	0.001019654	0.000322443	240
6	5	NRSB	Decision Tree-CART	0.998158756	0.001234299	0.00039032	240

Table B.4: Implementation of the subsystem of data partitioning by DSS. single-node, single-classifier (CART) for 8 disjoint partitions:

Single-Node Ensemble and Classifier.

ORDER	NODE #	Part #	Partitioning	Classifier	Accuracy	Standard D	Standard E	Runtime(Se
RANDOM	8	0	D	Decision Tre	0.99869	0.001764	0.000558	180
RANDOM	8	0	DB	Decision Tre	0.9982	0.00136	0.00043	180
RANDOM	8	0	SB	Decision Tre	0.998363	0.001637	0.000518	180
RANDOM	8	0	NRSB	Decision Tre	0.999018	0.0015	0.000474	180
RANDOM	8	1	D	Decision Tre	0.998691	0.001225	0.000387	180
RANDOM	8	1	DB	Decision Tre	0.997709	0.002096	0.000663	180
RANDOM	8	1	SB	Decision Tre	0.99869	0.000982	0.000311	180
RANDOM	8	1	NRSB	Decision Tre	0.998363	0.001937	0.000612	180
RANDOM	8	2	D	Decision Tre	0.998199	0.001859	0.000588	180
RANDOM	8	2	DB	Decision Tre	0.998691	0.001604	0.000507	180
RANDOM	8	2	SB	Decision Tre	0.998199	0.001859	0.000588	180
RANDOM	8	2	NRSB	Decision Tre	0.999182	0.001098	0.000347	180
RANDOM	8	3	D	Decision Tre	0.998854	0.001278	0.000404	180
RANDOM	8	3	DB	Decision Tre	0.998691	0.000982	0.000311	180
RANDOM	8	3	SB	Decision Tre	0.99869	0.001429	0.000452	180
RANDOM	8	3	NRSB	Decision Tre	0.99869	0.001604	0.000507	180
RANDOM	8	4	D	Decision Tre	0.998199	0.001146	0.000363	180
RANDOM	8	4	DB	Decision Tre	0.998691	0.001225	0.000387	180
RANDOM	8	4	SB	Decision Tre	0.998854	0.001278	0.000404	180
RANDOM	8	4	NRSB	Decision Tre	0.997872	0.002077	0.000657	180
RANDOM	8	5	D	Decision Tre	0.997217	0.001944	0.000615	180
RANDOM	8	5	DB	Decision Tre	0.999018	0.001669	0.000528	180
RANDOM	8	5	SB	Decision Tre	0.997544	0.002227	0.000704	180
RANDOM	8	5	NRSB	Decision Tre	0.998198	0.00237	0.000749	180
RANDOM	8	6	D	Decision Tre	0.99869	0.001226	0.000388	180
RANDOM	8	6	DB	Decision Tre	0.998527	0.00136	0.00043	180
RANDOM	8	6	SB	Decision Tre	0.998363	0.001268	0.000401	180
RANDOM	8	6	NRSB	Decision Tre	0.998199	0.001546	0.000489	180
RANDOM	8	7	D	Decision Tre	0.998526	0.001147	0.000363	180
RANDOM	8	7	DB	Decision Tre	0.998199	0.001862	0.000589	180
RANDOM	8	7	SB	Decision Tre	0.999018	0.001086	0.000343	180
RANDOM	8	7	NRSB	Decision Tre	0.998854	0.001048	0.000331	180

Table B.5: Implementation of the subsystem of data partitioning by DSS. single-node, single-classifier (CART) for 10 disjoint partitions:  
Single-Node Ensemble and Classifier.

ORDER	NODE	Part #	Partitic	Classifier	Accuracy	Standard Deviatc	Standard Erro	Runtime(S
RANDOM	10	0	D	Decision T	0.998976667	0.001373696	0.0004344	120
RANDOM	10	0	DB	Decision T	0.99856725	0.001843705	0.00058303	120
RANDOM	10	0	SB	Decision T	0.998363589	0.001530443	0.00048397	120
RANDOM	10	0	NRSB	Decision T	0.999182004	0.001356493	0.00042896	120
RANDOM	10	1	D	Decision T	0.997747997	0.002138004	0.0006761	120
RANDOM	10	1	DB	Decision T	0.99856725	0.001311332	0.00041468	120
RANDOM	10	1	SB	Decision T	0.99815909	0.001698747	0.00053719	120
RANDOM	10	1	NRSB	Decision T	0.997749254	0.001102586	0.00034867	120
RANDOM	10	2	D	Decision T	0.998976667	0.001372446	0.00043401	120
RANDOM	10	2	DB	Decision T	0.997341094	0.002594702	0.00082052	120
RANDOM	10	2	SB	Decision T	0.997544755	0.002004273	0.00063381	120
RANDOM	10	2	NRSB	Decision T	0.997750092	0.001101184	0.00034822	120
RANDOM	10	3	D	Decision T	0.998158671	0.002956466	0.00093492	120
RANDOM	10	3	DB	Decision T	0.997341094	0.002249375	0.00071131	120
RANDOM	10	3	SB	Decision T	0.998159509	0.001698696	0.00053717	120
RANDOM	10	3	NRSB	Decision T	0.999181166	0.001358515	0.0004296	120
RANDOM	10	4	D	Decision T	0.997953334	0.001585126	0.00050126	120
RANDOM	10	4	DB	Decision T	0.998977086	0.001372134	0.00043391	120
RANDOM	10	4	SB	Decision T	0.99836317	0.0020047	0.00063394	120
RANDOM	10	4	NRSB	Decision T	0.998773006	0.002085489	0.00065949	120
RANDOM	10	5	D	Decision T	0.998157833	0.002137603	0.00067597	120
RANDOM	10	5	DB	Decision T	0.998772587	0.001356746	0.00042904	120
RANDOM	10	5	SB	Decision T	0.998568088	0.001309629	0.00041414	120
RANDOM	10	5	NRSB	Decision T	0.997954591	0.002419665	0.00076517	120
RANDOM	10	6	D	Decision T	0.998157833	0.001432932	0.00045313	120
RANDOM	10	6	DB	Decision T	0.99795501	0.002419665	0.00076517	120
RANDOM	10	6	SB	Decision T	0.998772587	0.001356746	0.00042904	120
RANDOM	10	6	NRSB	Decision T	0.997340675	0.00275169	0.00087016	120
RANDOM	10	7	D	Decision T	0.998158671	0.001929329	0.00061011	120
RANDOM	10	7	DB	Decision T	0.997750511	0.002135032	0.00067516	120
RANDOM	10	7	SB	Decision T	0.997750092	0.002322625	0.00073448	120
RANDOM	10	7	NRSB	Decision T	0.998362751	0.002006497	0.00063451	120
RANDOM	10	8	D	Decision T	0.998156995	0.001702027	0.00053823	120
RANDOM	10	8	DB	Decision T	0.997748416	0.001932573	0.00061113	120
RANDOM	10	8	SB	Decision T	0.998158252	0.001930263	0.0006104	120
RANDOM	10	8	NRSB	Decision T	0.997953753	0.002421792	0.00076584	120
RANDOM	10	9	D	Decision T	0.998772168	0.00163746	0.00051781	120
RANDOM	10	9	DB	Decision T	0.998363589	0.001530443	0.00048397	120
RANDOM	10	9	SB	Decision T	0.998158671	0.001431614	0.00045272	120
RANDOM	10	9	NRSB	Decision T	0.998159509	0.001698696	0.00053717	120

Table B.6: Implementation of the subsystem of data partitioning by DSS. single-node, single-classifier (CART) for 12 disjoint partitions:

ORDER	NODE #	Part #	Partition	Classifier	Accuracy	Standard Deviat	Standard Error of	Runtime(Se
RANDOM	12	0	D	CART	0.9970546081	0.0018362429	0.0005806710	120
RANDOM	12	0	DB	CART	0.9982813027	0.0015726866	0.0004973272	120
RANDOM	12	0	SB	CART	0.9985264007	0.0016294289	0.0005152707	120
RANDOM	12	0	NRSB	CART	0.9977893000	0.0027905659	0.0008824544	120
RANDOM	12	1	D	CART	0.9987714988	0.0016482074	0.0005212089	120
RANDOM	12	1	DB	CART	0.9992635015	0.0011250212	0.0003557629	120
RANDOM	12	1	SB	CART	0.9982819049	0.0019167468	0.0006061286	120
RANDOM	12	1	NRSB	CART	0.9985257985	0.0036773046	0.0011628658	120
RANDOM	12	2	D	CART	0.9972997061	0.0023143606	0.0007318651	120
RANDOM	12	2	DB	CART	0.9973009105	0.0031878991	0.0010081022	120
RANDOM	12	2	SB	CART	0.9980350002	0.0021418277	0.0006773054	120
RANDOM	12	2	NRSB	CART	0.9975448042	0.0019016343	0.0006013496	120
RANDOM	12	3	D	CART	0.9968077034	0.0019196761	0.0006070549	120
RANDOM	12	3	DB	CART	0.9982807005	0.0024690778	0.0007807910	120
RANDOM	12	3	SB	CART	0.9985264007	0.0019653017	0.0006214830	120
RANDOM	12	3	NRSB	CART	0.9977899022	0.0017197313	0.0005438268	120
RANDOM	12	4	D	CART	0.9973003083	0.0033745387	0.0010671228	120
RANDOM	12	4	DB	CART	0.9977929132	0.0029919025	0.0009461226	120
RANDOM	12	4	SB	CART	0.9970540059	0.0026456046	0.0008366136	120
RANDOM	12	4	NRSB	CART	0.9985264007	0.0016294289	0.0005152707	120
RANDOM	12	5	D	CART	0.9985270029	0.0019634964	0.0006209121	120
RANDOM	12	5	DB	CART	0.9990171990	0.0016297911	0.0005153852	120
RANDOM	12	5	SB	CART	0.9980343980	0.0024073609	0.0007612744	120
RANDOM	12	5	NRSB	CART	0.9995092017	0.0009815975	0.0003104084	120
RANDOM	12	6	D	CART	0.9972991039	0.0027897195	0.0008821868	120
RANDOM	12	6	DB	CART	0.9980374091	0.0028586209	0.0009039753	120
RANDOM	12	6	SB	CART	0.9975429975	0.0021976098	0.0006949452	120
RANDOM	12	6	NRSB	CART	0.9987727032	0.0016473112	0.0005209255	120
RANDOM	12	7	D	CART	0.9985264007	0.0019653017	0.0006214830	120
RANDOM	12	7	DB	CART	0.9968089078	0.0022082961	0.0006983245	120
RANDOM	12	7	SB	CART	0.9992635015	0.0011250212	0.0003557629	120
RANDOM	12	7	NRSB	CART	0.9977899022	0.0023178051	0.0007329543	120
RANDOM	12	8	D	CART	0.9982800983	0.0022113022	0.0006992752	120
RANDOM	12	8	DB	CART	0.9987721010	0.0016477594	0.0005210673	120
RANDOM	12	8	SB	CART	0.9985257985	0.0019656020	0.0006215779	120
RANDOM	12	8	NRSB	CART	0.9965607988	0.0029486042	0.0009324305	120
RANDOM	12	9	D	CART	0.9982813027	0.0022109020	0.0006991486	120
RANDOM	12	9	DB	CART	0.9982825071	0.0024649449	0.0007794840	120
RANDOM	12	9	SB	CART	0.9987721010	0.0016477594	0.0005210673	120
RANDOM	12	9	NRSB	CART	0.9977899022	0.0027905133	0.0008824378	120
RANDOM	12	10	D	CART	0.9985276051	0.0016268879	0.0005144671	120
RANDOM	12	10	DB	CART	0.9973003083	0.0025631708	0.0008105458	120
RANDOM	12	10	SB	CART	0.9987721010	0.0016477594	0.0005210673	120
RANDOM	12	10	NRSB	CART	0.9985276051	0.0016268879	0.0005144671	120
RANDOM	12	11	D	CART	0.9972985017	0.0023180604	0.0007330351	120
RANDOM	12	11	DB	CART	0.9985282074	0.0019613879	0.0006202453	120
RANDOM	12	11	SB	CART	0.9980374091	0.0021371314	0.0006758203	120
RANDOM	12	11	NRSB	CART	0.9987733054	0.0025132726	0.0007947666	120



## Appendix C: Experimental results of DELS system

Table C.1: Implementation of the subsystem of data partitioning by DSS, single-node, single-classifier (MLP) for 2 disjoint partitions:

Single-Node Ensemble and Classifier.								
ORDER	NODE #	Part #	Partitioning Me	Classifier	Accuracy	Standard Deviation	Standard Error of	Runtime(Sr
RANDOM	2	0	D	MLP	0.9736045	0.004949548	0.001565185	4080
RANDOM	2	0	DB	MLP	0.97205079	0.003352801	0.001060249	4080
RANDOM	2	0	SB	MLP	0.97176267	0.00413519	0.001307662	4080
RANDOM	2	0	NRSB	MLP	0.97638706	0.005039876	0.001593749	4080
RANDOM	2	1	D	MLP	0.97155725	0.003165296	0.001000955	4080
RANDOM	2	1	DB	MLP	0.97761536	0.005537967	0.001751259	4080
RANDOM	2	1	SB	MLP	0.97196759	0.002641358	0.000835271	4080
RANDOM	2	1	NRSB	MLP	0.97753279	0.005803803	0.001835324	4080
	MEAN Accuracy	Mean SEM	MA - SEM	MA + SEM				
D	0.972580875	0.00128307	0.971297805	0.97386394				
DB	0.974833076	0.001405754	0.973427322	0.97623883				
SB	0.971865129	0.001071466	0.970793663	0.9729366				
NRSB	0.976959924	0.001714536	0.975245387	0.97867446				

**ORDER:** order of input dataset

**NODE:** number of processor on one or more systems

**PART#:** processor rank number

**PARTITION:** method of partitioning input dataset

**CLASSIFIER:** type of usage classifier

**ACCURACY:** Mean of cross validation score on 10 fold

**STD:** Standard Deviation

**SEM:** Standard Error of Mean

**CP (MASTER:CLIENT):** no of processors used in master and client machines controlled by MPI command

$$\text{Standard deviation } \varphi = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}} \quad \text{Variance} = \varphi^2$$

$$\text{Standard Error of Mean (SEM)} (\varphi_{\bar{x}}) = \frac{\varphi}{\sqrt{n}}$$

Where  $\bar{x}$  = the sample's mean & n = the sample size

Table C.2: Implementation of the subsystem of data partitioning by DSS, single-node, single-classifier (MLP) for 4 disjoint partitions:

Single-Node Ensemble and Classifier.

ORDER	NODE #	Part #	Partitior	Classifier	Accuracy	Standard Deviat	Standard Error o	Runtime(Sr
RANDOM	4	0	D	MLP	0.969471041	0.003377356	0.001068014	2520
RANDOM	4	0	DB	MLP	0.967346531	0.007678578	0.00242818	2520
RANDOM	4	0	SB	MLP	0.971517873	0.004101962	0.001297154	2520
RANDOM	4	0	NRSB	MLP	0.971926435	0.00310987	0.000983427	2520
RANDOM	4	1	D	MLP	0.969717545	0.004434248	0.001402232	2520
RANDOM	4	1	DB	MLP	0.969064824	0.003327991	0.001052403	2520
RANDOM	4	1	SB	MLP	0.970780437	0.0051135	0.001617031	2520
RANDOM	4	1	NRSB	MLP	0.969471845	0.005912589	0.001869725	2520
RANDOM	4	2	D	MLP	0.967997644	0.005998723	0.001896963	2520
RANDOM	4	2	DB	MLP	0.972829145	0.003869979	0.001223795	2520
RANDOM	4	2	SB	MLP	0.970699341	0.00388437	0.001228346	2520
RANDOM	4	2	NRSB	MLP	0.965869716	0.004500544	0.001423197	2520
RANDOM	4	3	D	MLP	0.967668301	0.003910078	0.001236475	2520
RANDOM	4	3	DB	MLP	0.970290042	0.004714657	0.001490905	2520
RANDOM	4	3	SB	MLP	0.967916548	0.003054454	0.000965903	2520
RANDOM	4	3	NRSB	MLP	0.968815438	0.005150237	0.001628648	2520

Table C.3: Implementation of the subsystem of data partitioning by DSS, single-node, single-classifier (MLP) for 6 disjoint partitions:  
Single-Node Ensemble and Classifier.

ORDER	NODE #	Part #	Partitioning	Classifier	Accuracy	Standard D	Standard E	Runtime(S)
RANDOM	6	0	D	MLP	0.966117	0.006245	0.001975	1440
RANDOM	6	0	DB	MLP	0.968947	0.004385	0.001387	1440
RANDOM	6	0	SB	MLP	0.965137	0.00485	0.001534	1440
RANDOM	6	0	NRSB	MLP	0.970904	0.006738	0.002131	1440
RANDOM	6	1	D	MLP	0.970412	0.004953	0.001566	1440
RANDOM	6	1	DB	MLP	0.966363	0.004985	0.001576	1440
RANDOM	6	1	SB	MLP	0.966609	0.006543	0.002069	1440
RANDOM	6	1	NRSB	MLP	0.968696	0.005631	0.001781	1440
RANDOM	6	2	D	MLP	0.969429	0.005361	0.001695	1440
RANDOM	6	2	DB	MLP	0.967962	0.007426	0.002348	1440
RANDOM	6	2	SB	MLP	0.968329	0.007082	0.00224	1440
RANDOM	6	2	NRSB	MLP	0.967714	0.006016	0.001902	1440
RANDOM	6	3	D	MLP	0.969185	0.006287	0.001988	1440
RANDOM	6	3	DB	MLP	0.967223	0.0069	0.002182	1440
RANDOM	6	3	SB	MLP	0.970292	0.004352	0.001376	1440
RANDOM	6	3	NRSB	MLP	0.966363	0.006175	0.001953	1440
RANDOM	6	4	D	MLP	0.967712	0.00583	0.001844	1440
RANDOM	6	4	DB	MLP	0.971154	0.005027	0.00159	1440
RANDOM	6	4	SB	MLP	0.967101	0.006714	0.002123	1440
RANDOM	6	4	NRSB	MLP	0.96698	0.006449	0.002039	1440
RANDOM	6	5	D	MLP	0.967955	0.006152	0.001946	1440
RANDOM	6	5	DB	MLP	0.967224	0.005029	0.00159	1440
RANDOM	6	5	SB	MLP	0.963909	0.007804	0.002468	1440
RANDOM	6	5	NRSB	MLP	0.968573	0.005097	0.001612	1440

Table C.4: Implementation of the subsystem of data partitioning by DSS, single-node, single-classifier (MLP) for 8 disjoint partitions:  
Single-Node Ensemble and Classifier.

ORDER	NODE #	Part #	Partitionin;	Classifier	Accuracy	Standard C	Standard E	Runtime(S)
RANDOM	8	0	D	MLP	0.965625	0.005069	0.001603	1140
RANDOM	8	0	DB	MLP	0.964648	0.007179	0.00227	1140
RANDOM	8	0	SB	MLP	0.962023	0.004622	0.001462	1140
RANDOM	8	0	NRSB	MLP	0.969228	0.009278	0.002934	1140
RANDOM	8	1	D	MLP	0.964315	0.004256	0.001346	1140
RANDOM	8	1	DB	MLP	0.963993	0.007607	0.002405	1140
RANDOM	8	1	SB	MLP	0.965134	0.007019	0.00222	1140
RANDOM	8	1	NRSB	MLP	0.962839	0.005692	0.0018	1140
RANDOM	8	2	D	MLP	0.962841	0.006128	0.001938	1140
RANDOM	8	2	DB	MLP	0.963339	0.010226	0.003234	1140
RANDOM	8	2	SB	MLP	0.965788	0.007154	0.002262	1140
RANDOM	8	2	NRSB	MLP	0.966116	0.005279	0.00167	1140
RANDOM	8	3	D	MLP	0.968572	0.00556	0.001758	1140
RANDOM	8	3	DB	MLP	0.96743	0.008198	0.002592	1140
RANDOM	8	3	SB	MLP	0.96546	0.007628	0.002412	1140
RANDOM	8	3	NRSB	MLP	0.96726	0.005684	0.001798	1140
RANDOM	8	4	D	MLP	0.964475	0.011642	0.003681	1140
RANDOM	8	4	DB	MLP	0.965794	0.006187	0.001956	1140
RANDOM	8	4	SB	MLP	0.967588	0.00764	0.002416	1140
RANDOM	8	4	NRSB	MLP	0.963496	0.006129	0.001938	1140
RANDOM	8	5	D	MLP	0.96677	0.004808	0.00152	1140
RANDOM	8	5	DB	MLP	0.964975	0.007179	0.00227	1140
RANDOM	8	5	SB	MLP	0.964152	0.01114	0.003778	1140
RANDOM	8	5	NRSB	MLP	0.964152	0.00726	0.002296	1140
RANDOM	8	6	D	MLP	0.966935	0.011284	0.003568	1140
RANDOM	8	6	DB	MLP	0.969231	0.00767	0.002425	1140
RANDOM	8	6	SB	MLP	0.963497	0.009821	0.003106	1140
RANDOM	8	6	NRSB	MLP	0.962023	0.00393	0.001243	1140
RANDOM	8	7	D	MLP	0.967913	0.006125	0.001937	1140
RANDOM	8	7	DB	MLP	0.969881	0.006872	0.002173	1140
RANDOM	8	7	SB	MLP	0.96628	0.008286	0.00262	1140
RANDOM	8	7	NRSB	MLP	0.967262	0.00828	0.002618	1140

Table C.5: Implementation of the subsystem of data partitioning by DSS, single-node, single-classifier (MLP) for 10 disjoint partitions:

Single-Node Ensemble and Classifier.

ORDER	NODE #	Part #	Partitioning	Classifier	Accuracy	Standard D	Standard E	Runtime(Sec)
RANDOM	10	0	D	MLP	0.965223	0.007365	0.002329	840
RANDOM	10	0	DB	MLP	0.961545	0.008429	0.002665	840
RANDOM	10	0	SB	MLP	0.963996	0.008869	0.002805	840
RANDOM	10	0	NRSB	MLP	0.967066	0.008445	0.00267	840
RANDOM	10	1	D	MLP	0.964193	0.007874	0.00249	840
RANDOM	10	1	DB	MLP	0.96256	0.007608	0.002406	840
RANDOM	10	1	SB	MLP	0.966447	0.008644	0.002733	840
RANDOM	10	1	NRSB	MLP	0.966244	0.006224	0.001968	840
RANDOM	10	2	D	MLP	0.965218	0.008116	0.002567	840
RANDOM	10	2	DB	MLP	0.962153	0.004105	0.001298	840
RANDOM	10	2	SB	MLP	0.96338	0.008603	0.002721	840
RANDOM	10	2	NRSB	MLP	0.962973	0.007275	0.002301	840
RANDOM	10	3	D	MLP	0.962347	0.008697	0.00275	840
RANDOM	10	3	DB	MLP	0.962967	0.010579	0.003345	840
RANDOM	10	3	SB	MLP	0.949469	0.006333	0.002003	840
RANDOM	10	3	NRSB	MLP	0.962356	0.010876	0.003439	840
RANDOM	10	4	D	MLP	0.964801	0.007831	0.002476	840
RANDOM	10	4	DB	MLP	0.963991	0.006684	0.002114	840
RANDOM	10	4	SB	MLP	0.959288	0.006303	0.001993	840
RANDOM	10	4	NRSB	MLP	0.96461	0.010818	0.003421	840
RANDOM	10	5	D	MLP	0.963778	0.007452	0.002356	840
RANDOM	10	5	DB	MLP	0.963788	0.008492	0.002685	840
RANDOM	10	5	SB	MLP	0.964404	0.006006	0.001899	840
RANDOM	10	5	NRSB	MLP	0.963996	0.006469	0.002046	840
RANDOM	10	6	D	MLP	0.959281	0.008449	0.002672	840
RANDOM	10	6	DB	MLP	0.963585	0.008072	0.002552	840
RANDOM	10	6	SB	MLP	0.954991	0.005262	0.001664	840
RANDOM	10	6	NRSB	MLP	0.964814	0.009576	0.003028	840
RANDOM	10	7	D	MLP	0.964804	0.005773	0.001825	840
RANDOM	10	7	DB	MLP	0.963994	0.006353	0.002009	840
RANDOM	10	7	SB	MLP	0.960106	0.005113	0.001617	840
RANDOM	10	7	NRSB	MLP	0.963994	0.00733	0.002318	840
RANDOM	10	8	D	MLP	0.963985	0.006163	0.001949	840
RANDOM	10	8	DB	MLP	0.965426	0.007228	0.002286	840
RANDOM	10	8	SB	MLP	0.964605	0.007329	0.002318	840
RANDOM	10	8	NRSB	MLP	0.959904	0.010467	0.00331	840
RANDOM	10	9	D	MLP	0.965626	0.006698	0.002118	840
RANDOM	10	9	DB	MLP	0.966654	0.010469	0.003311	840
RANDOM	10	9	SB	MLP	0.963994	0.008038	0.002542	840
RANDOM	10	9	NRSB	MLP	0.964608	0.01015	0.00321	840

Table C.6: Implementation of the subsystem of data partitioning by DSS, single-node, single-classifier (MLP) for 12 disjoint partitions:

RANDOM	12	0 D	MLP	0.958753	0.0087	0.002751	780
RANDOM	12	0 DB	MLP	0.960733	0.010213	0.00323	780
RANDOM	12	0 SB	MLP	0.96317	0.012086	0.003822	780
RANDOM	12	0 NRSB	MLP	0.963662	0.008847	0.002798	780
RANDOM	12	1 D	MLP	0.961944	0.007472	0.002363	780
RANDOM	12	1 DB	MLP	0.958519	0.007635	0.002414	780
RANDOM	12	1 SB	MLP	0.95753	0.013689	0.004329	780
RANDOM	12	1 NRSB	MLP	0.961215	0.008732	0.002761	780
RANDOM	12	2 D	MLP	0.964647	0.010066	0.003183	780
RANDOM	12	2 DB	MLP	0.961711	0.006685	0.002114	780
RANDOM	12	2 SB	MLP	0.961942	0.009346	0.002956	780
RANDOM	12	2 NRSB	MLP	0.961209	0.006831	0.00216	780
RANDOM	12	3 D	MLP	0.966117	0.003941	0.001246	780
RANDOM	12	3 DB	MLP	0.961951	0.004842	0.001531	780
RANDOM	12	3 SB	MLP	0.960959	0.011553	0.003653	780
RANDOM	12	3 NRSB	MLP	0.960965	0.009403	0.002974	780
RANDOM	12	4 D	MLP	0.966363	0.00572	0.001809	780
RANDOM	12	4 DB	MLP	0.968091	0.008769	0.002773	780
RANDOM	12	4 SB	MLP	0.965135	0.006842	0.002164	780
RANDOM	12	4 NRSB	MLP	0.964643	0.010377	0.003281	780
RANDOM	12	5 D	MLP	0.96317	0.011938	0.003775	780
RANDOM	12	5 DB	MLP	0.964897	0.008935	0.002826	780
RANDOM	12	5 SB	MLP	0.96096	0.012637	0.003996	780
RANDOM	12	5 NRSB	MLP	0.966854	0.0053	0.001676	780
RANDOM	12	6 D	MLP	0.956298	0.012408	0.003924	780
RANDOM	12	6 DB	MLP	0.966861	0.009143	0.002891	780
RANDOM	12	6 SB	MLP	0.965622	0.012539	0.003965	780
RANDOM	12	6 NRSB	MLP	0.945737	0.014838	0.004692	780
RANDOM	12	7 D	MLP	0.961697	0.011544	0.003651	780
RANDOM	12	7 DB	MLP	0.95311	0.010596	0.003351	780
RANDOM	12	7 SB	MLP	0.954341	0.011959	0.003782	780
RANDOM	12	7 NRSB	MLP	0.963654	0.014031	0.004437	780
RANDOM	12	8 D	MLP	0.960466	0.01104	0.003491	780
RANDOM	12	8 DB	MLP	0.958757	0.010953	0.003464	780
RANDOM	12	8 SB	MLP	0.965382	0.007872	0.002489	780
RANDOM	12	8 NRSB	MLP	0.960225	0.007435	0.002351	780
RANDOM	12	9 D	MLP	0.963165	0.009631	0.003046	780
RANDOM	12	9 DB	MLP	0.96341	0.012564	0.003973	780
RANDOM	12	9 SB	MLP	0.957523	0.009588	0.003032	780
RANDOM	12	9 NRSB	MLP	0.965132	0.010478	0.003313	780
RANDOM	12	10 D	MLP	0.956777	0.007791	0.002464	780
RANDOM	12	10 DB	MLP	0.961938	0.009425	0.00298	780
RANDOM	12	10 SB	MLP	0.957771	0.007347	0.002323	780
RANDOM	12	10 NRSB	MLP	0.961944	0.007712	0.002439	780
RANDOM	12	11 D	MLP	0.957514	0.009396	0.002971	780

## Appendix D: Experimental results of DELS system

Table D.1: Implementation of the subsystem of data partitioning by DSS, multi-node, single-classifier (CART) for 2 disjoint partitions:

Multi-Node Ensemble and Classifier.

ORDER	NODE #	Part #	Partitioning	Classifier	Accuracy	Standard D	Standard E	CP (Master	Runtime(S
RANDOM	2	0	D	Decision Tree-CART	0.999468	0.000686	0.000217	1:1	540
RANDOM	2	0	DB	Decision Tree-CART	0.999304	0.000661	0.000209	1:1	540
RANDOM	2	0	SB	Decision Tree-CART	0.999304	0.000519	0.000164	1:1	540
RANDOM	2	0	NRSB	Decision Tree-CART	0.999141	0.000532	0.000168	1:1	540
RANDOM	2	1	D	Decision Tree-CART	0.999304	0.00045	0.000142	1:1	540
RANDOM	2	1	DB	Decision Tree-CART	0.999468	0.000368	0.000116	1:1	540
RANDOM	2	1	SB	Decision Tree-CART	0.99955	0.00034	0.000108	1:1	540
RANDOM	2	1	NRSB	Decision Tree-CART	0.999386	0.000526	0.000166	1:1	540

CP (Master: Client) = number of processor of master and client systems in MPI

Table D.2: Implementation of the subsystem of data partitioning by DSS, multi-node, single-classifier (CART) for 4 disjoint partitions:

Multi-Node Ensemble and Classifier.

ORDER	NODE #	Part #	Partitioning	Classifier	Accuracy	Standard Deviation	Standard Error of	CP (Master	Runtime(S
RANDOM	4	0	D	CART	0.998526804	0.001022288	0.000323276	3:1	240
RANDOM	4	0	DB	CART	0.999017936	0.000612365	0.000193647	3:1	240
RANDOM	4	0	SB	CART	0.999345201	0.000612598	0.00019372	3:1	240
RANDOM	4	0	NRSB	CART	0.999263368	0.000772026	0.000244136	3:1	240
RANDOM	4	1	D	CART	0.999099635	0.000572994	0.000181197	3:1	240
RANDOM	4	1	DB	CART	0.998608704	0.000639188	0.000202129	3:1	240
RANDOM	4	1	SB	CART	0.999099635	0.000854728	0.000270289	3:1	240
RANDOM	4	1	NRSB	CART	0.999426833	0.000972476	0.000307524	3:1	240
RANDOM	4	2	D	CART	0.998608771	0.001554803	0.000491672	3:1	240
RANDOM	4	2	DB	CART	0.998690671	0.00116881	0.00036961	3:1	240
RANDOM	4	2	SB	CART	0.999017869	0.000801907	0.000253585	3:1	240
RANDOM	4	2	NRSB	CART	0.998608704	0.00127033	0.000401714	3:1	240
RANDOM	4	3	D	CART	0.999099568	0.000679886	0.000214999	3:1	240
RANDOM	4	3	DB	CART	0.999099702	0.00067974	0.000214953	3:1	240
RANDOM	4	3	SB	CART	0.99877237	0.000985345	0.000311593	3:1	240
RANDOM	4	3	NRSB	CART	0.998608771	0.001270361	0.000401723	3:1	240

CP (Master: Client) = number of processor of master and client systems in MPI are in order 3 and 1.

Table D.3: Implementation of the subsystem of data partitioning by DSS, multi-node, single-classifier (CART) for 6 disjoint partitions:

Multi-Node Ensemble and Classifier.									
ORDER	NODE	Part #	Partition	Classifier	Accuracy	Standard Deviat	Standard Er	CP (Maste	Runtime(S
RANDOM	6	0	D	Decision Tree-CART	0.999386352	0.000823205	0.0002603	4:2	180
RANDOM	6	0	DB	Decision Tree-CART	0.998404305	0.001349803	0.0004268	4:2	180
RANDOM	6	0	SB	Decision Tree-CART	0.9990175	0.001070914	0.0003387	4:2	180
RANDOM	6	0	NRSB	Decision Tree-CART	0.999017953	0.000918722	0.0002905	4:2	180
RANDOM	6	1	D	Decision Tree-CART	0.999017651	0.001203281	0.0003805	4:2	180
RANDOM	6	1	DB	Decision Tree-CART	0.998772403	0.001097794	0.0003472	4:2	180
RANDOM	6	1	SB	Decision Tree-CART	0.999017651	0.000919206	0.0002907	4:2	180
RANDOM	6	1	NRSB	Decision Tree-CART	0.998035905	0.001251353	0.0003957	4:2	180
RANDOM	6	2	D	Decision Tree-CART	0.998771951	0.001821959	0.0005762	4:2	180
RANDOM	6	2	DB	Decision Tree-CART	0.997790205	0.001533192	0.0004848	4:2	180
RANDOM	6	2	SB	Decision Tree-CART	0.999140652	0.000786342	0.0002487	4:2	180
RANDOM	6	2	NRSB	Decision Tree-CART	0.999263653	0.000981671	0.0003104	4:2	180
RANDOM	6	3	D	Decision Tree-CART	0.998649252	0.000859607	0.0002718	4:2	180
RANDOM	6	3	DB	Decision Tree-CART	0.999263201	0.000814714	0.0002576	4:2	180
RANDOM	6	3	SB	Decision Tree-CART	0.998158454	0.00125884	0.0003981	4:2	180
RANDOM	6	3	NRSB	Decision Tree-CART	0.998404154	0.001350036	0.0004269	4:2	180
RANDOM	6	4	D	Decision Tree-CART	0.998526401	0.001322969	0.0004184	4:2	180
RANDOM	6	4	DB	Decision Tree-CART	0.998404305	0.00165129	0.0005222	4:2	180
RANDOM	6	4	SB	Decision Tree-CART	0.999140501	0.000958931	0.0003032	4:2	180
RANDOM	6	4	NRSB	Decision Tree-CART	0.998404607	0.001456898	0.0004607	4:2	180
RANDOM	6	5	D	Decision Tree-CART	0.998281153	0.001251767	0.0003958	4:2	180
RANDOM	6	5	DB	Decision Tree-CART	0.999263502	0.000814532	0.0002576	4:2	180
RANDOM	6	5	SB	Decision Tree-CART	0.998527004	0.001202389	0.0003802	4:2	180
RANDOM	6	5	NRSB	Decision Tree-CART	0.998526703	0.001202635	0.0003803	4:2	180

Table D.4: Implementation of the subsystem of data partitioning by DSS, multi-node, single-classifier (CART) for 8 disjoint partitions:

Multi-Node Ensemble and Classifier.

ORDER	NODE #	Part #	Partitic	Classifier	Accuracy	Standard Deviation	Standard Error of MCP (Master)	Runtime(Sec)
RANDOM	8	0	D	CART	0.99787234	0.001943428	0.000614566	7:1 180
RANDOM	8	0	DB	CART	0.998690671	0.001426808	0.000451196	7:1 180
RANDOM	8	0	SB	CART	0.998690134	0.001225626	0.000387577	7:1 180
RANDOM	8	0	NRSB	CART	0.998690671	0.001426808	0.000451196	7:1 180
RANDOM	8	1	D	CART	0.998526737	0.000881419	0.000278729	7:1 180
RANDOM	8	1	DB	CART	0.997217676	0.00207669	0.000656707	7:1 180
RANDOM	8	1	SB	CART	0.9985262	0.001545816	0.00048883	7:1 180
RANDOM	8	1	NRSB	CART	0.998199404	0.001997778	0.000631753	7:1 180
RANDOM	8	2	D	CART	0.998199673	0.002476718	0.000783207	7:1 180
RANDOM	8	2	DB	CART	0.998036007	0.001762738	0.000557427	7:1 180
RANDOM	8	2	SB	CART	0.99836307	0.001636661	0.000517558	7:1 180
RANDOM	8	2	NRSB	CART	0.998690134	0.001427548	0.00045143	7:1 180
RANDOM	8	3	D	CART	0.998690671	0.001908659	0.000603571	7:1 180
RANDOM	8	3	DB	CART	0.998854337	0.000750012	0.000237175	7:1 180
RANDOM	8	3	SB	CART	0.999345336	0.000801797	0.00025355	7:1 180
RANDOM	8	3	NRSB	CART	0.99803547	0.002044534	0.000646538	7:1 180
RANDOM	8	4	D	CART	0.999345336	0.001500025	0.000474349	7:1 180
RANDOM	8	4	DB	CART	0.998690671	0.001224765	0.000387305	7:1 180
RANDOM	8	4	SB	CART	0.998690403	0.000982086	0.000310563	7:1 180
RANDOM	8	4	NRSB	CART	0.99836307	0.001035116	0.000327332	7:1 180
RANDOM	8	5	D	CART	0.998854337	0.001047975	0.000331399	7:1 180
RANDOM	8	5	DB	CART	0.998690671	0.001908659	0.000603571	7:1 180
RANDOM	8	5	SB	CART	0.998527005	0.001145663	0.00036229	7:1 180
RANDOM	8	5	NRSB	CART	0.998199404	0.00088132	0.000278698	7:1 180
RANDOM	8	6	D	CART	0.998199404	0.001359481	0.000429906	7:1 180
RANDOM	8	6	DB	CART	0.998690671	0.000981997	0.000310535	7:1 180
RANDOM	8	6	SB	CART	0.998527005	0.002250037	0.000711524	7:1 180
RANDOM	8	6	NRSB	CART	0.998854337	0.001047975	0.000331399	7:1 180
RANDOM	8	7	D	CART	0.999018003	0.001669073	0.000527807	7:1 180
RANDOM	8	7	DB	CART	0.998854337	0.001943428	0.000614566	7:1 180
RANDOM	8	7	SB	CART	0.998035738	0.002044146	0.000646416	7:1 180
RANDOM	8	7	NRSB	CART	0.999018003	0.001309329	0.000414046	7:1 180



Table D.5: Implementation of the subsystem of data partitioning by DSS, multi-node, single-classifier (CART) for 10 disjoint partitions:

Multi-Node Ensemble and Classifier.

ORDER	NODE	Part #	Partitic	Classifier	Accuracy	Standard Deviati	Standard Errc	CP (Master	Runtime(S
RANDOM	10	0	D	Decision Tree-CART	0.99754476	0.001531116	0.00048418	8:2	180
RANDOM	10	0	DB	Decision Tree-CART	0.99733942	0.003179674	0.0010055	8:2	180
RANDOM	10	0	SB	Decision Tree-CART	0.99815825	0.001699858	0.00053754	8:2	180
RANDOM	10	0	NRSB	Decision Tree-CART	0.99754476	0.001783456	0.00056398	8:2	180
RANDOM	10	1	D	Decision Tree-CART	0.99897709	0.001372134	0.00043391	8:2	180
RANDOM	10	1	DB	Decision Tree-CART	0.99754476	0.00238679	0.00075477	8:2	180
RANDOM	10	1	SB	Decision Tree-CART	0.99836317	0.002203456	0.00069679	8:2	180
RANDOM	10	1	NRSB	Decision Tree-CART	0.99856809	0.001309629	0.00041414	8:2	180
RANDOM	10	2	D	Decision Tree-CART	0.99856725	0.001598746	0.00050557	8:2	180
RANDOM	10	2	DB	Decision Tree-CART	0.99815825	0.00193204	0.00061096	8:2	180
RANDOM	10	2	SB	Decision Tree-CART	0.99877301	0.001874264	0.00059269	8:2	180
RANDOM	10	2	NRSB	Decision Tree-CART	0.99836317	0.001782974	0.00056383	8:2	180
RANDOM	10	3	D	Decision Tree-CART	0.99815825	0.002135957	0.00067545	8:2	180
RANDOM	10	3	DB	Decision Tree-CART	0.99815909	0.002322699	0.0007345	8:2	180
RANDOM	10	3	SB	Decision Tree-CART	0.99836359	0.0022026	0.00069652	8:2	180
RANDOM	10	3	NRSB	Decision Tree-CART	0.99815909	0.001431553	0.0004527	8:2	180
RANDOM	10	4	D	Decision Tree-CART	0.99795375	0.001830033	0.00057871	8:2	180
RANDOM	10	4	DB	Decision Tree-CART	0.99897751	0.001371821	0.00043381	8:2	180
RANDOM	10	4	SB	Decision Tree-CART	0.99713576	0.002085819	0.00065959	8:2	180
RANDOM	10	4	NRSB	Decision Tree-CART	0.99877301	0.001874264	0.00059269	8:2	180
RANDOM	10	5	D	Decision Tree-CART	0.99815825	0.001432872	0.00045311	8:2	180
RANDOM	10	5	DB	Decision Tree-CART	0.99836359	0.001782877	0.0005638	8:2	180
RANDOM	10	5	SB	Decision Tree-CART	0.99775051	0.002496228	0.00078938	8:2	180
RANDOM	10	5	NRSB	Decision Tree-CART	0.99836359	0.002003759	0.00063364	8:2	180
RANDOM	10	6	D	Decision Tree-CART	0.99815825	0.001699858	0.00053754	8:2	180
RANDOM	10	6	DB	Decision Tree-CART	0.99795459	0.001829095	0.00057841	8:2	180
RANDOM	10	6	SB	Decision Tree-CART	0.99795417	0.001294692	0.00040942	8:2	180
RANDOM	10	6	NRSB	Decision Tree-CART	0.99856809	0.001309629	0.00041414	8:2	180
RANDOM	10	7	D	Decision Tree-CART	0.99774842	0.001103986	0.00034911	8:2	180
RANDOM	10	7	DB	Decision Tree-CART	0.99897751	0.001648723	0.00052137	8:2	180
RANDOM	10	7	SB	Decision Tree-CART	0.99897751	0.001371821	0.00043381	8:2	180
RANDOM	10	7	NRSB	Decision Tree-CART	0.99713576	0.001874632	0.00059281	8:2	180
RANDOM	10	8	D	Decision Tree-CART	0.99897667	0.001372446	0.00043401	8:2	180
RANDOM	10	8	DB	Decision Tree-CART	0.9969321	0.003452216	0.00109169	8:2	180
RANDOM	10	8	SB	Decision Tree-CART	0.99815909	0.001698747	0.00053719	8:2	180
RANDOM	10	8	NRSB	Decision Tree-CART	0.99815867	0.001698798	0.00053721	8:2	180
RANDOM	10	9	D	Decision Tree-CART	0.99877091	0.001877922	0.00059385	8:2	180
RANDOM	10	9	DB	Decision Tree-CART	0.99734109	0.001597028	0.00050502	8:2	180
RANDOM	10	9	SB	Decision Tree-CART	0.99815867	0.001431614	0.00045272	8:2	180
RANDOM	10	9	NRSB	Decision Tree-CART	0.99897709	0.001372134	0.00043391	8:2	180

Table D.6: Implementation of the subsystem of data partitioning by DSS, multi-node, single-classifier (CART) for 12 disjoint partitions:

Multi-Node Ensemble and Classifier.

ORDER	NODE #	Part #	Partiti	Classifier	Accuracy	Standard Deviation	Standard Error of CP	(Master Runtime(S	
RANDOM	12	0	D	CART	0.998772101	0.0012279	0.000388296	8:4	120
RANDOM	12	0	DB	CART	0.998526401	0.00120319	0.000380482	8:4	120
RANDOM	12	0	SB	CART	0.998772101	0.0012279	0.000388296	8:4	120
RANDOM	12	0	NRSB	CART	0.997790504	0.002563862	0.000810764	8:4	120
RANDOM	12	1	D	CART	0.998526401	0.00120319	0.000380482	8:4	120
RANDOM	12	1	DB	CART	0.996317507	0.002746208	0.000868427	8:4	120
RANDOM	12	1	SB	CART	0.999018403	0.001202207	0.000380171	8:4	120
RANDOM	12	1	NRSB	CART	0.998526401	0.001629429	0.000515271	8:4	120
RANDOM	12	2	D	CART	0.998281303	0.001918519	0.000606689	8:4	120
RANDOM	12	2	DB	CART	0.997545406	0.002196266	0.00069452	8:4	120
RANDOM	12	2	SB	CART	0.998035	0.001838492	0.000581382	8:4	120
RANDOM	12	2	NRSB	CART	0.998527003	0.001963496	0.000620912	8:4	120
RANDOM	12	3	D	CART	0.9982807	0.001572968	0.000497416	8:4	120
RANDOM	12	3	DB	CART	0.998772703	0.001645516	0.000520358	8:4	120
RANDOM	12	3	SB	CART	0.997791709	0.002783992	0.000880376	8:4	120
RANDOM	12	3	NRSB	CART	0.998772101	0.001647759	0.000521067	8:4	120
RANDOM	12	4	D	CART	0.998035602	0.001473802	0.000466057	8:4	120
RANDOM	12	4	DB	CART	0.997791107	0.001322692	0.000418272	8:4	120
RANDOM	12	4	SB	CART	0.997053404	0.002645493	0.000836578	8:4	120
RANDOM	12	4	NRSB	CART	0.998526401	0.001629429	0.000515271	8:4	120
RANDOM	12	5	D	CART	0.998527003	0.001629066	0.000515156	8:4	120
RANDOM	12	5	DB	CART	0.999017801	0.001202944	0.000380404	8:4	120
RANDOM	12	5	SB	CART	0.998526401	0.001629429	0.000515271	8:4	120
RANDOM	12	5	NRSB	CART	0.999017199	0.00120368	0.000380637	8:4	120
RANDOM	12	6	D	CART	0.998772101	0.001647759	0.000521067	8:4	120
RANDOM	12	6	DB	CART	0.998772101	0.001980526	0.000626297	8:4	120
RANDOM	12	6	SB	CART	0.999018403	0.001202207	0.000380171	8:4	120
RANDOM	12	6	NRSB	CART	0.999018403	0.001626887	0.000514467	8:4	120
RANDOM	12	7	D	CART	0.998527605	0.001626888	0.000514467	8:4	120
RANDOM	12	7	DB	CART	0.998037409	0.002401831	0.000759526	8:4	120
RANDOM	12	7	SB	CART	0.998035602	0.001473802	0.000466057	8:4	120
RANDOM	12	7	NRSB	CART	0.997792311	0.00299294	0.000946451	8:4	120
RANDOM	12	8	D	CART	0.9982807	0.001572968	0.000497416	8:4	120
RANDOM	12	8	DB	CART	0.9982807	0.001572968	0.000497416	8:4	120
RANDOM	12	8	SB	CART	0.998037409	0.002401831	0.000759526	8:4	120
RANDOM	12	8	NRSB	CART	0.997545406	0.002193573	0.000693669	8:4	120
RANDOM	12	9	D	CART	0.997790504	0.001717926	0.000543256	8:4	120
RANDOM	12	9	DB	CART	0.999264104	0.001570052	0.000496494	8:4	120
RANDOM	12	9	SB	CART	0.997791107	0.002039203	0.000644853	8:4	120
RANDOM	12	9	NRSB	CART	0.998525799	0.001629791	0.000515385	8:4	120
RANDOM	12	10	D	CART	0.998034398	0.001474201	0.000466183	8:4	120
RANDOM	12	10	DB	CART	0.996807703	0.002703197	0.000854826	8:4	120
RANDOM	12	10	SB	CART	0.998772703	0.00197866	0.000625707	8:4	120
RANDOM	12	10	NRSB	CART	0.997054006	0.003063399	0.000968732	8:4	120
RANDOM	12	11	D	CART	0.998036205	0.002642585	0.000835659	8:4	120
RANDOM	12	11	DB	CART	0.997546611	0.00290106	0.000917396	8:4	120
RANDOM	12	11	SB	CART	0.998526401	0.001629429	0.000515271	8:4	120
RANDOM	12	11	NRSB	CART	0.998281303	0.002701281	0.00085422	8:4	120

## Appendix E: Experimental results of DELS system

Table E.1: Implementation of the subsystem of data partitioning by DSS, multi-node, single-classifier (MLP) for 2 disjoint partitions: Single-Node Ensemble and Classifier.

ORDER	NODE #	Part #	Partition	Classifier	Accuracy	Standard D	Standard Error	CP (Master)	Runtime(Seconds)
RANDOM	2	0	D	MLP	0.97761494	0.004914	0.0015541	1:1	6180
RANDOM	2	0	DB	MLP	0.973400788	0.002518	0.000796416	1:1	6180
RANDOM	2	0	SB	MLP	0.972499586	0.002434	0.000769582	1:1	6180
RANDOM	2	0	NRSB	MLP	0.975077964	0.006757	0.002136732	1:1	6180
RANDOM	2	1	D	MLP	0.972743846	0.003037	0.000960459	1:1	6180
RANDOM	2	1	DB	MLP	0.97164044	0.003617	0.001143921	1:1	6180
RANDOM	2	1	SB	MLP	0.97438168	0.003677	0.001162923	1:1	6180
RANDOM	2	1	NRSB	MLP	0.972417703	0.002228	0.00070463	1:1	6180

Table E.2: Implementation of the subsystem of data partitioning by DSS, multi-node, single-classifier (MLP) for 4 disjoint partitions: Single-Node Ensemble and Classifier.

ORDER	NODE #	Part #	Partition	Classifier	Accuracy	Standard D	Standard Error	CP (Master)	Runtime(Seconds)
RANDOM	4	0	D	MLP	0.967834	0.005114	0.001617	3:1	2280
RANDOM	4	0	DB	MLP	0.971601	0.003846	0.001216	3:1	2280
RANDOM	4	0	SB	MLP	0.968243	0.005539	0.001751	3:1	2280
RANDOM	4	0	NRSB	MLP	0.965952	0.004914	0.001554	3:1	2280
RANDOM	4	1	D	MLP	0.967916	0.004235	0.001339	3:1	2280
RANDOM	4	1	DB	MLP	0.969474	0.003086	0.000976	3:1	2280
RANDOM	4	1	SB	MLP	0.958668	0.008723	0.002759	3:1	2280
RANDOM	4	1	NRSB	MLP	0.970618	0.004677	0.001479	3:1	2280
RANDOM	4	2	D	MLP	0.971762	0.005784	0.001829	3:1	2280
RANDOM	4	2	DB	MLP	0.968983	0.003632	0.001148	3:1	2280
RANDOM	4	2	SB	MLP	0.970371	0.00306	0.000968	3:1	2280
RANDOM	4	2	NRSB	MLP	0.969636	0.004775	0.00151	3:1	2280
RANDOM	4	3	D	MLP	0.970041	0.004473	0.001414	3:1	2280
RANDOM	4	3	DB	MLP	0.968653	0.005575	0.001763	3:1	2280
RANDOM	4	3	SB	MLP	0.97209	0.004276	0.001352	3:1	2280
RANDOM	4	3	NRSB	MLP	0.970535	0.004885	0.001545	3:1	2280

Table E.3: Implementation of the subsystem of data partitioning by DSS, multi-node, single-classifier (MLP) for 6 disjoint partitions: Single-Node Ensemble and Classifier.

ORDER	NODE #	Part #	Partition	Classifier	Accuracy	Standard De	Standard E	CP (Master	Runtime(S
RANDOM	6	0	D	MLP	0.969188	0.0085979	0.002719	4:02	1560
RANDOM	6	0	DB	MLP	0.966491	0.0064921	0.002053	4:02	1560
RANDOM	6	0	SB	MLP	0.964647	0.0072276	0.002286	4:02	1560
RANDOM	6	0	NRSB	MLP	0.971276	0.005118	0.001618	4:02	1560
RANDOM	6	1	D	MLP	0.960712	0.0073257	0.002317	4:02	1560
RANDOM	6	1	DB	MLP	0.968698	0.0048436	0.001532	4:02	1560
RANDOM	6	1	SB	MLP	0.967348	0.005725	0.00181	4:02	1560
RANDOM	6	1	NRSB	MLP	0.96882	0.0056262	0.001779	4:02	1560
RANDOM	6	2	D	MLP	0.967466	0.003249	0.001027	4:02	1560
RANDOM	6	2	DB	MLP	0.968696	0.0064115	0.002027	4:02	1560
RANDOM	6	2	SB	MLP	0.96759	0.006413	0.002028	4:02	1560
RANDOM	6	2	NRSB	MLP	0.967344	0.0055883	0.001767	4:02	1560
RANDOM	6	3	D	MLP	0.969674	0.0057657	0.001823	4:02	1560
RANDOM	6	3	DB	MLP	0.964766	0.0065081	0.002058	4:02	1560
RANDOM	6	3	SB	MLP	0.959243	0.0077468	0.00245	4:02	1560
RANDOM	6	3	NRSB	MLP	0.970045	0.0045785	0.001448	4:02	1560
RANDOM	6	4	D	MLP	0.969431	0.0041483	0.001312	4:02	1560
RANDOM	6	4	DB	MLP	0.968451	0.0055719	0.001762	4:02	1560
RANDOM	6	4	SB	MLP	0.967348	0.0070922	0.002243	4:02	1560
RANDOM	6	4	NRSB	MLP	0.965628	0.0051146	0.001617	4:02	1560
RANDOM	6	5	D	MLP	0.964028	0.0041393	0.001309	4:02	1560
RANDOM	6	5	DB	MLP	0.968328	0.0048716	0.001541	4:02	1560
RANDOM	6	5	SB	MLP	0.9644	0.003995	0.001263	4:02	1560
RANDOM	6	5	NRSB	MLP	0.960472	0.0070662	0.002235	4:02	1560

Table E.4: Implementation of the subsystem of data partitioning by DSS, multi-node, single-classifier (MLP) for 8 disjoint partitions:

Single-Node Ensemble and Classifier.

ORDER	NODE #	Part #	Partitionin;	Classifier	Accuracy	Standard D	Standard E	CP (Master	Runtime(Sec)
RANDOM	8	0	D	MLP	0.966117	0.006746	0.002133	7:1	960
RANDOM	8	0	DB	MLP	0.969231	0.006245	0.001975	7:1	960
RANDOM	8	0	SB	MLP	0.969062	0.004481	0.001417	7:1	960
RANDOM	8	0	NRSB	MLP	0.961041	0.01182	0.003738	7:1	960
RANDOM	8	1	D	MLP	0.962678	0.0072	0.002277	7:1	960
RANDOM	8	1	DB	MLP	0.970213	0.006202	0.001961	7:1	960
RANDOM	8	1	SB	MLP	0.965296	0.010194	0.003223	7:1	960
RANDOM	8	1	NRSB	MLP	0.96759	0.005158	0.001631	7:1	960
RANDOM	8	2	D	MLP	0.965459	0.008528	0.002697	7:1	960
RANDOM	8	2	DB	MLP	0.966121	0.005126	0.001621	7:1	960
RANDOM	8	2	SB	MLP	0.966773	0.008048	0.002545	7:1	960
RANDOM	8	2	NRSB	MLP	0.95482	0.009414	0.002977	7:1	960
RANDOM	8	3	D	MLP	0.965462	0.010815	0.00342	7:1	960
RANDOM	8	3	DB	MLP	0.962193	0.003835	0.001213	7:1	960
RANDOM	8	3	SB	MLP	0.965299	0.006693	0.002117	7:1	960
RANDOM	8	3	NRSB	MLP	0.965952	0.004787	0.001514	7:1	960
RANDOM	8	4	D	MLP	0.967917	0.006469	0.002046	7:1	960
RANDOM	8	4	DB	MLP	0.96252	0.008736	0.002763	7:1	960
RANDOM	8	4	SB	MLP	0.966113	0.008073	0.002553	7:1	960
RANDOM	8	4	NRSB	MLP	0.963332	0.009666	0.003057	7:1	960
RANDOM	8	5	D	MLP	0.966934	0.008145	0.002576	7:1	960
RANDOM	8	5	DB	MLP	0.962684	0.005754	0.00182	7:1	960
RANDOM	8	5	SB	MLP	0.969062	0.003226	0.00102	7:1	960
RANDOM	8	5	NRSB	MLP	0.963003	0.011401	0.003605	7:1	960
RANDOM	8	6	D	MLP	0.966279	0.004463	0.001411	7:1	960
RANDOM	8	6	DB	MLP	0.958756	0.010346	0.003272	7:1	960
RANDOM	8	6	SB	MLP	0.969225	0.00496	0.001568	7:1	960
RANDOM	8	6	NRSB	MLP	0.966772	0.007092	0.002243	7:1	960
RANDOM	8	7	D	MLP	0.967748	0.006943	0.002196	7:1	960
RANDOM	8	7	DB	MLP	0.967755	0.005851	0.00185	7:1	960
RANDOM	8	7	SB	MLP	0.969226	0.006619	0.002093	7:1	960
RANDOM	8	7	NRSB	MLP	0.958095	0.008819	0.002789	7:1	960

Table E.5: Implementation of the subsystem of data partitioning by DSS, multi-node, single-classifier (MLP) for 10 disjoint partitions:

Single-Node Ensemble and Classifier.

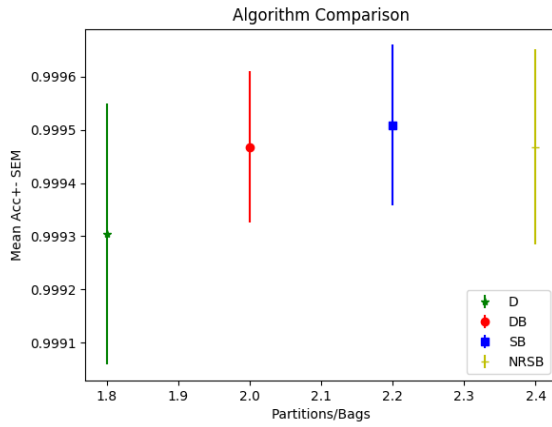
ORDER	NODE #	Part #	Partitioning	Classifier	Accuracy	Standard D	Standard E	CP (Master)	Runtime(S)
RANDOM	10	0	D	MLP	0.967881	0.007598	0.002403	8:2	780
RANDOM	10	0	DB	MLP	0.964411	0.006852	0.002167	8:2	780
RANDOM	10	0	SB	MLP	0.955605	0.008286	0.00262	8:2	780
RANDOM	10	0	NRSB	MLP	0.965222	0.003651	0.001155	8:2	780
RANDOM	10	1	D	MLP	0.966232	0.008469	0.002678	8:2	780
RANDOM	10	1	DB	MLP	0.967466	0.008031	0.00254	8:2	780
RANDOM	10	1	SB	MLP	0.967062	0.00631	0.001995	8:2	780
RANDOM	10	1	NRSB	MLP	0.965839	0.01058	0.003346	8:2	780
RANDOM	10	2	D	MLP	0.967465	0.008258	0.002611	8:2	780
RANDOM	10	2	DB	MLP	0.961741	0.009433	0.002983	8:2	780
RANDOM	10	2	SB	MLP	0.963585	0.004735	0.001497	8:2	780
RANDOM	10	2	NRSB	MLP	0.961128	0.004933	0.00156	8:2	780
RANDOM	10	3	D	MLP	0.962964	0.008887	0.00281	8:2	780
RANDOM	10	3	DB	MLP	0.959083	0.005787	0.00183	8:2	780
RANDOM	10	3	SB	MLP	0.959075	0.012411	0.003925	8:2	780
RANDOM	10	3	NRSB	MLP	0.960308	0.008309	0.002627	8:2	780
RANDOM	10	4	D	MLP	0.96624	0.004579	0.001448	8:2	780
RANDOM	10	4	DB	MLP	0.964813	0.005764	0.001823	8:2	780
RANDOM	10	4	SB	MLP	0.969922	0.011809	0.003734	8:2	780
RANDOM	10	4	NRSB	MLP	0.965429	0.008194	0.002591	8:2	780
RANDOM	10	5	D	MLP	0.960713	0.00786	0.002486	8:2	780
RANDOM	10	5	DB	MLP	0.966859	0.007471	0.002362	8:2	780
RANDOM	10	5	SB	MLP	0.960721	0.009627	0.003044	8:2	780
RANDOM	10	5	NRSB	MLP	0.962564	0.007535	0.002383	8:2	780
RANDOM	10	6	D	MLP	0.961325	0.009079	0.002871	8:2	780
RANDOM	10	6	DB	MLP	0.965019	0.006354	0.002009	8:2	780
RANDOM	10	6	SB	MLP	0.960722	0.014738	0.004661	8:2	780
RANDOM	10	6	NRSB	MLP	0.96706	0.010456	0.003306	8:2	780
RANDOM	10	7	D	MLP	0.966031	0.008355	0.002642	8:2	780
RANDOM	10	7	DB	MLP	0.963992	0.008497	0.002687	8:2	780
RANDOM	10	7	SB	MLP	0.961948	0.006349	0.002008	8:2	780
RANDOM	10	7	NRSB	MLP	0.968498	0.006905	0.002183	8:2	780
RANDOM	10	8	D	MLP	0.957639	0.0087	0.002751	8:2	780
RANDOM	10	8	DB	MLP	0.966859	0.007068	0.002235	8:2	780
RANDOM	10	8	SB	MLP	0.963585	0.004641	0.001468	8:2	780
RANDOM	10	8	NRSB	MLP	0.961538	0.00993	0.00314	8:2	780
RANDOM	10	9	D	MLP	0.962349	0.008295	0.002623	8:2	780
RANDOM	10	9	DB	MLP	0.962971	0.006233	0.001971	8:2	780
RANDOM	10	9	SB	MLP	0.969109	0.006681	0.002113	8:2	780
RANDOM	10	9	NRSB	MLP	0.958879	0.006499	0.002055	8:2	780

Table E.6: Implementation of the subsystem of data partitioning by DSS, multi-node, single-classifier (MLP) for 12 disjoint partitions:

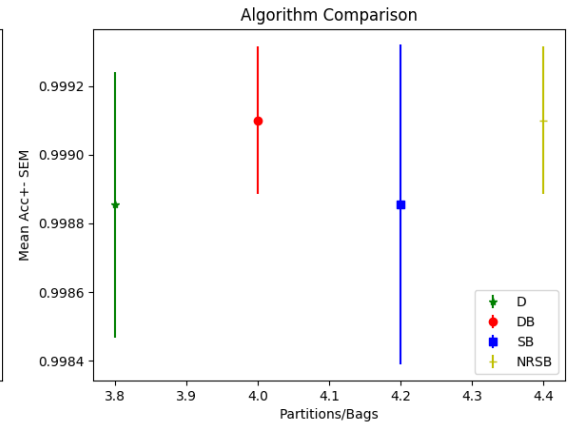
Single-Node Ensemble and Classifier.

ORDER	NODE #	Part #	Partitionin	Classifier	Accuracy	Standard D	Standard E	CP (Master	Runtime(S
RANDOM	12	0	D	MLP	0.962928	0.010326	0.003265	8:4	960
RANDOM	12	0	DB	MLP	0.96269	0.010172	0.003217	8:4	960
RANDOM	12	0	SB	MLP	0.956052	0.006439	0.002036	8:4	960
RANDOM	12	0	NRSB	MLP	0.968574	0.00884	0.002795	8:4	960
RANDOM	12	1	D	MLP	0.960223	0.00439	0.001388	8:4	960
RANDOM	12	1	DB	MLP	0.959494	0.011286	0.003569	8:4	960
RANDOM	12	1	SB	MLP	0.957284	0.014064	0.004448	8:4	960
RANDOM	12	1	NRSB	MLP	0.955559	0.010849	0.003431	8:4	960
RANDOM	12	2	D	MLP	0.960964	0.009471	0.002995	8:4	960
RANDOM	12	2	DB	MLP	0.968338	0.009398	0.002972	8:4	960
RANDOM	12	2	SB	MLP	0.955555	0.010579	0.003345	8:4	960
RANDOM	12	2	NRSB	MLP	0.965627	0.009049	0.002862	8:4	960
RANDOM	12	3	D	MLP	0.947467	0.012726	0.004024	8:4	960
RANDOM	12	3	DB	MLP	0.958517	0.009144	0.002892	8:4	960
RANDOM	12	3	SB	MLP	0.95974	0.011565	0.003657	8:4	960
RANDOM	12	3	NRSB	MLP	0.96268	0.008558	0.002706	8:4	960
RANDOM	12	4	D	MLP	0.964891	0.00769	0.002432	8:4	960
RANDOM	12	4	DB	MLP	0.963424	0.010891	0.003444	8:4	960
RANDOM	12	4	SB	MLP	0.958016	0.010673	0.003375	8:4	960
RANDOM	12	4	NRSB	MLP	0.961702	0.00865	0.002735	8:4	960
RANDOM	12	5	D	MLP	0.959738	0.011187	0.003538	8:4	960
RANDOM	12	5	DB	MLP	0.955823	0.009358	0.002959	8:4	960
RANDOM	12	5	SB	MLP	0.961697	0.009785	0.003094	8:4	960
RANDOM	12	5	NRSB	MLP	0.962675	0.010012	0.003166	8:4	960
RANDOM	12	6	D	MLP	0.956547	0.007176	0.002269	8:4	960
RANDOM	12	6	DB	MLP	0.961709	0.010141	0.003207	8:4	960
RANDOM	12	6	SB	MLP	0.962188	0.008171	0.002584	8:4	960
RANDOM	12	6	NRSB	MLP	0.961944	0.009012	0.00285	8:4	960
RANDOM	12	7	D	MLP	0.967583	0.004641	0.001468	8:4	960
RANDOM	12	7	DB	MLP	0.965621	0.00914	0.00289	8:4	960
RANDOM	12	7	SB	MLP	0.959734	0.006333	0.002003	8:4	960
RANDOM	12	7	NRSB	MLP	0.963663	0.010513	0.003325	8:4	960
RANDOM	12	8	D	MLP	0.965619	0.004526	0.001431	8:4	960
RANDOM	12	8	DB	MLP	0.960958	0.00718	0.002271	8:4	960
RANDOM	12	8	SB	MLP	0.966614	0.010348	0.003272	8:4	960
RANDOM	12	8	NRSB	MLP	0.959252	0.012056	0.003812	8:4	960
RANDOM	12	9	D	MLP	0.955552	0.015275	0.00483	8:4	960
RANDOM	12	9	DB	MLP	0.965875	0.010322	0.003264	8:4	960
RANDOM	12	9	SB	MLP	0.957279	0.009962	0.00315	8:4	960
RANDOM	12	9	NRSB	MLP	0.959738	0.0056	0.001771	8:4	960
RANDOM	12	10	D	MLP	0.96685	0.00958	0.003029	8:4	960
RANDOM	12	10	DB	MLP	0.963417	0.00818	0.002587	8:4	960
RANDOM	12	10	SB	MLP	0.958751	0.008715	0.002756	8:4	960
RANDOM	12	10	NRSB	MLP	0.96538	0.007569	0.002393	8:4	960
RANDOM	12	11	D	MLP	0.966112	0.007177	0.00227	8:4	960
RANDOM	12	11	DB	MLP	0.961945	0.009006	0.002848	8:4	960
RANDOM	12	11	SB	MLP	0.958265	0.011966	0.003784	8:4	960
RANDOM	12	11	NRSB	MLP	0.96096	0.008963	0.002834	8:4	960

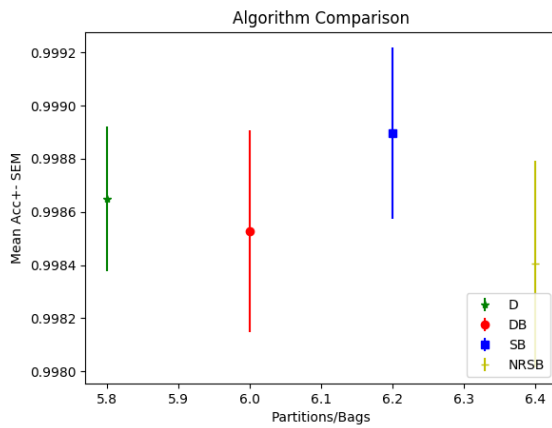
## Appendix F: Comparison on “KDDCUP” data set with CART classifier on single node



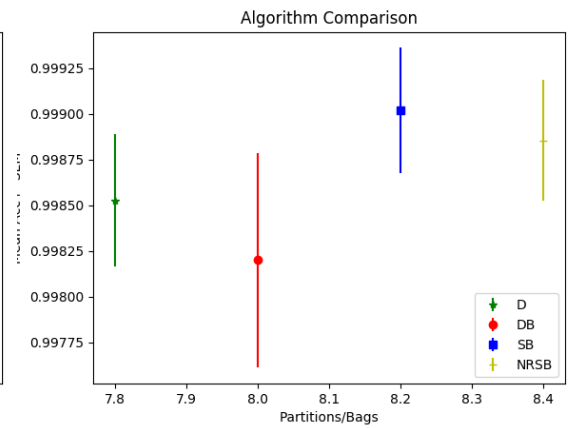
Plot F.1 CART classifier on 2 partitions, DS



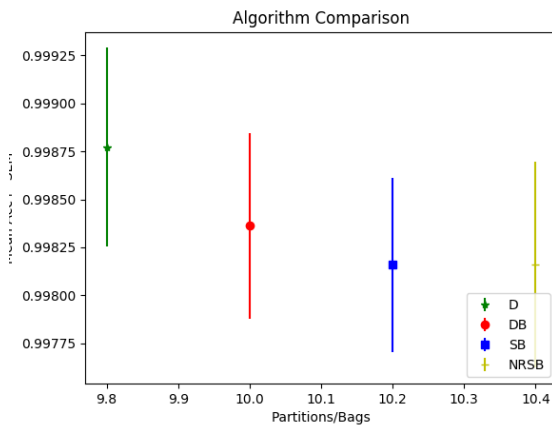
Plot F.2 CART classifier on 4 partitions, DS



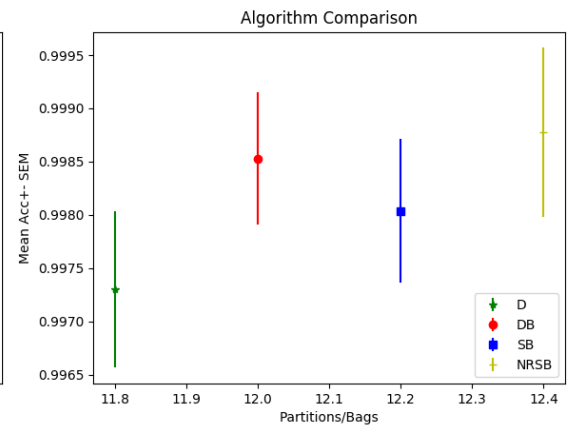
Plot F.3 CART classifier on 6 partitions, DS



Plot F.4 CART classifier on 8 partitions, DS



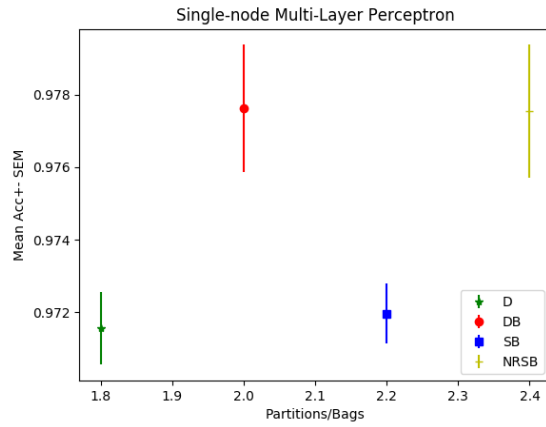
Plot F.5 CART classifier on 10 partitions, DS



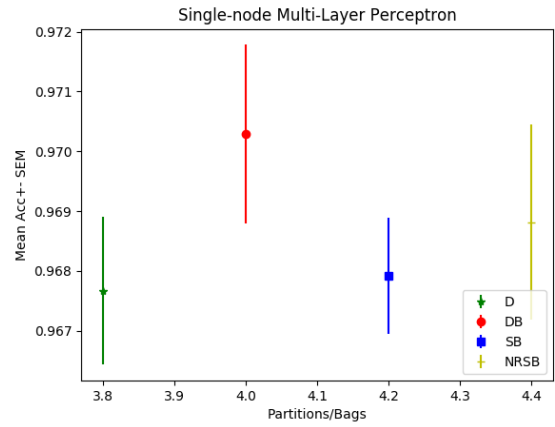
Plot F.6 CART classifier on 12 partition, DS



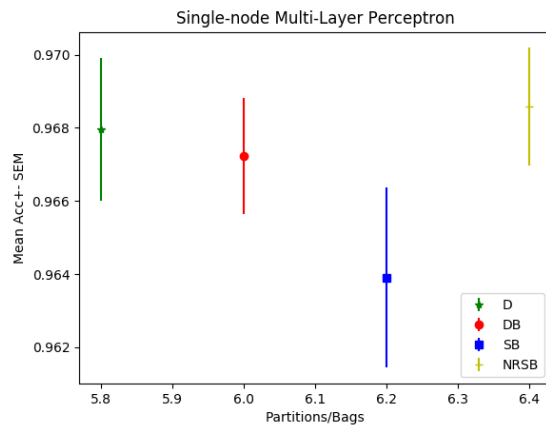
## Appendix G: Comparison on “KDDCUP” data set with MLP classifier on single node



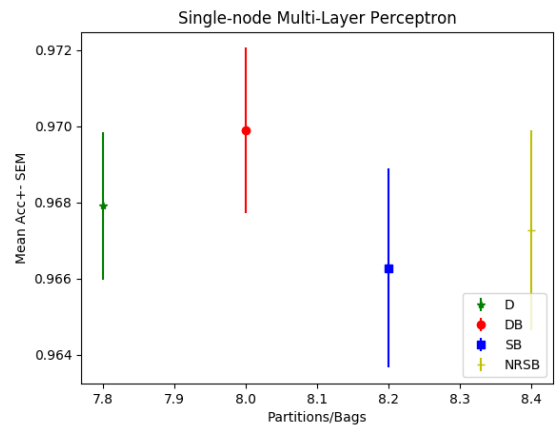
Plot G.1 MLP classifier on 2 partitions, DS



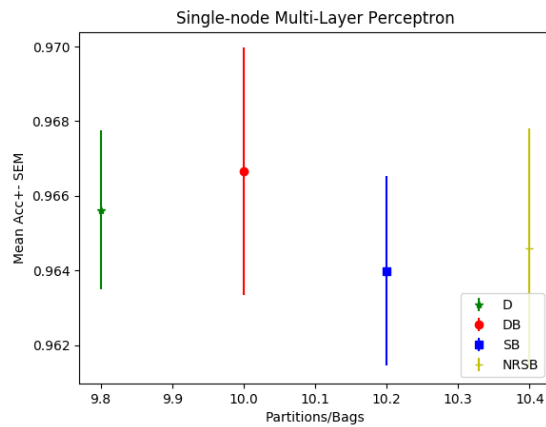
Plot G.2 MLP classifier on 4 partitions, DS



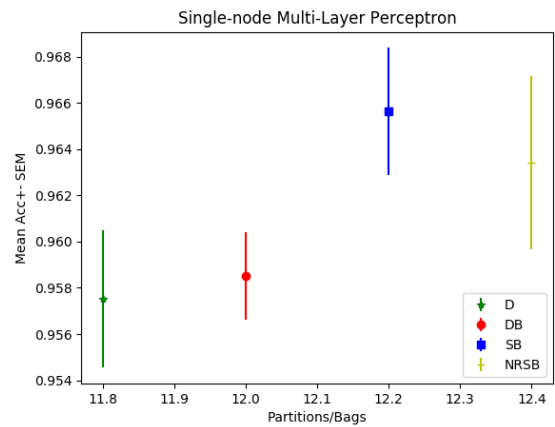
Plot G.3 MLP classifier on 6 partitions, DS



Plot G.4 MLP classifier on 8 partitions, DS

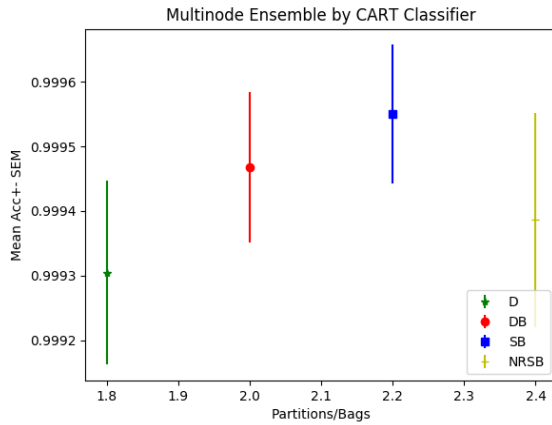


Plot G.5 MLP classifier on 10 partitions, DS

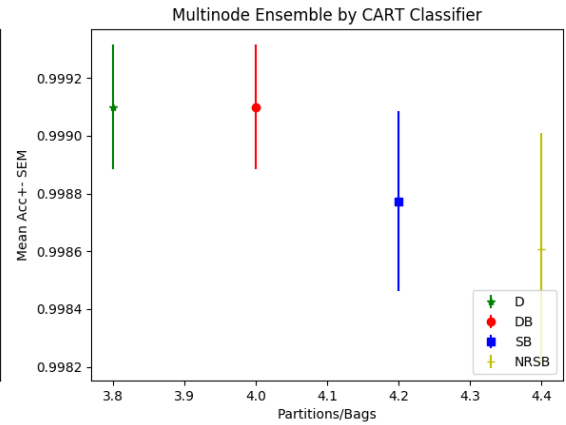


Plot G.6 MLP classifier on 12 partitions, DS

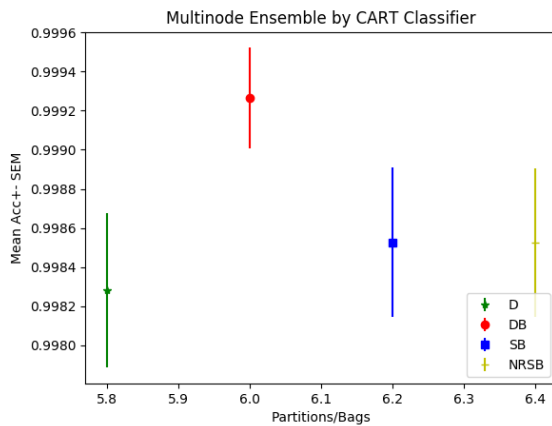
## Appendix H: Comparison on “KDDCUP” data set with CART classifier on multi-node system



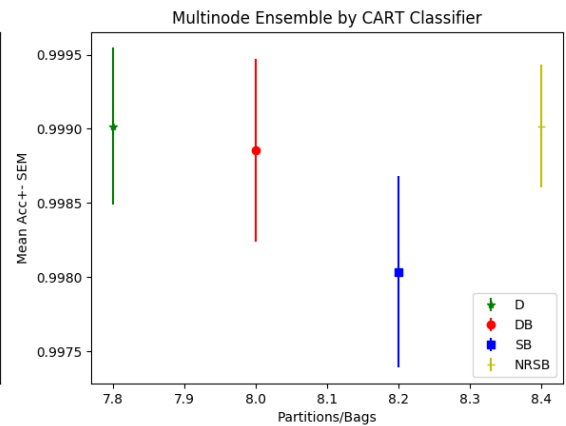
Plot H.1 CART classifier on 2 partitions, DS



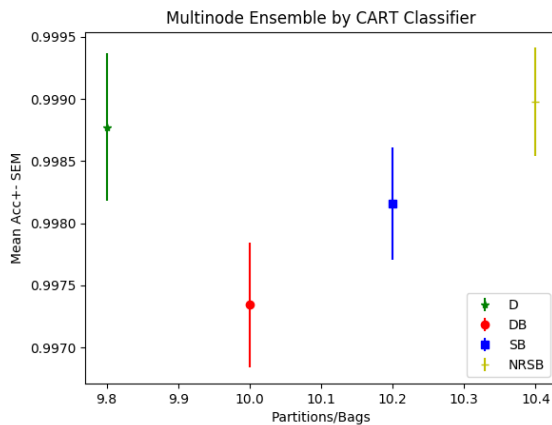
Plot H.2 CART classifier on 4 partitions, DS



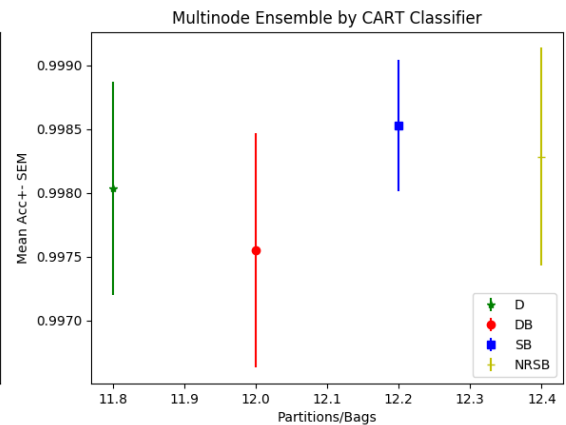
Plot H.3 CART classifier on 6 partitions, DS



Plot H.4 CART classifier on 8 partitions, DS

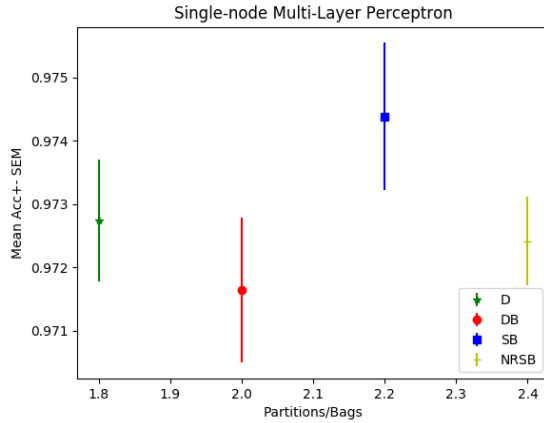


Plot H.5 CART classifier on 10 partitions, DS

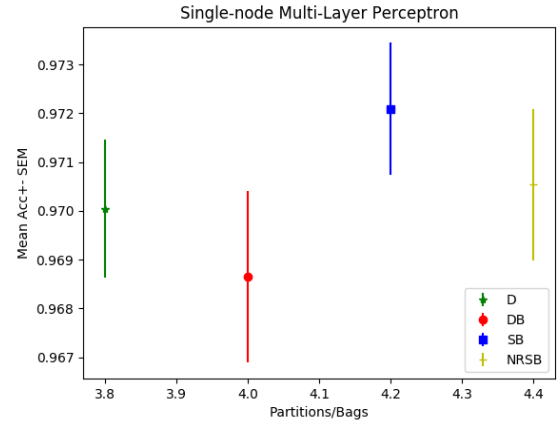


Plot H.6 CART classifier on 12 partitions, DS

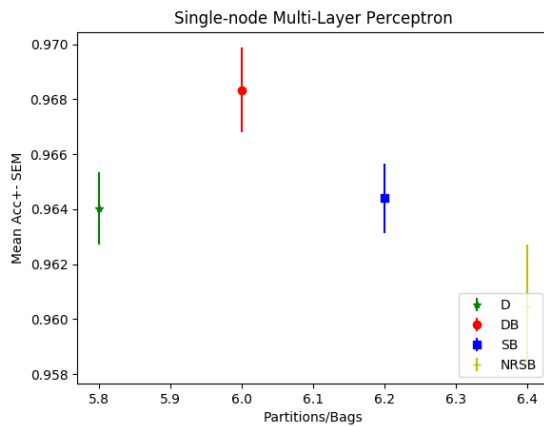
## Appendix I: Comparison on “KDDCUP” data set with MLP classifier on multi-node System



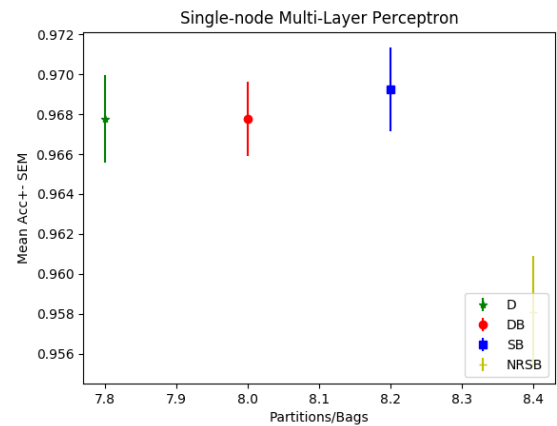
Plot I.1 MLP classifier on 2 partitions, DS



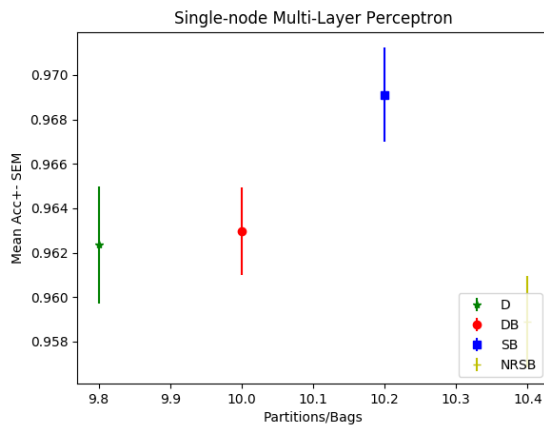
Plot I.2 MLP classifier on 4 partitions, DS



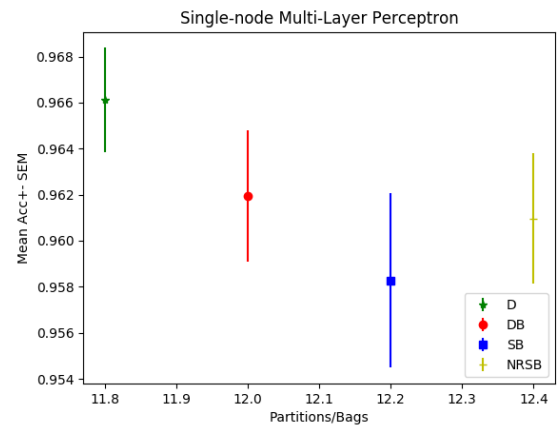
Plot I.3 MLP classifier on 6 partitions, DS



Plot I.4 MLP classifier on 8 partitions, DS



Plot I.5 MLP classifier on 10 partitions, DS



Plot I.6 MLP classifier on 12 partitions, DS

## Appendix J: Experimental results of DELS system

Table J.1: Multi-node ensemble-MLP on random input by Bagging-like method-2 dataset partitions:

NC Par	Partiti	Class	Cross-Val-Scc	Test-Score	Time of Par	Time of Transf	Time of Training	Ensemble-S
2	0	D	MLP	0.82968834	0.955467	1.700584 ('Node0', 0)	('Node0', 15.09672)	1901.261
2	0	DB	MLP	0.82968834	0.955467	2.323832 ('Node0', 0)	('Node0', 3.041651)	1901.261
2	0	SB	MLP	0.82968834	0.955467	0.466872 ('Node0', 0)	('Node0', 6.798189)	1901.261
2	0	NRSB	MLP	0.82968834	0.955467	0.487792 ('Node0', 0)	('Node0', 15.36328)	1901.261
2	1	D	MLP	0.82968834	0.955467	1.700584 ('Node1', 10.71)	('Node1', 17.68260)	1901.261
2	1	DB	MLP	0.82968834	0.955467	2.323832 ('Node1', 10.71)	('Node1', 7.907098)	1901.261
2	1	SB	MLP	0.82968834	0.955467	0.466872 ('Node1', 10.71)	('Node1', 8.230186)	1901.261
2	1	NRSB	MLP	0.82968834	0.955467	0.487792 ('Node1', 10.71)	('Node1', 17.92719)	1901.261

Table J.2: Multi-node ensemble-MLP on random input by Bagging-like method-4 dataset partitions:

ORDER	NO Part	Partiti	Class	Cross-Val-Sc	Test-Score	Time of Partiti	Time of Tran	Time of Training	Ensemble-Scc
RANDOM	4	0	D	MLP	0.8296883	0.9554672	1.77120924 ('Node0', 0)	('Node0', 2.37478)	4099.96613
RANDOM	4	0	DB	MLP	0.8296883	0.9554672	2.158751726 ('Node0', 0)	('Node0', 8.94402)	4099.96613
RANDOM	4	0	SB	MLP	0.8296883	0.9554672	0.426065207 ('Node0', 0)	('Node0', 12.5548)	4099.96613
RANDOM	4	0	NRSB	MLP	0.8296883	0.9554672	0.596661091 ('Node0', 0)	('Node0', 5.99413)	4099.96613
RANDOM	4	1	D	MLP	0.8296883	0.9554672	1.77120924 ('Node0', 0)	('Node1', 5.00672)	4099.96613
RANDOM	4	1	DB	MLP	0.8296883	0.9554672	2.158751726 ('Node0', 0)	('Node1', 9.37188)	4099.96613
RANDOM	4	1	SB	MLP	0.8296883	0.9554672	0.426065207 ('Node0', 0)	('Node1', 2.50551)	4099.96613
RANDOM	4	1	NRSB	MLP	0.8296883	0.9554672	0.596661091 ('Node0', 0)	('Node1', 3.94633)	4099.96613
RANDOM	4	2	D	MLP	0.8296883	0.9554672	1.77120924 ('Node0', 0)	('Node2', 3.37505)	4099.96613
RANDOM	4	2	DB	MLP	0.8296883	0.9554672	2.158751726 ('Node0', 0)	('Node2', 5.40550)	4099.96613
RANDOM	4	2	SB	MLP	0.8296883	0.9554672	0.426065207 ('Node0', 0)	('Node2', 4.57897)	4099.96613
RANDOM	4	2	NRSB	MLP	0.8296883	0.9554672	0.596661091 ('Node0', 0)	('Node2', 3.27178)	4099.96613
RANDOM	4	3	D	MLP	0.8296883	0.9554672	1.77120924 ('Node3', 5.3)	('Node3', 10.0073)	4099.96613
RANDOM	4	3	DB	MLP	0.8296883	0.9554672	2.158751726 ('Node3', 5.3)	('Node3', 4.22596)	4099.96613
RANDOM	4	3	SB	MLP	0.8296883	0.9554672	0.426065207 ('Node3', 5.3)	('Node3', 4.91375)	4099.96613
RANDOM	4	3	NRSB	MLP	0.8296883	0.9554672	0.596661091 ('Node3', 5.3)	('Node3', 3.92149)	4099.96613

Table J.3: Multi-node ensemble-MLP on random input by Bagging-like method-6 dataset partitions:

ORDER	NC Par	Partiti	Class	Cross-Val-	Test-Score	Time of Part	Time of Trans	Time of Training	Ensemble-Scc
RANDOM	6	0	D	MLP	0.82969	0.95547	1.9361341 ('Node0', 0)	('Node0', 6.109520435333252)	5708.42344
RANDOM	6	0	DB	MLP	0.82969	0.95547	2.7440946 ('Node0', 0)	('Node0', 1.467219591140747)	5708.42344
RANDOM	6	0	SB	MLP	0.82969	0.95547	0.5213635 ('Node0', 0)	('Node0', 1.8763961791992188)	5708.42344
RANDOM	6	0	NRSB	MLP	0.82969	0.95547	0.5457218 ('Node0', 0)	('Node0', 8.472907304763794)	5708.42344
RANDOM	6	1	D	MLP	0.82969	0.95547	1.9361341 ('Node0', 0)	('Node1', 3.2719459533691406)	5708.42344
RANDOM	6	1	DB	MLP	0.82969	0.95547	2.7440946 ('Node0', 0)	('Node1', 4.2388317584991455)	5708.42344
RANDOM	6	1	SB	MLP	0.82969	0.95547	0.5213635 ('Node0', 0)	('Node1', 2.7688090801239014)	5708.42344
RANDOM	6	1	NRSB	MLP	0.82969	0.95547	0.5457218 ('Node0', 0)	('Node1', 3.0151898860931396)	5708.42344
RANDOM	6	2	D	MLP	0.82969	0.95547	1.9361341 ('Node0', 0)	('Node2', 12.221603870391846)	5708.42344
RANDOM	6	2	DB	MLP	0.82969	0.95547	2.7440946 ('Node0', 0)	('Node2', 3.6746206283569336)	5708.42344
RANDOM	6	2	SB	MLP	0.82969	0.95547	0.5213635 ('Node0', 0)	('Node2', 2.077237606048584)	5708.42344
RANDOM	6	2	NRSB	MLP	0.82969	0.95547	0.5457218 ('Node0', 0)	('Node2', 2.0695748329162598)	5708.42344
RANDOM	6	3	D	MLP	0.82969	0.95547	1.9361341 ('Node0', 0)	('Node3', 2.9128472805023193)	5708.42344
RANDOM	6	3	DB	MLP	0.82969	0.95547	2.7440946 ('Node0', 0)	('Node3', 8.182551860809326)	5708.42344
RANDOM	6	3	SB	MLP	0.82969	0.95547	0.5213635 ('Node0', 0)	('Node3', 3.125713348388672)	5708.42344
RANDOM	6	3	NRSB	MLP	0.82969	0.95547	0.5457218 ('Node0', 0)	('Node3', 7.693204879760742)	5708.42344
RANDOM	6	4	D	MLP	0.82969	0.95547	1.9361341 ('Node0', 0)	('Node4', 6.948272466659546)	5708.42344
RANDOM	6	4	DB	MLP	0.82969	0.95547	2.7440946 ('Node0', 0)	('Node4', 6.029633283615112)	5708.42344

Table J.4: Multi-node ensemble-MLP on random input by Bagging-like method-8 dataset partitions:

ORDER	Nl	Par	Partiti	Classi	Cross-Val	Test-Score	Time of P	Time of Trar	Time of Training	Ensemble-
RANDOM	8	0	D	MLP	0.82969	0.955467	2.26275 ('Node0', 0)	('Node0', 1.5941722393035	8279.656	
RANDOM	8	0	DB	MLP	0.82969	0.955467	2.97346 ('Node0', 0)	('Node0', 0.5960898399353	8279.656	
RANDOM	8	0	SB	MLP	0.82969	0.955467	0.58898 ('Node0', 0)	('Node0', 3.1591906547546	8279.656	
RANDOM	8	0	NRSB	MLP	0.82969	0.955467	0.57667 ('Node0', 0)	('Node0', 2.0272414684295	8279.656	
RANDOM	8	1	D	MLP	0.82969	0.955467	2.26275 ('Node0', 0)	('Node1', 1.1301524639129	8279.656	
RANDOM	8	1	DB	MLP	0.82969	0.955467	2.97346 ('Node0', 0)	('Node1', 6.7677462100982	8279.656	
RANDOM	8	1	SB	MLP	0.82969	0.955467	0.58898 ('Node0', 0)	('Node1', 4.5087900161743	8279.656	
RANDOM	8	1	NRSB	MLP	0.82969	0.955467	0.57667 ('Node0', 0)	('Node1', 8.6372427940368	8279.656	
RANDOM	8	2	D	MLP	0.82969	0.955467	2.26275 ('Node0', 0)	('Node2', 3.4615898132324	8279.656	
RANDOM	8	2	DB	MLP	0.82969	0.955467	2.97346 ('Node0', 0)	('Node2', 6.4030058383941	8279.656	
RANDOM	8	2	SB	MLP	0.82969	0.955467	0.58898 ('Node0', 0)	('Node2', 3.4348425865173	8279.656	
RANDOM	8	2	NRSB	MLP	0.82969	0.955467	0.57667 ('Node0', 0)	('Node2', 3.5794072151184	8279.656	
RANDOM	8	3	D	MLP	0.82969	0.955467	2.26275 ('Node0', 0)	('Node3', 3.3754463195800	8279.656	
RANDOM	8	3	DB	MLP	0.82969	0.955467	2.97346 ('Node0', 0)	('Node3', 3.5487773418426	8279.656	
RANDOM	8	3	SB	MLP	0.82969	0.955467	0.58898 ('Node0', 0)	('Node3', 3.6351270675659	8279.656	
RANDOM	8	3	NRSB	MLP	0.82969	0.955467	0.57667 ('Node0', 0)	('Node3', 4.7199556827545	8279.656	
RANDOM	8	4	D	MLP	0.82969	0.955467	2.26275 ('Node0', 0)	('Node4', 3.5255312919616	8279.656	
RANDOM	8	4	DB	MLP	0.82969	0.955467	2.97346 ('Node0', 0)	('Node4', 2.7935407161712	8279.656	
RANDOM	8	4	SB	MLP	0.82969	0.955467	0.58898 ('Node0', 0)	('Node4', 2.6297540664672	8279.656	
RANDOM	8	4	NRSB	MLP	0.82969	0.955467	0.57667 ('Node0', 0)	('Node4', 4.1186203956604	8279.656	
RANDOM	8	5	D	MLP	0.82969	0.955467	2.26275 ('Node0', 0)	('Node5', 4.6793901920318	8279.656	
RANDOM	8	5	DB	MLP	0.82969	0.955467	2.97346 ('Node0', 0)	('Node5', 4.0455284118652	8279.656	
RANDOM	8	5	SB	MLP	0.82969	0.955467	0.58898 ('Node0', 0)	('Node5', 11.991963863372	8279.656	
RANDOM	8	5	NRSB	MLP	0.82969	0.955467	0.57667 ('Node0', 0)	('Node5', 4.3554017543792	8279.656	
RANDOM	8	6	D	MLP	0.82969	0.955467	2.26275 ('Node0', 0)	('Node6', 2.4738111495971	8279.656	
RANDOM	8	6	DB	MLP	0.82969	0.955467	2.97346 ('Node0', 0)	('Node6', 9.1460583209991	8279.656	
RANDOM	8	6	SB	MLP	0.82969	0.955467	0.58898 ('Node0', 0)	('Node6', 4.5181188583374	8279.656	
RANDOM	8	6	NRSB	MLP	0.82969	0.955467	0.57667 ('Node0', 0)	('Node6', 1.9176461696624	8279.656	
RANDOM	8	7	D	MLP	0.82969	0.955467	2.26275 ('Node6', 5.3('Node7',	5.8291897773742	8279.656	
RANDOM	8	7	DB	MLP	0.82969	0.955467	2.97346 ('Node6', 5.3('Node7',	3.1146378517150	8279.656	
RANDOM	8	7	SB	MLP	0.82969	0.955467	0.58898 ('Node6', 5.3('Node7',	9.0259761810302	8279.656	
RANDOM	8	7	NRSB	MLP	0.82969	0.955467	0.57667 ('Node6', 5.3('Node7',	4.1822018623352	8279.656	

Table J.5: Multi-node ensemble-MLP on random input by Bagging-like method-10 dataset partitions:

ORDER	NO Par	Partiti	Classi	Cross-Val-	Test-Score	Time of Partiti	Time of Tran:	Time of Training	Ensemble-
RANDOM	10	0	D	MLP	0.829688	0.955467246	2.560717583 ('Node0', 0)	('Node0', 1.0018336	10083.26
RANDOM	10	0	DB	MLP	0.829688	0.955467246	2.950899363 ('Node0', 0)	('Node0', 1.7233045	10083.26
RANDOM	10	0	SB	MLP	0.829688	0.955467246	0.820222616 ('Node0', 0)	('Node0', 1.0325791	10083.26
RANDOM	10	0	NRSB	MLP	0.829688	0.955467246	0.812052965 ('Node0', 0)	('Node0', 0.8630368	10083.26
RANDOM	10	1	D	MLP	0.829688	0.955467246	2.560717583 ('Node0', 0)	('Node1', 0.6656236	10083.26
RANDOM	10	1	DB	MLP	0.829688	0.955467246	2.950899363 ('Node0', 0)	('Node1', 2.0221908	10083.26
RANDOM	10	1	SB	MLP	0.829688	0.955467246	0.820222616 ('Node0', 0)	('Node1', 2.1763992	10083.26
RANDOM	10	1	NRSB	MLP	0.829688	0.955467246	0.812052965 ('Node0', 0)	('Node1', 2.3905766	10083.26
RANDOM	10	2	D	MLP	0.829688	0.955467246	2.560717583 ('Node0', 0)	('Node2', 1.6372430	10083.26
RANDOM	10	2	DB	MLP	0.829688	0.955467246	2.950899363 ('Node0', 0)	('Node2', 5.8643944	10083.26
RANDOM	10	2	SB	MLP	0.829688	0.955467246	0.820222616 ('Node0', 0)	('Node2', 5.5286519	10083.26
RANDOM	10	2	NRSB	MLP	0.829688	0.955467246	0.812052965 ('Node0', 0)	('Node2', 2.2787077	10083.26
RANDOM	10	3	D	MLP	0.829688	0.955467246	2.560717583 ('Node0', 0)	('Node3', 8.5118451	10083.26
RANDOM	10	3	DB	MLP	0.829688	0.955467246	2.950899363 ('Node0', 0)	('Node3', 5.9629311	10083.26
RANDOM	10	3	SB	MLP	0.829688	0.955467246	0.820222616 ('Node0', 0)	('Node3', 5.1251468	10083.26
RANDOM	10	3	NRSB	MLP	0.829688	0.955467246	0.812052965 ('Node0', 0)	('Node3', 2.9890360	10083.26
RANDOM	10	4	D	MLP	0.829688	0.955467246	2.560717583 ('Node0', 0)	('Node4', 3.2317349	10083.26
RANDOM	10	4	DB	MLP	0.829688	0.955467246	2.950899363 ('Node0', 0)	('Node4', 4.4335861	10083.26
RANDOM	10	4	SB	MLP	0.829688	0.955467246	0.820222616 ('Node0', 0)	('Node4', 10.333437	10083.26
RANDOM	10	4	NRSB	MLP	0.829688	0.955467246	0.812052965 ('Node0', 0)	('Node4', 4.5913050	10083.26
RANDOM	10	5	D	MLP	0.829688	0.955467246	2.560717583 ('Node0', 0)	('Node5', 10.703117	10083.26
RANDOM	10	5	DB	MLP	0.829688	0.955467246	2.950899363 ('Node0', 0)	('Node5', 0.5796527	10083.26
RANDOM	10	5	SB	MLP	0.829688	0.955467246	0.820222616 ('Node0', 0)	('Node5', 8.4324722	10083.26
RANDOM	10	5	NRSB	MLP	0.829688	0.955467246	0.812052965 ('Node0', 0)	('Node5', 4.4462616	10083.26
RANDOM	10	6	D	MLP	0.829688	0.955467246	2.560717583 ('Node0', 0)	('Node6', 1.6056003	10083.26
RANDOM	10	6	DB	MLP	0.829688	0.955467246	2.950899363 ('Node0', 0)	('Node6', 3.8291213	10083.26
RANDOM	10	6	SB	MLP	0.829688	0.955467246	0.820222616 ('Node0', 0)	('Node6', 3.1513106	10083.26
RANDOM	10	6	NRSB	MLP	0.829688	0.955467246	0.812052965 ('Node0', 0)	('Node6', 2.3550515	10083.26
RANDOM	10	7	D	MLP	0.829688	0.955467246	2.560717583 ('Node0', 0)	('Node7', 2.1878378	10083.26
RANDOM	10	7	DB	MLP	0.829688	0.955467246	2.950899363 ('Node0', 0)	('Node7', 9.8276822	10083.26
RANDOM	10	7	SB	MLP	0.829688	0.955467246	0.820222616 ('Node0', 0)	('Node7', 1.7528812	10083.26
RANDOM	10	7	NRSB	MLP	0.829688	0.955467246	0.812052965 ('Node0', 0)	('Node7', 3.8887782	10083.26
RANDOM	10	8	D	MLP	0.829688	0.955467246	2.560717583 ('Node0', 0)	('Node8', 2.1880710	10083.26
RANDOM	10	8	DB	MLP	0.829688	0.955467246	2.950899363 ('Node0', 0)	('Node8', 5.3935475	10083.26
RANDOM	10	8	SB	MLP	0.829688	0.955467246	0.820222616 ('Node0', 0)	('Node8', 11.903022	10083.26
RANDOM	10	8	NRSB	MLP	0.829688	0.955467246	0.812052965 ('Node0', 0)	('Node8', 4.7135775	10083.26
RANDOM	10	9	D	MLP	0.829688	0.955467246	2.560717583 ('Node7', 5.9('Node9', 5.9('Node9', 11.665583	10083.26	
RANDOM	10	9	DB	MLP	0.829688	0.955467246	2.950899363 ('Node7', 5.9('Node9', 5.9('Node9', 1.9536871	10083.26	
RANDOM	10	9	SB	MLP	0.829688	0.955467246	0.820222616 ('Node7', 5.9('Node9', 5.9('Node9', 3.2058639	10083.26	
RANDOM	10	9	NRSB	MLP	0.829688	0.955467246	0.812052965 ('Node7', 5.9('Node9', 5.9('Node9', 2.3365669	10083.26	

Table J.6: Multi-node ensemble-MLP on random input by Bagging-like method-12 dataset partitions:

ORDEI	NODE	Part	Partitio	Classifit	Cross-Val-!	Test-Score	Time of Pa	Time of Trans	Time of Training	Ensemble-
RANDI	12	0	D	MLP	0.829688	0.955467	3.890158	('Node0', 0)	('Node0', 2.6370270	11709.46
RANDI	12	0	DB	MLP	0.829688	0.955467	4.403519	('Node0', 0)	('Node0', 1.3273177	11709.46
RANDI	12	0	SB	MLP	0.829688	0.955467	0.837627	('Node0', 0)	('Node0', 0.9136846	11709.46
RANDI	12	0	NRSB	MLP	0.829688	0.955467	1.156866	('Node0', 0)	('Node0', 1.1390390	11709.46
RANDI	12	1	D	MLP	0.829688	0.955467	3.890158	('Node0', 0)	('Node1', 4.9650335	11709.46
RANDI	12	1	DB	MLP	0.829688	0.955467	4.403519	('Node0', 0)	('Node1', 4.4876430	11709.46
RANDI	12	1	SB	MLP	0.829688	0.955467	0.837627	('Node0', 0)	('Node1', 12.752262	11709.46
RANDI	12	1	NRSB	MLP	0.829688	0.955467	1.156866	('Node0', 0)	('Node1', 9.6697261	11709.46
RANDI	12	2	D	MLP	0.829688	0.955467	3.890158	('Node0', 0)	('Node2', 2.6619265	11709.46
RANDI	12	2	DB	MLP	0.829688	0.955467	4.403519	('Node0', 0)	('Node2', 9.6812012	11709.46
RANDI	12	2	SB	MLP	0.829688	0.955467	0.837627	('Node0', 0)	('Node2', 2.5964581	11709.46
RANDI	12	2	NRSB	MLP	0.829688	0.955467	1.156866	('Node0', 0)	('Node2', 3.4135105	11709.46
RANDI	12	3	D	MLP	0.829688	0.955467	3.890158	('Node0', 0)	('Node3', 13.154320	11709.46
RANDI	12	3	DB	MLP	0.829688	0.955467	4.403519	('Node0', 0)	('Node3', 7.4917912	11709.46
RANDI	12	3	SB	MLP	0.829688	0.955467	0.837627	('Node0', 0)	('Node3', 3.2438330	11709.46
RANDI	12	3	NRSB	MLP	0.829688	0.955467	1.156866	('Node0', 0)	('Node3', 4.8717164	11709.46
RANDI	12	4	D	MLP	0.829688	0.955467	3.890158	('Node0', 0)	('Node4', 2.6938998	11709.46
RANDI	12	4	DB	MLP	0.829688	0.955467	4.403519	('Node0', 0)	('Node4', 9.2159466	11709.46
RANDI	12	4	SB	MLP	0.829688	0.955467	0.837627	('Node0', 0)	('Node4', 6.3810095	11709.46
RANDI	12	4	NRSB	MLP	0.829688	0.955467	1.156866	('Node0', 0)	('Node4', 3.6128857	11709.46
RANDI	12	5	D	MLP	0.829688	0.955467	3.890158	('Node0', 0)	('Node5', 2.3795359	11709.46
RANDI	12	5	DB	MLP	0.829688	0.955467	4.403519	('Node0', 0)	('Node5', 3.3796424	11709.46
RANDI	12	5	SB	MLP	0.829688	0.955467	0.837627	('Node0', 0)	('Node5', 7.3237602	11709.46
RANDI	12	5	NRSB	MLP	0.829688	0.955467	1.156866	('Node0', 0)	('Node5', 9.5763037	11709.46
RANDI	12	6	D	MLP	0.829688	0.955467	3.890158	('Node0', 0)	('Node6', 6.9890644	11709.46
RANDI	12	6	DB	MLP	0.829688	0.955467	4.403519	('Node0', 0)	('Node6', 4.2851009	11709.46
RANDI	12	6	SB	MLP	0.829688	0.955467	0.837627	('Node0', 0)	('Node6', 6.2046265	11709.46
RANDI	12	6	NRSB	MLP	0.829688	0.955467	1.156866	('Node0', 0)	('Node6', 8.6979887	11709.46
RANDI	12	7	D	MLP	0.829688	0.955467	3.890158	('Node0', 0)	('Node7', 2.3174395	11709.46
RANDI	12	7	DB	MLP	0.829688	0.955467	4.403519	('Node0', 0)	('Node7', 8.9908702	11709.46
RANDI	12	7	SB	MLP	0.829688	0.955467	0.837627	('Node0', 0)	('Node7', 10.409826	11709.46
RANDI	12	7	NRSB	MLP	0.829688	0.955467	1.156866	('Node0', 0)	('Node7', 2.5763995	11709.46
RANDI	12	8	D	MLP	0.829688	0.955467	3.890158	('Node0', 0)	('Node8', 3.6308524	11709.46
RANDI	12	8	DB	MLP	0.829688	0.955467	4.403519	('Node0', 0)	('Node8', 2.4792826	11709.46
RANDI	12	8	SB	MLP	0.829688	0.955467	0.837627	('Node0', 0)	('Node8', 9.1237764	11709.46
RANDI	12	8	NRSB	MLP	0.829688	0.955467	1.156866	('Node0', 0)	('Node8', 3.3221747	11709.46
RANDI	12	9	D	MLP	0.829688	0.955467	3.890158	('Node0', 0)	('Node9', 3.6233942	11709.46
RANDI	12	9	DB	MLP	0.829688	0.955467	4.403519	('Node0', 0)	('Node9', 1.5142960	11709.46
RANDI	12	9	SB	MLP	0.829688	0.955467	0.837627	('Node0', 0)	('Node9', 9.7609095	11709.46
RANDI	12	9	NRSB	MLP	0.829688	0.955467	1.156866	('Node0', 0)	('Node9', 9.8287725	11709.46
RANDI	12	10	D	MLP	0.829688	0.955467	3.890158	('Node0', 0)	('Node10', 4.762289	11709.46
RANDI	12	10	DB	MLP	0.829688	0.955467	4.403519	('Node0', 0)	('Node10', 7.483974	11709.46
RANDI	12	10	SB	MLP	0.829688	0.955467	0.837627	('Node0', 0)	('Node10', 2.169569	11709.46
RANDI	12	10	NRSB	MLP	0.829688	0.955467	1.156866	('Node0', 0)	('Node10', 10.92497	11709.46
RANDI	12	11	D	MLP	0.829688	0.955467	3.890158	('Node8', 6.25	('Node11', 2.078769	11709.46
RANDI	12	11	DB	MLP	0.829688	0.955467	4.403519	('Node8', 6.25	('Node11', 1.356108	11709.46
RANDI	12	11	SB	MLP	0.829688	0.955467	0.837627	('Node8', 6.25	('Node11', 11.43080	11709.46
RANDI	12	11	NRSB	MLP	0.829688	0.955467	1.156866	('Node8', 6.25	('Node11', 8.807789	11709.46

## Appendix K: Experimental results of DELS System on HIGGS

Table K.1: Training time for multi-node on sorted LADEL-[2:12] dataset partitions:

NODE #	PART #	Training time-CART	Training time-MLP
2	0	207.5243757	4.410811901
2	1	204.1257854	4.642113447
4	0	49.1583724	0.364773989
4	1	48.09182978	0.183308601
4	2	49.64368272	0.18478775
4	3	48.76984859	0.593172169
6	0	24.20863676	0.213551998
6	1	22.4028132	0.192713737
6	2	24.37811208	0.220369339
6	3	24.65529346	0.135885477
6	4	22.50022268	0.319504023
6	5	25.81299663	0.545813236
8	0	14.80749965	0.479813347
8	1	14.77436113	0.215327024
8	2	15.66281271	0.165551424
8	3	16.26888108	0.121687412
8	4	16.7793715	0.205135584
8	5	16.61184478	0.20334506
8	6	17.26438093	0.366477251
8	7	18.18044829	0.348156691
10	0	10.05238175	0.035876513
10	1	10.01142478	0.161104679
10	2	9.666127205	0.173743725
10	3	10.19817948	0.175217152
10	4	9.781116962	0.171903372
10	5	9.507634163	0.225968838
10	6	9.525583267	0.394062996
10	7	10.03168988	0.436469746
10	8	10.24550438	0.405379944
10	9	11.77484488	0.393129797
12	0	8.569123745	0.113939047
12	1	8.693429708	0.227532387
12	2	8.95452714	0.184660196
12	3	7.505750179	0.209269524
12	4	8.463288546	0.267534494
12	5	7.852916718	0.201194286
12	6	8.692966461	0.157743216
12	7	8.900878668	0.203767061
12	8	7.143722773	0.423821459
12	9	8.914180279	0.461218805
12	10	7.010122299	0.400116167
12	11	7.478554964	0.244750738



Table K.2: Training time for single-node on sorted LADEL-[2:12] parts:

NODE #	PART #	Training Time-CART	Training Time-MLP
2	0	368.0136001	8.07707929611
2	1	970.722302	10.23761057854
4	0	36.88137269	0.30437040329
4	1	36.30187917	0.19760179520
4	2	174.765486	0.19239258766
4	3	171.3531091	0.66925148964
6	0	14.69121647	0.16747665405
6	1	14.73460817	0.13596582413
6	2	14.72076845	0.13284564018
6	3	15.35086513	0.13331699371
6	4	73.97567487	0.55787944794
6	5	75.85161018	0.51566553116
8	0	10.64567995	0.18248271942
8	1	8.559438467	0.17937278748
8	2	8.2943995	0.18115735054
8	3	10.69076514	0.15203356743
8	4	10.93174195	0.19345068932
8	5	10.85930777	0.14757585526
8	6	43.62651348	0.38914155960
8	7	46.47767973	0.48854189873
10	0	5.748681307	0.37717294693
10	1	7.654337406	0.18286013603
10	2	7.887929678	0.17950129509
10	3	8.000450373	0.13058304787
10	4	7.701223373	0.17162775993
10	5	7.940872192	0.19384765625
10	6	8.090419054	0.17492222786
10	7	8.063848019	0.43742585182
10	8	27.62767053	0.33964848518
10	9	29.19114542	0.38032364845
12	0	4.066499472	0.10958361626
12	1	6.098992348	0.12338876724
12	2	6.16697526	0.15580511093
12	3	5.383090496	0.15455269814
12	4	5.984343052	0.14924144745
12	5	5.957971096	0.17402577400
12	6	6.268441439	0.15457987785
12	7	6.728176355	0.15197563171
12	8	20.10471988	0.46822776794
12	9	20.44731212	0.42445044518
12	10	20.67884898	0.32329983711
12	11	21.39685726	0.43457781792

## Appendix L: Experimental results of DELS system on HIGGS

Table L.1: Scoring time for single-node on sorted LADEL-[2:12] parts:

NODE #	Scoring Time-CART	Scoring Time-MLP
2	250.2270966	140.074163
4	515.9956222	380.5397527
6	809.6825156	645.9810624
8	1027.764865	681.3745151
10	1007.854003	628.9259033
12	958.8259308	991.0405636

Table L.2: Scoring time for multi-node on sorted LADEL-[2:12] parts:

NODE #	Scoring Time-CART	Scoring Time-MLP
2	182.9336224	220.3684473
4	391.2762494	354.730495
6	557.8512392	538.9778781
8	755.0983987	836.9345078
10	899.5738959	1022.924756
12	1062.528551	1117.677441

## Appendix M: Experimental results of DELS system on KDD Cup 99

Table M.1: Comparison in MLP and CART for LADEL :

NODE #	Classifier	SRV Cross-Val-Score	SRV Test-Score
2	MLP	0.853202909	0.268667613
4	MLP	0.219282428	0.929996416
6	MLP	0.853514463	0.194401995
8	MLP	0.93022953	0.93138082
10	MLP	0.929814416	0.589724741
12	MLP	0.852754759	0.930086886
2	CART	0.930196829	0.590309968
4	CART	0.925985781	0.583278876
6	CART	0.930075073	0.931987161
8	CART	0.280884555	0.930511535
10	CART	0.930754784	0.192626914
12	CART	0.853274294	0.582389684