Big Data Research 15 (2019) 1–11

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

Label-Aware Distributed Ensemble Learning: A Simplified Distributed Classifier Training Model for Big Data

Shadi Khalifa [a,*], Patrick Martin [a], Rebecca Young [b]

[a] School of Computing, Queen's University, Kingston, ON, Canada
[b] IBM, Markham, ON, Canada

ARTICLE INFO

Article history:
Received 27 October 2017
Received in revised form 24 October 2018
Accepted 12 November 2018
Available online 16 November 2018

Keywords:
Big Data
Analytics
Distributed
Machine learning
Classification

ABSTRACT

Label-Aware Distributed Ensemble Learning (LADEL) is a programming model and an associated implementation for distributing any classifier training to handle Big Data. It only requires users to specify the training data source, the classification algorithm and the desired parallelization level. First, a distributed stratified sampling algorithm is proposed to generate stratified samples from large, pre-partitioned datasets in a shared-nothing architecture. It executes in a single pass over the data and minimizes inter-machine communication. Second, the specified classification algorithm training is parallelized and executed on any number of heterogeneous machines. Finally, the trained classifiers are aggregated to produce the final classifier.

Data miners can use LADEL to run any classification algorithm on any distributed framework, without any experience in parallel and distributed systems. The proposed LADEL model can be implemented on any distributed framework (Drill, Spark, Hadoop, etc.) to speed up the development of its data mining capabilities. It is also generic and can be used to distribute the training of any classification algorithm of any sequential single-node data mining library (Weka, R, scikit-learn, etc.). Distributed frameworks can implement LADEL to distribute the execution of existing data mining libraries without rewriting the algorithms to run in parallel.

As a proof-of-concept, the LADEL model is implemented on Apache Drill to distribute the training execution of Weka's classification algorithms. Our empirical studies show that LADEL classifiers have similar and sometimes even better accuracy to the single-node classifiers and they have a significantly faster training and scoring times.

1. Introduction

Data mining is the process of discovering hidden patterns in data and using these patterns to predict the likelihood of future events. Several problems can be addressed using data mining like:

- Classification: Predict the category (discrete) of a new data point.
- Regression: Predict the value (continuous) of a new data point.
- Clustering: Split data points into categories.
- Association Rules: Find relationships between attributes.

In this work, we focus on the Classification problem and ways of making it Big Data ready. Classification is a supervised learning approach consisting of two phases: (1) *Training:* a classifier is built using historical labeled data (i.e. data with known category) and (2) *Scoring:* the trained classifier is used to predict the category of new data points (i.e. with unknown category). With the large volume of Big Data, classifier training time and memory requirements are a real challenge.

Scalable distributed data mining libraries like Apache Mahout [1], Cloudera Oryx [2], Oxdata H_2O [3], MLlib [4,5] and Deeplearning4j [6] implement distributed versions of the classification algorithms to run on Hadoop [7] and Spark [8]. Distributing classifier training significantly reduces the training time and enables digesting of Big Data. However, the approach used by scalable libraries requires rewriting the classification algorithms to execute in parallel. The rewriting process is complex, time-consuming and the quality of the modified algorithm depends entirely on the contributors' expertise. Thus, scalable libraries fail to support as many algorithms as sequential single-node libraries like R [9], Weka [10], scikit-learn [11] and RapidMiner [12]. Rewriting the algorithms makes scalable libraries hard to maintain and extend as it does not provide a unified model for distributing the algorithms, rather it uses a custom distributed implementation for each one [13].

* Corresponding author.
E-mail address: khalifa@cs.queensu.ca (S. Khalifa).

https://doi.org/10.1016/j.bdr.2018.11.001
2214-5796/

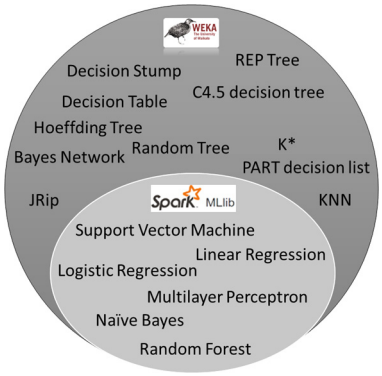


Fig. 1. Classification algorithms supported by Weka & Spark MLlib.

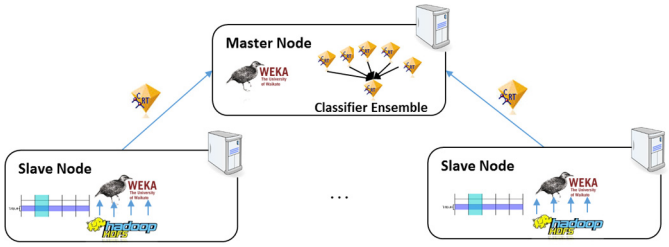


Fig. 2. Distributed classifier Ensemble training (Diamond icons represent classifiers).

Given the existing data mining libraries, data miners are left with two choices. First, they may sacrifice prediction accuracy for a faster training execution. To elaborate, users can leverage a distributed data mining library and benefit from the faster distributed training execution. However, this limits users to a small number of classification algorithms that might not provide the best accuracy since there is no one algorithm that outperforms all others for all situations [14]. Second, they may sacrifice the training execution time for a better prediction accuracy. They can use a sequential single-node data mining library and utilize the larger number of supported classification algorithms to achieve the highest accuracy but be limited to the single node processing capabilities. Fig. 1 illustrates the limited number of classification algorithms supported by scalable distributed libraries like Mahout and MLlib compared to sequential single-node libraries like Weka.

To get out of this dilemma, several papers were published on extending single-node data mining libraries to support MapReduce [15]. DistributedWekaBase [16], DistributedWekaHadoop [17] and DistributedWekaSpark [13,18] extend Weka to access the Hadoop Distributed File System (HDFS) [19], Hadoop and Spark, respectively. Another solution is RHadoop [20] which allows running R code on Hadoop and gives access to HDFS.

The approach used in the published work on distributing sequential single-node algorithms operates as follows. First, the training data is partitioned into several sequential splits distributed across the cluster nodes. Second, each Slave node runs the training algorithm on the local data split(s) to train an intermediate classifier. Finally, a vote-based ensemble is created of the trained intermediate classifiers at the Master node as illustrated in Fig. 2. In the scoring phase, the vote-based ensemble works by having each classifier in the ensemble predict the category of the new data point, then the category with the majority vote is selected as the ensemble's final prediction.

Distributing sequential single-node algorithms gives them the ability to scale to digest Big Data. It also provides data miners parallel versions of the richer set of classification algorithms supported by the single-node libraries. Unfortunately, the published work in this area suffers from a major pitfall. As far as we know, no other work evaluates the effects of their algorithm distribution approach on the classifier's prediction accuracy nor proposes any measures for maintaining the prediction accuracy after distributing the classifier's training.

In this paper, we make the following contributions. First, we evaluate the prediction accuracy of published work on distributing single-node classification algorithms. Second, we propose the Distributed Stratified Sampling (DSS) algorithm for generating stratified samples from large, pre-partitioned datasets in a shared-nothing architecture without recombining the partitions. It executes in a single pass over the data and minimizes inter-machine communication. The DSS algorithm ensures that each ensemble classifier is trained on records of all class labels. Finally, we propose the unified Label-Aware Distributed Ensemble Learning (LADEL) model for distributing the training of any sequential single-node classification algorithm on any distributed framework. LADEL eliminates the need to rewrite classification algorithms using a distributed framework. It also maintains the classifier's prediction accuracy after distributing its training.

As a proof-of-concept, we use the proposed LADEL model to implement the distributed Weka-on-Drill (codename QDrill) that extends the Apache Drill distributed framework [21] with data mining capabilities. We choose Apache Drill as it provides end-users with an easy-to-use SQL interface which requires minimum learning compared to Scala or Java used in Spark and Hadoop, respectively. We choose Weka [10] because it is a well-established widely used data mining library with an extensive set of classification algorithms.

It should be noted that the LADEL model can distribute the training of any single-node library on any distributed framework (i.e. LADEL can implement Weka-on-Spark, Weka-on-Hadoop, R-on-Spark, etc.). The distributed training time depends on the underlying distributed framework that is used. This means that a LADEL Weka-on-Spark implementation results in faster training compared to a LADEL Weka-on-Drill implementation, simply because Spark is faster than Drill. On the other hand, a LADEL Weka-on-Spark implementation provides more algorithms to choose from compared to the natively distributed Spark MLlib while maintaining similar processing speeds.

As far as we know, our work is the first to consider the effects of distributing classifier training on the prediction accuracy and the first to design measures to prevent distributed classifier training from affecting the classifier's prediction accuracy.

The rest of the paper is organized as follows: Section 2 introduces the distributed classifier training approach and highlights the key pitfalls in the related work. Following that, the Label-Aware Distributed Ensemble Learning (LADEL) model is presented in Section 3. Evaluation follows in Section 4. Finally, section 5 has the conclusions and future work.

2. Distributed classifier training: benefits and pitfalls

The main issue with many sequential single-node classification algorithms is that their training cannot be distributed, unless the algorithm is rewritten, because they need the entire dataset loaded in-memory to train the classifier. Distributed Classifier Ensembles offer a solution to this problem. In this section, we first introduce the original single-node Classifier Ensemble approach. Second, we illustrate the modifications done to the single-node Classifier Ensemble approach to allow distributed training. Finally, we highlight the pitfalls caused by these modifications and show how they affect the overall Ensemble prediction accuracy.

2.1. Single-node Classifier Ensembles

The Classifier Ensemble method trains several classifiers on the training dataset and then groups all trained classifiers into a more

powerful classifier. Classifier Ensembles allow classifiers in the Ensemble to mitigate the faults of the individual classifiers. An Ensemble allows taking the majority vote of the individual classifier results, thus eliminating the risk of picking a bad classifier. In recent years, there has been a growing attention aimed at building Classifier Ensembles of better performance with the same or better accuracy [22].

Previous research was conducted to exploit the local behavior of different algorithms to enhance the accuracy of the overall Ensemble [14]. Now, it is established that an Ensemble of a *large enough* set of *diverse unstable well-trained* classifiers of *more than random accuracy* has the same or better prediction accuracy than a single classifier [23–26]. The reasons behind this are:

- A *large enough* set of classifiers where a classifier's mistakes get corrected by the other classifiers in the Ensemble [14].
- A *Diverse* set of classifiers means that the Ensemble classifiers should make uncorrelated errors with respect to one another. That is, each classifier in the Ensemble misclassifies a different set of records [27]. Unfortunately, the problem of how to measure classifier diversity is still an open research topic. However, several approaches can be used for classifier diversification [22,26,28]:
  - Different *training set records*. Each classifier is trained on a different bootstrap sample of the training data records. This approach is convenient in the cases of a shortage or excess of learning examples. In case of a shortage of records, different joint random subsets can be generated from the data. In case of an excess amount of data that cannot fit in memory, disjoint subsets that fit in memory can be generated. The most popular techniques are Bootstrap Aggregating (Bagging) [29] and Boosting [30].
  - Different *training set attributes* (*columns*/*features*). Each classifier is trained on the entire set of training records but on a sub-set of the columns [31,32].
  - Different *labels*. Each classifier is designed to classify only a sub-set of the problem labels. For example, a multi-class classification problem can be decomposed into a set of binary classification problems [33].
  - Different *algorithm parameters*. Each classifier is initialized with a different set of values for its parameters and thus have a different local optimum [22].
  - Different *algorithms*. Each classifier is built using a different algorithm that is good at handling a sub-set of the problem patterns. This approach takes advantage of the different biases of each algorithm [34].
- *Unstable* means that a classifier's accuracy depends on the training dataset. Changing the training dataset results in a different model of a different accuracy. Unstable classifiers are Multilayer Perceptron, Decision Trees and Linear Regression [35].
- *Well-trained* means that the training dataset needs to be large enough to ensure the classifier has seen all different patterns.
- *More than random accuracy* is required for the classifiers to increase the likelihood of making the correct decision according to the Condorcet Jury Theorem [36].

2.2. Distributed Classifier Ensembles

The Classifier Ensemble approach trains multiple classifiers independently before joining them to create the final classifier. This parallel nature of Classifier Ensembles makes it well-suited for our problem of distributing sequential single-node classification algorithms across many computing nodes. However, the original Bagging algorithm [29] used to create Classifier Ensembles cannot be directly implemented for our problem.

The original Bagging algorithm is a sequential single-node algorithm. It runs on a single node and requires having all training data on a single node. It then works by having each of the Ensemble classifiers trained on a bootstrap sample of the training dataset of the same number of records as the original full training set. For example, given a training dataset of 1k records occupying 1TB, Bagging creates 5 training datasets to train 5 classifiers in the Ensemble where each training dataset is of size 1k records and uses 1TB of disk space. With Big Data, training using a single multi-gigabyte dataset is impractical as it does not fit in memory, so using Bagging to create multiple datasets of sizes equal to the original dataset, one for each classifier, is therefore impractical.

The Bagging-like approach [27] addresses this challenge by training the Ensemble classifiers on disjoint partitions of the training dataset that can fit in memory, where these partitions, if 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 [37]. Training the Ensemble classifiers on partitions of the original dataset provides a diverse set of classifiers which improves the Ensemble accuracy. The Bagging-like approach is less complex and faster than the original Bagging approach. Empirical studies [27] also show that Bagging-like Ensembles have prediction accuracy similar to that of Bagging Ensembles [29] and can exceed the prediction accuracy of sequential single-node classifiers trained on the entire training dataset.

Published work on distributing sequential single-node algorithms like DistributedWekaHadoop [17], DistributedWekaSpark [13,18] and RHadoop [20] implement the Bagging-like approach to realize distributed training of their classifier Ensembles. Training data is uploaded to the HDFS causing it to get sequentially split into 64MB disjoint partitions which then are distributed across the cluster nodes. Slave nodes then run the sequential single-node training algorithm on the local data split(s) to train an intermediate classifier. Finally, a vote-based Ensemble is created of the trained intermediate classifiers at the Master node.

2.3. Distributed ensembles pitfalls

Empirical studies [27] show that the Bagging-like disjoint partitioning approach for splitting the training dataset among the Ensemble classifiers makes each classifier biased towards its own training dataset. This creates two pitfalls in the published work on distributing sequential single-node classification algorithms.

The first pitfall is when the training dataset is ordered by the class label attribute such that records belonging to one class label come together as illustrated in Fig. 3a compared to the unordered dataset in Fig. 3b. While having a dataset ordered by the class label attribute might not be very common, a more common scenario that causes the same problem is having a continuous chunk of many records belonging mostly to the same class label. With this kind of dataset, when HDFS partitions the dataset sequentially into 64MB splits it can cause splits to mostly contain records belonging to only one class. Using these splits to train classifiers results in classifiers that always predict the same class since each classi-

Att_1	...	Att_n	label
			C0
			C0
			C0
			...
			C1
			C1

a) Ordered training dataset

Att_1	...	Att_n	label
			C0
			C1
			C0
			...
			C0
			C1

b) Unordered training dataset

Fig. 3. Training datasets: a) Ordered. b) Unordered.

fier is trained on only one class which renders the entire Ensemble useless.

The second pitfall occurs when the training dataset is highly skewed with a small number of records belonging to a minority class. In this scenario, some Ensemble classifiers will not have any records of the minority class in their training dataset, causing them to always fail in recognizing records belonging to this minority class.

The two pitfalls can be avoided if the disjoint training partitions are made to represent the original training dataset. In other words, the two pitfalls can be avoided by making sure that each partition contains records belonging to all class labels including minority classes. By achieving this, classifiers become well-trained and able to recognize records belonging to all class labels. Each Ensemble classifier is still biased towards its own training dataset, but an individual classifier's bias becomes an advantage when the classifiers are combined in an Ensemble where a protocol of knowledge sharing is established.

3. The Label-Aware Distributed Ensemble Learning (LADEL) model

The Label-Aware Distributed Ensemble Learning (LADEL) model is an extension to the Bagging-Like Ensemble model introduced in the previous section. It aims to avoid the above pitfalls of Distributed Ensembles by ensuring that the distributed classifier prediction accuracy remains similar to that of the sequential single-node classifier.

The LADEL model provides a unified model for distributing all sequential single-node classification algorithms without rewriting them, rather than having a custom distributed implementation for each algorithm as in Spark MLlib. Having a unified model significantly reduces the development time for creating distributed classifiers and supports more algorithms.

Data miners can use the LADEL model to distribute the training of any sequential single-node classification algorithm on any distributed framework (Drill, Spark, Hadoop, etc.), without any experience in parallel and distributed systems. This represents an advantage over solutions like TensorFlow [38] which have a limited set of algorithms and require users to manually specify the cluster details, which can be tedious in large clusters.

3.1. Execution model

The execution process takes a training data source, a name of a classification algorithm along with its arguments and a desired parallelization level, and produces a trained classifier. The following process illustrated in Fig. 4 happens behind the scenes:

- First, the training data is divided into a set of partitions whose class distribution matches that of the complete dataset using the proposed Distributed Stratified Sampling (DSS) approach. That is, the percentage of records belonging to each class label is equal across all partitions and is equal to that of the original training dataset.
- Second, a sequential single-node classification algorithm is executed on each data partition in parallel to produce intermediate classifiers. The number of intermediate classifiers is determined by the parallelization level specified by the user. This brings the classifier training (i.e. processing) to where data exists.
- Finally, the intermediate classifiers are moved to a single node where they are aggregated using the Voting technique. The aggregated model is then returned to the user to use for scoring. The distributed learning approach allows handling Big Data that is too large to fit in the memory of a single node while maintaining the prediction accuracy.

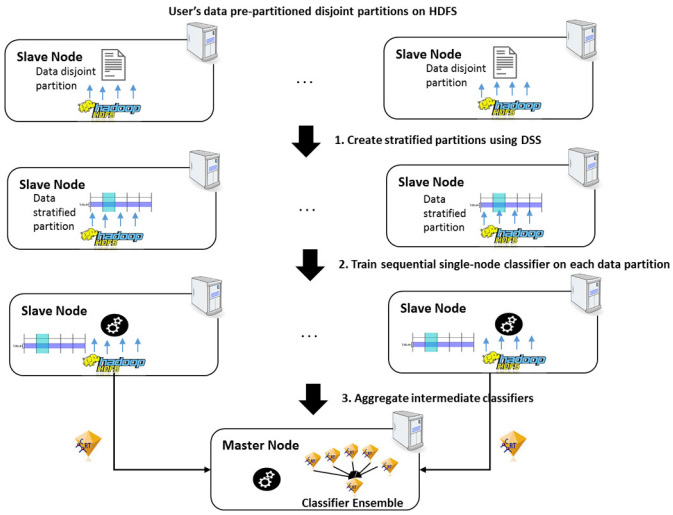


Fig. 4. LADEL execution overview.

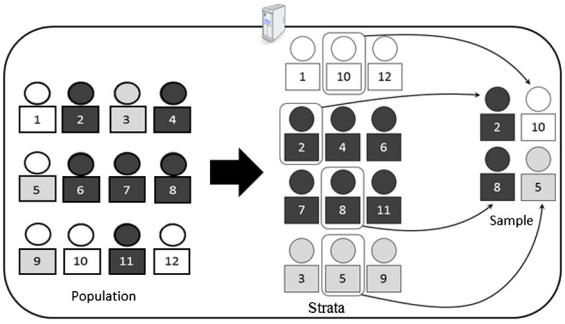


Fig. 5. Sequential single-node stratified sampling.

3.2. Distributed Stratified Sampling (DSS)

The Disjoint Partitioning (DP) approach used in the literature for creating distributed classifier Ensembles works by having each training record exist in only one partition where a partition can fit in memory. DP has a major shortcoming when dealing with ordered and highly skewed datasets [27]. The DP partitions do not include records of all class labels. This results in poorly trained Ensemble classifiers. Utilizing Stratified Sampling instead of the Simple Random Sampling used in DP can solve this problem by ensuring that each partition has records of all class labels.

The original sequential single-node Stratified Sampling illustrated in Fig. 5 works by having all data on a single machine. The algorithm goes through the entire dataset (Population) twice. First, it divides the dataset into several separate strata (categories) based on the class label attribute. Second, each stratum is sampled as an independent sub-population, out of which individual records are randomly selected for the different partitions. Stratified Sampling ensures that each partition has records of all class labels and that each partition follows the same class label distribution as the original dataset. Thus, all Ensemble classifiers are trained on all class labels, allowing the Ensemble to maintain a good accuracy even with ordered and highly skewed datasets with minority classes.

Big Data, with its large volumes, renders sequential single-node Stratified Sampling impractical. Big Data does not fit in the memory of a single node and is usually randomly partitioned when uploaded to a distributed filesystem like HDFS. These partitions likely do not include records of all class labels so building clas-

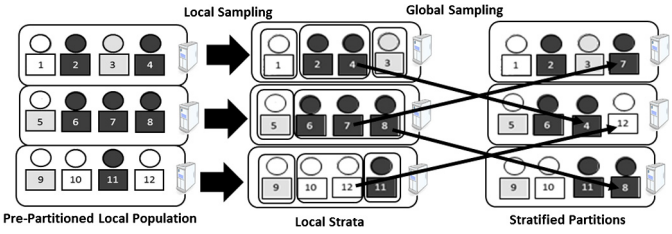


Fig. 6. Distributed stratified sampling.

sifiers from these partitions results in a poorly trained Ensemble classifiers.

Recombining the distributed Big Data for repartitioning using sequential single-node Stratified Sampling is expensive in terms of disk and network usage and processing time. The proposed Distributed Stratified Sampling (DSS) addresses these issues.

DSS creates stratified partitions in parallel from the distributed filesystem partitions without recombining them. The DSS algorithm, illustrated in Fig. 6, runs independently on each machine in the cluster in a shared-nothing architecture. It executes in a single pass over the data and minimizes inter-machine communication. DSS consists of two phases, namely: Local and Global sampling, which works as follows.

Given P distributed filesystem partitions, the Local phase runs in parallel on each machine separately. First, the DSS creates strata based on the records' class labels on each machine. Second, it runs a round-robin selector of size P on each stratum on each machine. Third, for each round-robin cycle, one record is kept for the current machine and one record is sent to each of the $(P-1)$ other machines in cluster.

In the Global phase, each machine combines its remaining records with the records sent to it from the other machines to create the training partition for its local intermediate classifier. This new partition contains records of all class labels required for producing a well-trained classifier.

The DSS algorithm run-time is the summation of the time taken to conduct the Local sampling in parallel to create the local strata and the time to do the Global sampling in parallel to exchange data records. The DSS run-time is given by:

$$DSS\ runtime = \frac{R}{P} + \frac{\sum_P \sum_C \frac{R_{c,p}}{P}(P-1)}{P} \quad (1)$$

where R is the total number of records, P is the number of partitions, C is the number of class labels and $R_{c,p}$ is the number of records belonging to class label C in partition P. DSS is scalable and its run-time decreases as the number of partitions increases. On the other hand, recombining the distributed Big Data for repartitioning using sequential single-node Stratified Sampling has a run-time of $2 \times R$ which is orders of magnitude longer than the DSS run-time.

Approaches such as the Efficient Sample Generator [39] provide a general, single-pass algorithm for generating samples from large, block-partitioned datasets stored in a distributed filesystem. However, they do not support stratified sampling nor disjoint partitioning required for building well-trained distributed Ensembles.

3.3. Implementation

The LADEL model can distribute the training of any single-node library on any distributed framework. As a proof-of-concept, we use the proposed LADEL model to implement a cloud-ready distributed Weka-on-Drill solution (codename QDrill). The Weka-on-Drill extends Apache Drill [21], a distributed SQL query engine for non-relational storage, with data mining capabilities. We choose Apache Drill as it provides end-users with an easy-to-use SQL interface which requires minimum learning compared to Scala or Java used in Spark and Hadoop, respectively. We choose Weka [10] because it is a well-established widely used data mining library with an extensive set of classification algorithms. The implementation is available as the enhanced QDrill stand-alone, free and open-source solution [40,41] and as an IBM SPSS Modeler plugin with a GUI on the IBM SPSS Predictive Analytics Gallery [42].

3.3.1. Architecture

Apache Drill is powerful in terms of accessing and joining data from heterogeneous sources using its Storage Adaptor, which is usually a cumbersome task when done in data mining libraries. On the other hand, Drill does not have any data mining capabilities. Developing data mining algorithms for Drill is time consuming and so would likely be limited to a handful of algorithms, nothing compared to those available in the well-established data mining libraries. Our Weka-on-Drill implementation solves these issues by using Drill to load and join data from heterogeneous sources and using the pre-existing classification algorithms of the well-established Weka data mining library to train and score the classifiers but in a distributed fashion.

The full system architecture is illustrated in Fig. 7, showing the unmodified components of Drill (UI and JDBC/ODBC connection), the modified (Distributed Query Planner, Query Execution Engine and the Storage Adapter) and the newly added (Analytics Adapter).

We add the Analytics Adaptor to Drill to optimize and provide access to the various data mining libraries. The Analytics Adaptor works with Analytics Plugins for various data mining libraries, which transform the data loaded by Drill to a data structure understandable by the data mining libraries. This way, algorithms from more than one library can be used together, leaving it to the Analytics Adaptor to resolve the inter-library data format conversion.

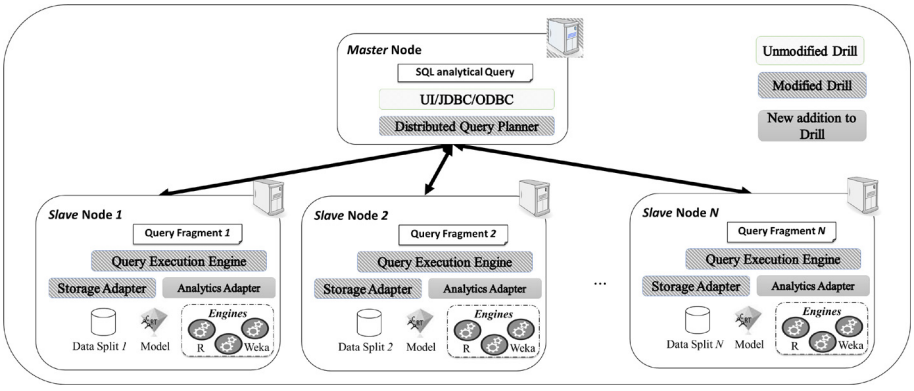


Fig. 7. LADEL Weka-on-Drill architecture.

In addition, the plugins invoke the data mining library APIs to train and score classifiers.

In this proof-of-concept, we implement the Weka Analytics plugin. The plugin provides access to WEKA's classification algorithms and converts the data loaded by Drill to the ARFF format required by WEKA.

To train a new classifier using Weka-on-Drill, users submit the training data source, the classification algorithm along with its arguments and the desired parallelization level to the Master node. First, the Master node runs the DSS algorithm on the Slave nodes to create stratified partitions. Second, the Master node uses the Weka Analytics plugin on the Slave nodes to run Weka's sequential single-node classification algorithms on the stratified partitions on each Slave node. Finally, intermediate classifiers trained on the Slave nodes are aggregated at the Master node to produce the final classifier.

3.3.2. Classifier interoperability

Our Weka-on-Drill implementation adds a *Model* Storage plugin to Drill's Storage Adapter to write and read data mining models (i.e. trained classifiers). The Model Storage plugin can store and load Weka's models from any data store supported by Drill. This allows classifier interoperability where a classifier trained using Weka-on-Drill can be used for scoring on Weka-on-Spark or Weka-on-Hadoop or even on the sequential single-node Weka.

3.3.3. Distributed Analytics Query Language (DAQL) interface

Extracting patterns from data is the main aim of data miners. They do not wish to learn new complex programming paradigms and languages. Our LADEL Weka-on-Drill implementation (QDrill) provides a high-level SQL-based interface which is compiled by the run-time system to the low-level primitives required for the parallel execution. Using a SQL-based interface to access data mining functionalities make the distribution of the data mining tasks transparent to the user.

We modify Drill's Distributed Query Planner and Query Execution Engine to introduce a number of new Keywords for running distributed Analytics operations. The name "Distributed Analytics Query Language (DAQL)" is given to the modified SQL syntax to reflect its Analytics capabilities. DAQL allows invoking classification algorithms from within Drill's standard SQL query statements. Interested readers can see our other work [41] for more details on the DAQL language.

The statements in Fig. 8 train a Weka classifier in a distributed fashion using the proposed LADEL model. The first statement changes the storage location to a writable location. The second statement tells Drill's Storage Adaptor to use the introduced Model Storage plugin to save the classifier after training. The third statement consists of three nested DAQL statement:

- The *inner statement* invokes the DSS partitioning algorithm using our `qdm_DSS` function with arguments: the number of partitions `<num parts>`, the record's attributes `<columns>` and the record's label `<label_column>`, respectively. This statement fetches the training data from any Drill-supported data store using the `FROM` clause. The `FROM` clause can also have a join between two heterogeneous data sources. The `WHERE` clause specifies conditions on the records to fetch.
- The *middle statement* uses our `qdm_ensemble_weka` function to train a classifier on each data partition using the `GROUP BY` clause to send records belonging to different partitions to the different Slave Nodes. Our `qdm_ensemble_weka` function defines the classifier algorithm, sets its arguments, specifies the data columns to use for training and specifies the label column, respectively.

```
1> USE dfs.tmp;
2> ALTER SESSION SET 'store.format '='model';
3> TRAIN MODEL <Model Name> AS
   SELECT qdm_ensemble_weka (mymodel)
   FROM (SELECT qdm_ensemble_weka ('<Algorithm>','<Args>',
              data.columns, data.label_column) as mymodel
         FROM (SELECT columns, qdm_DSS (<Num Parts>,
                     columns, label_column) as partition
               FROM '<Data Source>'
               WHERE <Conditions>
         ) as data
               GROUP BY data.partition);
```

Fig. 8. DAQL statement for training a classifier.

- The *outer statement* uses our `qdm_ensemble_weka` function to aggregate the classifiers trained on the Slave Nodes into an Ensemble. Finally, the statement uses our `TRAIN MODEL` clause to save the Ensemble under `<model name>`.

3.3.4. IBM SPSS modeler graphical interface

Our Weka-on-Drill inherits the Drill's JDBC/ODBC interface. This allows connecting any program or Business Intelligence tool to our solution and instantaneously adding analytical capabilities to that program or extending its capabilities, as within the IBM SPSS Modeler. The Modeler plugin [43] is written in the R language and has an easy-to-use GUI interface illustrated in Fig. 9. The plugin uses Drill's JDBC/ODBC interface to send data and configuration to our Weka-on-Drill solution. Our solution then uses the LADEL model to distribute the training of the specified classification algorithm and returns the trained model back to Modeler. The trained classifier can be used in Modeler like any other classifier trained within Modeler. This shows the capability of the proposed solution to work in conjunction with other analytics solutions.

4. Evaluation

In this section, we evaluate three approaches for training a classifier. We do not evaluate the training time of the different approaches since it is already well established that some distributed frameworks are faster than others and because the LADEL model can be ported to any distributed framework (e.g. Spark, Hadoop, etc.). Instead, we focus on evaluating the prediction accuracy of classifiers trained using the different approaches.

The first approach uses our LADEL proof-of-concept implementation on the *Apache Drill 1.2* distributed framework and the *weka-dev-3.7.13* sequential single-node data mining library. Code is available for free and is open-source [40]. The second approach uses an implementation of *weka-dev-3.7.13* vote-based distributed Ensemble learning representing the published work on distributing sequential single-node classification algorithms [13,17,20]. The third approach uses the original sequential single-node *weka-dev-3.7.13* classification algorithm (no Ensemble) running on a single machine trained on the entire training dataset. The third approach represents the baseline for our evaluation.

Algorithms: The *Hoeffding Tree* and the *Multilayer Perceptron* (Neural Networks and Deep Learning) algorithms are used in the evaluations here. Both algorithms are frequently used to address various classification problems. They both require loading the entire training dataset in-memory to do the training, thus both require a complete rewrite to run in parallel. Other algorithms like C4.5 decision tree, KNN, Random Tree, Decision Table, Decision Stump, REP Tree, PART decision list, Support Vector Machine, and Linear Regression suffer from the same distribution problem. We only include evaluations for Hoeffding Tree and the Multilayer Perceptron as they represent LADEL's worst and best cases in terms of training time, respectively and LADEL's best and worst cases in terms of accuracy, respectively, and to keep charts easy to read.

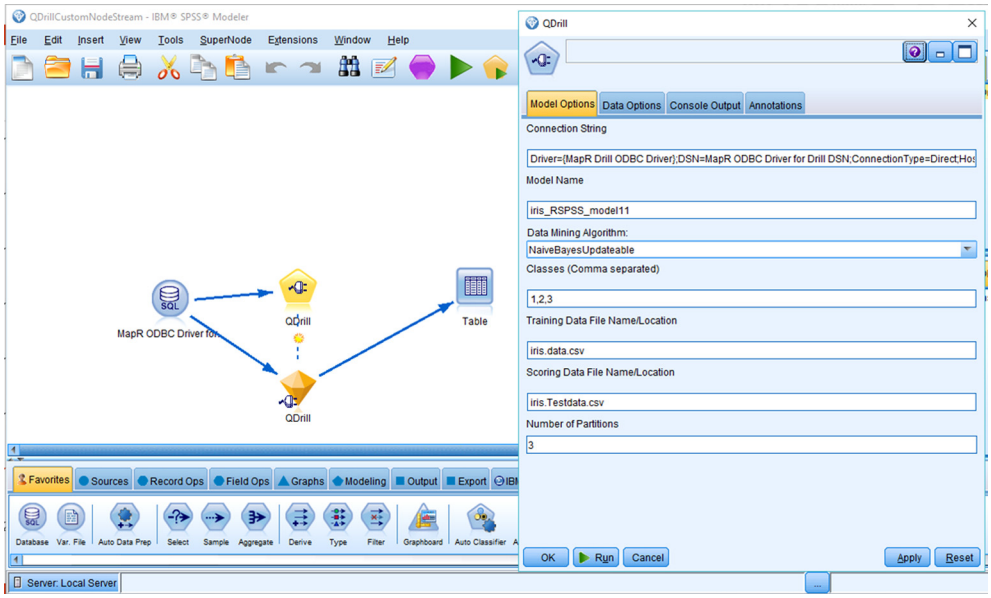


Fig. 9. IBM SPSS modeler graphical interface.

Table 1
HIGGS data sets characteristics summary.

# Training records	# Validation records	# Attributes	# Class labels	Labels distribution	Size [GB]
7,158,118	3,385,684	28	2	Balanced 1:1	4.65

Dataset: At the beginning of our experiments we ran on 4 datasets: KDD Cup 1999 [44], SUSY [45], HEPMASS ALL [46] and HIGGS [45]. The results were very similar but consumed enormous amounts of time to run on the different algorithms with different dataset configurations and so we decided to only use the HIGGS dataset for the remainder of the experiments.

Since we always compare with the single classifier performance, only the dataset characteristics matter. We tested the LADEL's effectiveness in different scenarios using datasets that are artificially generated from the HIGGS dataset. The original unordered HIGGS dataset was modified to create three extra datasets to test LADEL in a controlled environment. These are Ordered dataset, 1:2 skewed dataset, and 1:10 skewed dataset. The generated datasets are described later in each analysis.

The HIGGS real-life dataset from the UCI repository is used with the characteristics summarized in Table 1. A separate set of records, not included in the training, is used for validating the accuracy of the trained classifier. We modify the dataset, both training and validation sets, by removing records with missing values and converting nominal values to numeric so that any data mining algorithm can work on them. We do not use any other pre-processing operations, data transformation or feature selection to prevent bias in the results.

Environment setup: We use the Amazon EC2 *T2.Large* [47] Ubuntu instances with two cores (3.3 GHz each), 8 GB RAM, 8 GB Swap space and a 100 GB Solid State Disk. Our cluster for evaluating LADEL and existing vote-based Ensemble solutions uses 10 of these nodes providing up to 40 threads, 20 cores, 80 GB of RAM and 700 GB of distributed HDFS storage space that is configured to have three replicas per block. For this configuration, the HIGGS dataset is Big Data since its data structure cannot fit in a node's memory. The single machine for evaluating the original sequential single-node classification algorithm has two cores (3.3 GHz each), 24 GB RAM and 50 GB disk space which is big enough to hold the entire dataset in memory.

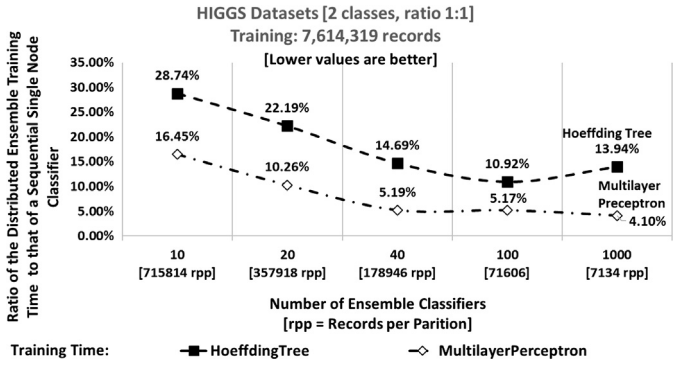


Fig. 10. Ratio of the distributed Ensemble training time to that of a sequential single-node classifier.

4.1. Distributed Ensembles evaluation

In this analysis, we investigate the appropriate size of a distributed Ensemble for producing predictions of the same, or better accuracy than a sequential single-node classifier trained on the entire training dataset, while reducing the training and scoring time. A distributed Ensemble is built by training classifiers on splits of the training data on multiple Slave nodes. The classifiers are then grouped together in a vote-based Ensemble on the Master node.

4.1.1. Training time

In terms of training time, adding more classifiers to the Ensemble leads to increasing the training parallelization which leads to a shorter training time as illustrated in Fig. 10. Our empirical studies show that using distributed Ensembles with any classification algorithm and any number of partitions consumes at most 30% the time required for training a sequential single-node classifier on the entire training dataset (HoeffdingTree: 7.2 min – MultilayerPerceptron: 14.14 hrs). It should be noted here that with some classification algorithms (e.g. Hoeffding Tree) having too many Ensemble classifiers (e.g. 1000) leads to competition for the computational resources which slightly increases the training time.

4.1.2. Scoring time

In terms of scoring time, the scoring time of an Ensemble is directly proportional to the number of its classifiers as illustrated in

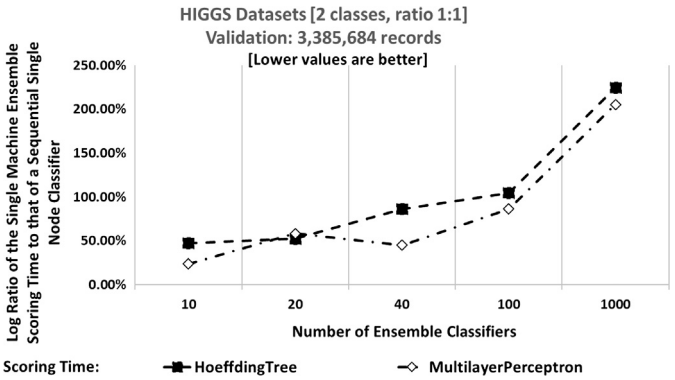


Fig. 11. Log ratio of the single machine scoring time of an Ensemble to that of a single classifier on a single machine.

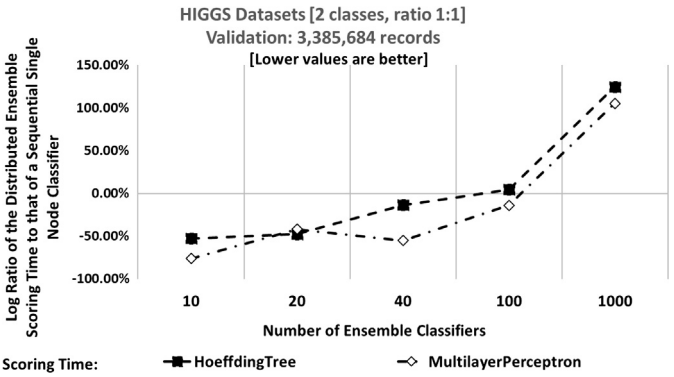


Fig. 12. Log ratio of the distributed scoring time on 10 machines of an Ensemble to that of a single classifier on a single machine.

Fig. 11. As the number of Ensemble classifiers increases, the more classifiers need to score each new data point before voting for the final classification. In Fig. 11, the trained distributed Ensemble is deployed on a single machine and used to score the validation data records. As can be seen in Fig. 11, using Ensembles significantly increases the scoring time compared to having a single classifier when running on a single machine.

To reduce the scoring time, the Ensemble is copied to 10 nodes and the validation data is split across them to parallelize the scoring operations. Distributing the scoring operation across 10 machines does reduce the scoring time and it becomes shorter than that of a single classifier running on a single machine as illustrated in Fig. 12. However, distributing the scoring operation does not help when the number of the Ensemble classifiers becomes too large (e.g. 1000). Distributed scoring on 10 machines for very large Ensembles can consume 18X the scoring time of a sequential single-node classifier running on a single machine since still each record needs to be scored against 1000 classifiers then a vote need to be made among the 1000 scores.

4.1.3. Prediction accuracy

The Ensemble error percentage E_e is calculated as $E_e = (M_e - M_s)/V$, where M_e is the number of records misclassified by the Ensemble, M_s is the number of records misclassified by the Single Classifier and V is the total number of validation records.

In terms of prediction accuracy, the LADEL Ensembles achieve accuracies significantly better (16.84% less misclassifications for Hoeffding Tree with 20 partitions) than that of a sequential single-node classifier trained on the entire training dataset as illustrated in Fig. 13. The LADEL Ensembles create a diverse set of weak classifiers that overcome each other's misclassifications, thus achieving better accuracies.

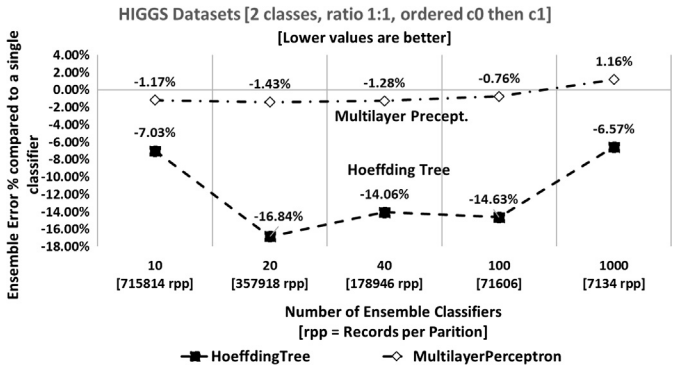


Fig. 13. Prediction error of the LADEL Ensemble compared to that of a sequential single-node classifier trained on all data.

The Ensembles' accuracy, however, decreases as the number of Ensemble classifiers increases since the number of records per training data partition decreases. For example, in the case of 1000 partitions, there are only 7,134 records per partition that is 3,567 records per class label per partition. Our interpretation is that the amount of records becomes not enough to adequately train the classifiers, which leads to the increase in the prediction error.

4.1.4. Observations

A LADEL distributed classifier Ensemble achieves accuracies that are similar or better than that of a single-node classifier trained on the entire training dataset. As per our experiments, this statement holds for all Weka's algorithms and is consistent with previous research [23,27,48].

The number of Ensemble classifiers to use depends on the number of records in the training dataset. It is better to have more classifiers with bigger datasets to speed up the training without affecting the Ensemble's accuracy. Having too many classifiers, each trained on fewer records results in a poorly trained classifier. From our observation here, we find that having from 20 to 40 classifiers results in the best accuracy for the HIGGS dataset. With 20 to 40 classifiers, both the training and scoring times are acceptable. Having more classifiers leads to reducing the training time but causes lowering the accuracy and increasing the scoring time

4.2. Prediction accuracy evaluation for LADEL versus regular vote-based Ensembles

In this analysis, the proposed LADEL Ensemble is evaluated against regular vote-based Ensembles commonly used in the literature for distributing sequential single-node classification algorithms [13,17,20]. Evaluation is done for the two pitfalls introduced earlier. First, we study the Ensembles' prediction accuracy when given a training dataset ordered by the class label attribute. Second, we study the effect of the training dataset skewness on the Ensembles' prediction accuracy.

4.2.1. Ordered training dataset evaluation

In this evaluation, the HIGGS dataset records are ordered by the class label attribute. The ordered dataset is uploaded to the HDFS, split into 64 MB splits and automatically distributed among the cluster 10 nodes.

As illustrated in Fig. 14, creating a distributed Ensemble using the proposed LADEL model results in a significantly better accuracy compared to a single classifier trained on the entire training dataset (14% less misclassifications for Hoeffding Tree). It also results in a significantly better accuracy compared to regular vote-based distributed Ensembles commonly used in the literature for distributing sequential single-node classification algorithms (17.65% less misclassifications for Multilayer Perceptron).

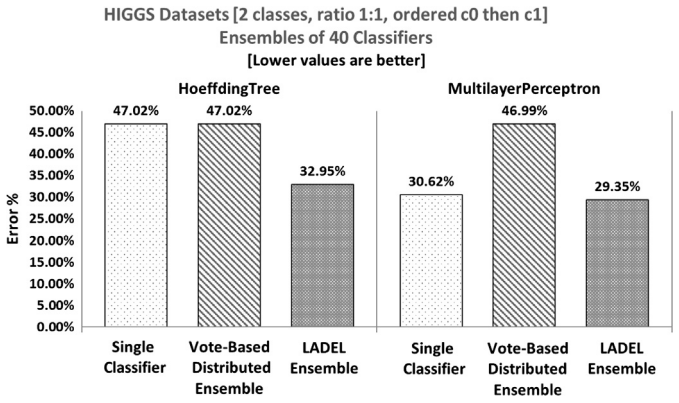


Fig. 14. Prediction error of single classifier, vote-based distributed Ensemble and LADEL Ensemble on ordered datasets.

For the Hoeffding Tree, both the single classifier and the regular vote-based distributed Ensemble misclassify almost 50% of the validation records. The LADEL Ensemble with its well-trained classifiers lower that error to 33%.

The deficiency of regular vote-based distributed Ensembles is highlighted when using the Hoeffding Tree algorithm. The Hoeffding Tree Ensemble should benefit from the diversity in its sub-optimal trees (i.e. make errors on different validation records) to produce accuracies better than that of a single classifier [27]. As shown in Fig. 14, there was no accuracy gain when using the regular vote-based distributed Ensemble because its trees are each trained on a single class label.

Using the regular vote-based distributed Ensemble approaches can even lead to significantly worse accuracies compared to a sequential single-node classifier trained on the entire training dataset which is the case for the Multilayer Perceptron (16.37% more misclassifications). The Ensemble classifiers get trained on records belonging to only one class label due to the way HDFS splits the training dataset. The lack of training records representing all class labels creates a classifier that always predicts the class label it is trained on which significantly reduces the Ensemble accuracy. This scenario is very common and the LADEL model is designed to handle this scenario by having records belonging to all class labels in all data splits.

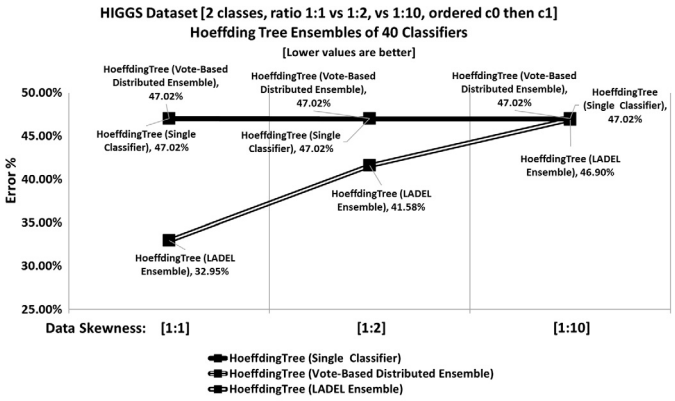
4.2.2. Skewed training dataset evaluation

In this analysis, we investigate the effect of training dataset skewness towards one of the class labels on the classifiers' prediction accuracy. Three scenarios are considered in this analysis. The first scenario uses the original balanced HIGGS dataset [1:1] which has an equal number of records belonging to each class. The second scenario [1:2] reduces the number of records belonging to class 0 to half the number of records belonging to class 1. The third scenario [1:10] further reduces the number of records belonging to class 0 to one tenth the number of records belonging to class 1. The skewed datasets are then uploaded to HDFS, split into 64 MB splits and automatically distributed among the cluster 10 nodes.

The main observation from the analysis illustrated in Fig. 15 is that as the training dataset becomes more biased towards one of the class labels, that is one of the class labels becomes less represented, the classifier accuracy decreases as it becomes more biased towards choosing the more represented class label when predicting.

The LADEL model ensures that each data split contains records belonging to all class labels no matter how little a class label is represented in the original training dataset. Thus, all LADEL Ensemble classifiers are trained on all class labels, allowing the LADEL Ensemble to maintain a good accuracy even with highly skewed

(a) Hoeffding Tree



(b) Multilayer Preceptron

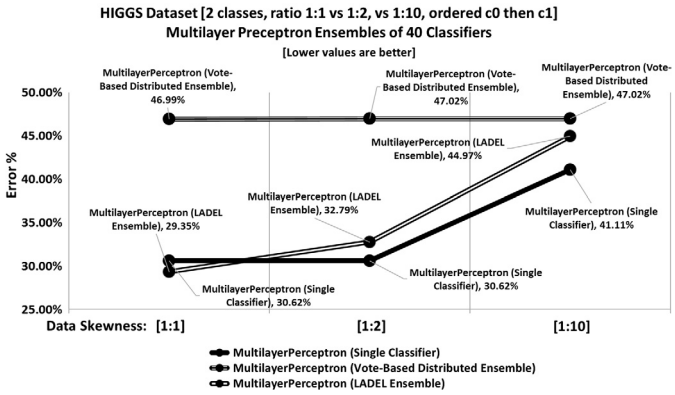


Fig. 15. Prediction error of single classifier, vote-based distributed Ensemble and LADEL Ensemble on skewed training datasets.

datasets with minority classes. That is why the LADEL error does not go above 4% of that of a single classifier (Multilayer Perceptron [1:10]) while the regular vote-based distributed Ensemble approaches reported in literature exceed 16% (Multilayer Perceptron [1:2]).

The experiment (Multilayer Perceptron [1:10]) was repeated using Simple Random Sampling (SRS) which yielded a 47.02% error rate while taking the same time as DSS. With SRS, all minority class records are misclassified because the majority of the ensemble classifiers are not trained on records belonging to the minority class. This shows that DSS leads to higher accuracy in more situations than SRS.

With skewed datasets with minority classes, it is recommended to have as few Ensemble classifiers as possible to ensure that there are enough records belonging to the minority class in each training data split to adequately train the classifiers.

4.3. Distributed stratified sampling overhead

The prediction accuracy maintained by the LADEL model comes at a price. Distributed training using LADEL includes the cost of the DSS algorithm to create stratified partitions to assure that each Ensemble classifier has records belonging to all class labels. Using DSS increases the training time compared to other vote-based distributed Ensemble learning approaches as illustrated in Fig. 16.

5. Conclusions and future work

In this article, we introduce the LADEL model for distributing the training of classifier Ensembles while avoiding the pitfalls suffered by other vote-based distributed Ensembles. The LADEL model

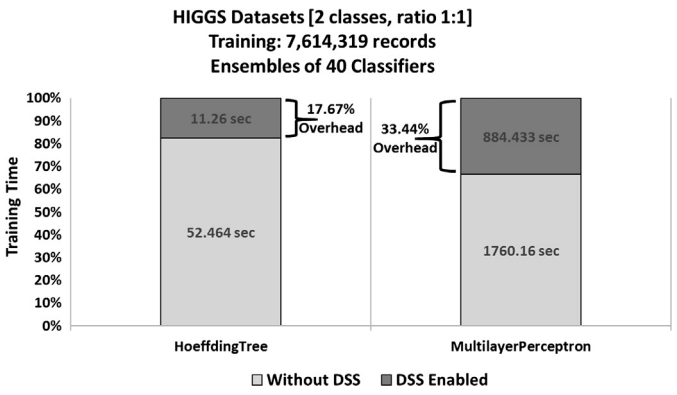


Fig. 16. Training time with and without DSS.

is easy to use, even for data miners without experience with parallel and distributed systems, since it hides the details of parallelization. It significantly reduces the development time for creating distributed classification algorithms and adds scalability to a large set of classification algorithms that are either unimplemented in distributed data mining libraries or unusable for Big Data in their sequential single-node implementation. The LADEL model relies on the proposed Distributed Stratified Sampling (DSS) algorithm to generate stratified samples from large, pre-partitioned datasets in a shared-nothing architecture. It executes in a single pass over the data and minimizes inter-machine communication.

An open-source proof-of-concept LADEL implementation is built on the Apache Drill distributed framework and the Weka sequential single-node data mining library. The proof-of-concept allows distributing the training of all Weka's classification algorithms on Apache Drill without rewriting the algorithms. The LADEL model, however, can be implemented on any distributed framework to distribute the training of any sequential single-node data mining library.

The empirical study shows that LADEL Ensembles can have classification accuracy that is similar and sometimes better than having a single classifier trained on the entire training dataset. The DSS algorithm implemented in the LADEL model guarantees having records representing all class labels in the training partitions. The empirical study also shows that other vote-based distributed Ensembles reported in literature tend to have significantly worse classification accuracies with ordered datasets compared to the single classifier due to the lack of records representing all class labels in its classifiers' training data.

For the future work, the prediction accuracy of distributed regression Ensembles and distributed clustering approaches is to be evaluated. This evaluation is then to be used to devise models for distributing regression and clustering that maintains their accuracies.

Acknowledgements

This work is supported by IBM Canada Centre for Advanced Studies (865), Ontario Research Fund (05 065) and National Science and Engineering Research Council (453459-13).

The authors would like to thank the Center for Machine Learning and Intelligent Systems, Bren School of Information and Computer Science, University of California, Irvine, for their publicly available Machine Learning Repository [http://archive.ics.uci.edu/ml] from which data sets were used in this work.

References

[1] Mahout [Online]. Available: https://mahout.apache.org/.
[2] Oryx [Online]. Available: https://github.com/cloudera/oryx.
[3] H_2O [Online]. Available: http://0xdata.com/h2o-2/.
[4] MlLib [Online]. Available: https://spark.apache.org/mllib/.
[5] E. Sparks, A. Talwalkar, V. Smith, X. Pan, J. Gonzales, T. Kraska, M. Jordan, M. Franklin, MLI: an API for distributed machine learning, in: IEEE 13th Int'l Conf. on Data Mining (ICDM), 2013.
[6] Deeplearning4j [Online]. Available: http://deeplearning4j.org/.
[7] T. White, Hadoop: The Definitive Guide, 1st ed., O'Reilly Media, Inc., 2009.
[8] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, Spark: cluster computing with working sets, in: 2nd USENIX Conf. on Hot Topics in Cloud Computing (HotCloud'10), 2010.
[9] R [Online]. Available: https://www.r-project.org/.
[10] Weka [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/.
[11] scikit-learn [Online]. Available: http://scikit-learn.org/stable/.
[12] RapidMiner [Online]. Available: https://rapidminer.com/.
[13] A.-K. Koliopoulos, P. Yiapanis, F. Tekiner, G. Nenadic, J. Keane, A parallel distributed weka framework for big data mining using spark, in: IEEE International Congress on Big Data, 2015.
[14] M. Fernández-Delgado, E. Cernadas, S. Barro, D. Amorim, Do we need hundreds of classifiers to solve real world classification problems?, J. Mach. Learn. Res. 15 (2014) 3133–3181.
[15] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, in: Symp. on Operating Systems Design Implementation (OSDI'04), 2004.
[16] DistributedWekaBase [Online]. Available: http://goo.gl/wcJrCa.
[17] DistributedWekaHadoop [Online]. Available: http://goo.gl/69lVLE.
[18] DistributedWekaSpark [Online]. Available: http://goo.gl/sWngFD.
[19] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The hadoop distributed file system, in: IEEE 26th Symp. on Mass Storage Sys. and Technologies (MSST '10), 2010.
[20] RHadoop [Online]. Available: https://goo.gl/CsZad3.
[21] Apache Drill [Online]. Available: https://drill.apache.org/.
[22] M. Wozniak, M. Graña, E. Corchado, A survey of multiple classifier systems as hybrid systems, Inf. Fusion 16 (2014) 3–17.
[23] L.K. Hansen, P. Salamon, Neural network ensembles, IEEE Trans. Pattern Anal. Mach. Intell. 12 (10) (1990) 993–1001.
[24] A. Krogh, J. Vedelsby, Neural network ensembles, cross validation, and active learning, Adv. Neural Inf. Process. Syst. 7 (1995) 231–238.
[25] D. Optiz, J. Shavlik, Generating accurate and diverse members of a neural-network ensemble, Adv. Neural Inf. Process. Syst. 8 (1996) 535–541.
[26] L. Rokach, Ensemble-based classifiers, Artif. Intell. Rev. 33 (1) (2010) 1–39.
[27] N. Chawla, T. Moore, L. Hall, K. Bowyer, P. Kegelmeyer, C. Springer, Distributed learning with bagging-like performance, Pattern Recognit. Lett. 24 (2003) 455–471.
[28] G. Brown, J. Wyatt, R. Harris, X. Yao, Diversity creation methods: a survey and categorisation, Inf. Fusion 6 (1) (2005) 5–20.
[29] L. Breiman, Bagging predictors, Mach. Learn. 24 (2) (1996) 123–140.
[30] Y. Freund, Boosting a weak learning algorithm by majority, in: The 3rd Annual Workshop on Computational Learning Theory (COLT '90), 1990.
[31] R. Bryll, R. Gutierrez-Osuna, F. Quek, Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets, Pattern Recognit. 36 (6) (2003) 1291–1302.
[32] K.M. Ting, J.R. Wells, S.C. Tan, S.W. Teng, G.I. Webb, Feature-subspace aggregating: ensembles for stable and unstable learners, Mach. Learn. 82 (2011) 375–397.
[33] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, F. Herrera, An overview of ensemble methods for binary classifiers in multi-class problems: experimental study on one-vs-one and one-vs-all schemes, Pattern Recognit. 44 (2011) 1761–17760.
[34] D.H. Wolpert, The supervised learning no-free-lunch theorems, in: 6th Online World Conf. on Soft Computing in Industrial Applications, 2001.
[35] D. Opitz, R. Maclin, Popular ensemble methods: an empirical study, J. Artif. Intell. Res. 11 (1999) 169–198.
[36] L. Shapley, B. Grofman, Optimizing group judgmental accuracy in the presence of interdependencies, Public Choice 43 (3) (1984) 329–333.
[37] C. Perlich, F. Provost, J.S. Simonoff, Tree induction vs. logistic regression: a learning-curve analysis, J. Mach. Learn. Res. 4 (2003) 211–255.
[38] TensorFlow [Online]. Available: https://www.tensorflow.org.
[39] S. Schelter, J. Soto, V. Markl, D. Burdick, B. Reinwald, A. Evfimievski, Efficient sample generation for scalable meta learning, in: IEEE 31st International Conference on Data Engineering, Seoul, 2015.
[40] QDrill [Online]. Available: https://github.com/skhalifa/QDrill.
[41] S. Khalifa, Achieving Consumable Big Data Analytics by Distributing Data Mining Algorithms, PhD thesis, Queen's University, Canada, 2017.
[42] IBM SPSS predictive analytics gallery [Online]. Available: http://ibmpredictiveanalytics.github.io/.
[43] QDrill [Online]. Available: https://github.com/IBMPredictiveAnalytics/QDrill.
[44] KDD Cup 1999 UCI Data Set [Online]. Available: http://archive.ics.uci.edu/ml/datasets/kdd+cup+1999+data, 1999.
[45] P. Baldi, P. Sadowski, D. Whiteson, Searching for exotic particles in high-energy physics with deep learning, Nat. Commun. 5 (2014) 4308.

[46] HEPMASS UCI data set [Online]. Available: https://archive.ics.uci.edu/ml/datasets/HEPMASS.
[47] Amazon EC2 instance types [Online]. Available: https://aws.amazon.com/ec2/instance-types/.
[48] R.E. Schapire, Y. Freund, P. Barlett, W.S. Lee, Boosting the margin: a new explanation for the effectiveness of voting methods, in: 14th Int'l Conf. on Machine Learning (ICML '97), 1997.