SCALABLE DATA ANALYTICS FOR ENSEMBLE LEARNING

BY SUHAS R AITHAL

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Manish Parashar

and approved by

_____

_____

_____

New Brunswick, New Jersey

May, 2013

ABSTRACT OF THE THESIS

Scalable Data Analytics for Ensemble Learning

by Suhas R Aithal

Thesis Director: Manish Parashar

Data Anlaytic techniques have enhanced human ability to solve a lot of data related problems. It has opened a window through which an analyst can devise techniques to solve a given problem by just looking at the related data alone. Such techniques may not be directly visible to the programmer. Machine learning in short is programming computers to generate such algorithm to solve a given problem using example data or past experience. Numerous models or functions can be developed which fits a distribution of data points. The magnitude and breadth of this data plays a major role in determining which model fits best. In the current world, data is growing at an alarming rate both in terms of size and the information it conveys. Analytics on large-scale data requires very high execution time with limited resources. Implementing such techniques to generate different models on a single machine becomes tedious and time consuming. Implementing the same on a distributed network is a possible solution which is highly interesting and challenging. Data Analytic Algorithms, to create these models, should be carefully chosen on the basis of reliability and data integrity and at the same time should be easily distributable. The range of algorithms should be wide enough to incorporate the same analytics on all the data with required performance. An analysis of various metrics such as measure of scalability, accuracy, execution time helps classify individual algorithm's impact on the required performance. The above

metrics are highly dependent on the data, type of algorithm, and type of platform under consideration.

This work proposes a unique system for Scalable Data Analytics and gives an analysis of its scalability. It also shows how this system implements a unique way of ensemble learning in useful time without compromising on accuracy. Finally it demonstrates scalability on two levels 1. Scalability at an individual algorithm level 2. Scalability at implementing ensemble learning for different use cases.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Current world data analytics provides strong tools to analyze highly complex data and their features. Data analysis varies from numerous machine learning techniques for classification and regression to data clustering and natural language processing. At the same time every system in the world produces enormous amount of information. This information is processed and stored in various different forms depending on the type of data analysis done on them. For example, the data representation for natural language processing is well described in the work by Ivan Shamshurin et al [22]. In this work, data has been represented as sets of features that describe the data and its internal relationship. This data can be centrally located or distributed among many machines.

Machine Learning techniques makes use of this inter dependability of data to produce models and functions which fit the predictor attributes to the response. Various statistical algorithms provide a backbone for machine learning. The works of Kuncheva, L / Whitaker, C [17] and Sollich, P / Krogh, A [25] provides an interesting observation through experimentation that an ensemble of classifiers almost always produces output better than any single classifier. They proved that generating multiple models to fit functions for a particular dataset and averaging over the same may most certainly provide better results than any single model. Generating these models is computation intensive and requires lot of memory and CPU time.

Modern world requires computation and results in real time or at least in useful time. For example predicting the ailment to be provided to a patient depending on the data which describes his/her condition has to be done in useful time without compromising the health of the patient. This prediction may involve enormous amount

of computation and calculations which might be highly time consuming. Michael J. Kearns et al. [14] in his work, provides a detailed analysis of the involved computations and complexities in machine learning algorithms providing lower bounds on certain on few sample complexities and limitations in polynomial time learnings [14]. One logical way to do this is to distribute the computation over many systems and then finally aggregate the result to find the final response. R. Bekkerman, J. Langford, M. Bilenko [4] in their work have distributed the algorithm LambdaMART and have shown promising results. Scaling the model generation not just provides an improvement in execution time with respect to the algorithm but also reduces the cost of non-required data transfer and storage. The data thus can also be distributed among nodes and compuations can be done on each node for its respective data. Only the final required result can be extracted.

This work gives a system which distributes the generation of different models, for a unique type of ensemble learning, in a cluster of machines. It also gives a detailed analysis of scaling up the system on different platforms and distributed network. This system also proposes a novel and simple way of blending the response of individual algorithms which enables easy scalability.

1.2 Problem Description

Analyzing a system requires knowledge about its representing data. The representing data associated with a system can be very complex and high on storage size. Data regression and classification which involves a lot of predictive analysis thus becomes very compute intensive and requires lots of time. Machine learning algorithms help us solve these problems statistically.

Data representing a system can be very diverse and the data points can be highly distributed. Due to this, fitting one exact model or function to represent all the data points is highly difficult and impractical. Giorgio Valentini and Francesco Masulli's research [29] in this field has revealed that an ensemble of weak learners when used together by some mechanism or function produce better results than any single learner.

This indirectly necessitates the use of multiple models and algorithms which can constitute the weak learners. But constructing all these various models will invariably require lot of execution time. Hence the problem is not just at one algorithm level but also at a level where numerous types of these algorithms have to be combined.

One way to alleviate this problem is to perform the computation on a distributed platform. The problem can be broken down at both the levels mentioned above to achieve higher performance in execution time. Questions such as - what are the features of an algorithm which makes it possible for it to be implemented on a distributed network? How to schedule multiple model generation on multiple nodes and how to aggregate the results without compromising the accuracy? - needs to be answered.

1.3 Research Overview

This work proposes a system and its evaluation which scales upto 100's of cores on distributed network of systems. Two platforms are used to demonstrate the scalability and its evaluations are recorded. Machine learning algorithms are used in many domains. This research work mainly concentrates on data from medical and social media domain. It also provides a measure of scalability on two levels individual model/algorithm, ensemble of models/algorithms.

The approach towards performing the predictive analysis is done both on a single algorithm and ensemble of algorithms level. The approach towards ensemble learning is highly motivated by research in areas of data classification/regression where the data points have irregular distribution in a multi-dimensional space where each axis is defined by attributes which describe the data. It will be shown in this work as to how ensemble learning can be accomplished in a distributed manner there by gaining a lot of advantages on the execution time. The system proposed is unique in a way that the learning of different models to form an ensemble is done dynamically by learning each prediction recursively. This will be explained in the further sections. At the same time this research also brings out the limitations and conditions for parallel implementation of certain algorithms.

Finally an anlysis of this proposed system is done on datasets from two different domains and giving their evaluation results. A note of certain features of algorithms is made which define the extent to which they can be applied in a scalable environment.

1.3.1 Contributions

The following are the contributions of this research:

- Traditionally the ensemble learning has been implemented using techniques such as feature weighted mechanism, boosting, and bagging. In this work in addition to those techiniques, a unique blend of such techniques using recursive learning is proposed.
- The unique blend of ensemble methods is done by generating the constituent models parallely in a cluster of nodes using packages which distribute work both at multiple core and node level.
- Give an execution time evaluation of this super model on two different levels - single algorithm and ensemble of algorithms
- Two use cases where this algorithm is implemented giving its importance and usefulness.

With time the complexity and the size of data representing a system is growing. To do data analytics it becomes important that algorithms be carefully analyzed on its various features. With increased size and complexity comes increased computation and execution time. For this reason, algorithms and analysis, as the one presented in this work, becomes very important as they give a lot of information on choosing the right algorithm which can be scaled as required. With new algorithms being developed, an analysis and scope for scalability can motivate scientists to incorporate features in their work which could potentially be exploited for scalability. We also expect that this work advances the use of distributed data analytics in the fields where dynamic results are required.

1.4 Thesis Overview

The organization of the thesis is as follows. Chapter 2 gives the background and related work of the thesis. It introduces the various algorithms used and also the concept of ensemble learning. Chapter 3 covers the infrastructure on which we build the algorithm. It illustrates the two frameworks Mahout and R which we have used to build our algorithm and analyze the scalability. Chapter 4 presents our algorithm and work on distributing our algorithm over a cluster of machines. It describes two procedures of ensemble learning and how we have successfully implemented them on multi-node systems. Chapter 5 will then show the analysis of implementing this algorithm on two datasets from different domains. It provides the improvement in execution time and performance of using multiple nodes over a single machine. Chapter 6 provides conclusion and the future work of our research.

Chapter 2

Background and Related Work

The main aim of this research is to provide a system for efficient data analysis, in particular prediction analysis, on variuos different data and analyse the execution time performance of all. This helps to choose the right direction and have a pre-emptive approximation of the time required for such analysis in practical scenarios. This chapter gives certain background of this research and elaborate the related work in the field of machine learning, ensemble learning and distributed data analysis.

2.1 Machine Learning

Machine learning, a branch of artificial intelligence, is about the construction and study of systems that can learn from data. The core of machine learning deals with representation and generalization. Representation of data instances and functions evaluated on these instances are part of all machine learning systems. There is a wide variety of machine learning tasks and successful applications.

Arthur Samuel defined machine learning as a "Field of study that gives computers the ability to learn without being explicitly programmed" [3]. Machine learning algorithms can have various approaches encompassing decision tree learning, association rule learning, aritificial neural networks, genetic programming, inductive logic programming, support vector machines, clustering, bayesian networks, and reinforcement learning. This thesis is concentrated mainly on data regression and classification [18] with supervised learning.

Decision tree analysis is one of the widely used techinqiues for predictive analysis which uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. Classification trees and regression trees

are two such tree models in which leaves represent class labels and branches represent conjunctions of features that lead to those class labels.

Leo Breiman et al. [5][6] illustrates in their work the importance of bagging predictors. They proved with the help of tests on real and simulated data sets using classification and regression trees and subset selection in linear regression that bagging can give substantial gains in accuracy. They propose an algorithm called as 'Random Forest' which is a combination of many decision trees giving an aggregated decison. Their appraoch is towards an ensemble of trees and the final prediction being the mode/majority of the decisons of each tree.

Andy Liaw and Matthew Wiener et al. [16] in their work provide a detailed explanation and demonstration of the random forest on statistical language R. They give individual examples explaining the various features of random forest in both classification and regression.

Jerome H. Friedman et al. [12] research on stochastic gradient boosting provides perdictive analysis which form an additive regression model by sequentially fitting a simple parameterized function to current "psuedo"-residuals by least-squares at each iteration. The residuals are the gradient of the loss function being minimized with each iteration. They also proved that inducing randomization into this process significantly improves both accuracy and execution speed.

2.2 Ensemble Learning

Sill et al. [23] build upon stacking techniques, predictions of a collection of models are given as inputs to a second-level learning algorithm, using a linear function to determine the weight each model should be given. Their approach is to create a collection of linear functions in order to avoid predetermining weights for each model used for prediction.

Street et al. [26] look to address the issue of resampling, such as Boosting and Bagging, which is slow for data mining. They separate classifiers on sequential chunks of training points and combine them into a fixed-size ensemble using a heuristic replacement strategy. This allows them to process data faster and at a larger scale than could

not be done when using resampling techniques.

Weiss et al. [30] presents the role that rare classes and rare cases, which they show are of great value in real world applications, play in data mining. They demonstrate that these rare classes and rare cases are similar since-both forms cause similar problems during data mining and benefit from the same solutions.

Dietterich [10] Gives and extensive study on some previous studies comparing ensemble methods, and present new experiments that demonstrate that Adaboost does not overfit rapidly. The paper specifically reviews Bayesian averaging,Bagging, and boosting.

2.3 Distributed Data Analytics

Panda et al. [19] have created PLANET which is a scalable distributed framework for learning tree models over large datasets. They looked to eliminate the bottleneck of tree learning algorithms to that date which required users to place all information in memory so that it could be processed. Their framework takes advantage of the MapReduce programming paradigm in order to distribute the workload across many nodes and use a DFS to distribute the dataset across all the nodes.

Chu et al. [8] develop a applicable parallel programming method that can be applied on different learning algorithms. The specifically look at algorithms that fit that Statistical Query model that can be written in a certain summation form. They adapt Google's map-reduce paradigm to show that speed up can be achieved on a variety of learning algorithms.

Provost et al. [20] summarizes, categorizes, and compares existing work on scaling up inductive algorithms. They concentrate on algorithms that build decision trees and rule sets. They highlight similarities among scaling techniques by categorizing them into three main approaches. They use the preceding analysis to suggest how to proceed when dealing with a large problem, and where to focus future research.

Gaber et al. [13] provide a comprehensive review on data stream mining. They explain the field where the top concern is to extracting knowledge structures represented

in models and patterns on data streams. They provide an outline and discuss research problems in processing these data streams. They also include some discussion on the issues that should be addressed in order to create a robust systems that can fulfill the needs for applications that create these data streams.

This work uses cooncepts of bagging, boosting and decision trees to generate various models with each having a unique set of parameters. These models are integrated in a unique way by learning the performance of each model recursively. It proposes two aprroaches which will be discussed in the further sections.

Chapter 3

Infrastructure and Frameworks supporting Scalable Predictive Data Analytics

This section gives you the details about the infrastructure and frameworks on which these system were built and analyzed. The implementation of the algorithms were done on two different platforms Apache Mahout and Open-source language R. Apache Mahout is an Apache Project consisting of implementations of distributed / scalable machine learning algorithms on the Hadoop platform. R is an open source programming language which supports statistical computing. R is also widely used among data miners for data analysis.

3.1 Apache Mahout

Apache Mahout consists of a collection of algorithms for clustering, classification, and batch based collaborative filtering and they are implemented on top of Apache Hadoop. Apache Hadoop is an open-source software framework that supports data-intensive distributed applications. It supports the running of applications on large clusters of commodity hardware and implements a computational paradigm named map/reduce. It also provides a distributed file system that stores data on the compute nodes of the cluster. It is a highly reliable and a robust framework providing automatic handling for node failures.

This research used algorithms for the classification method. The algorithm which we particularly concentrated was the Random Forest. The details about the algorithm will be provided in the subsequent chapters.

Map-Reduce

MapReduce 3.1 is a framework which implements parallelizable problems on big

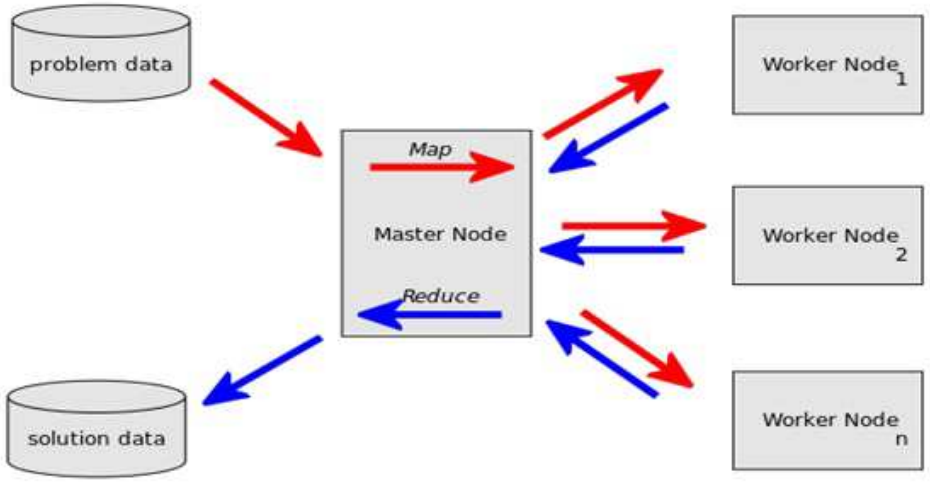

Figure 3.1: Pictorial representation of Map-Reduce Paradigm

datasets across large number of nodes which form a distributed network such as clusters, grids, clouds [9]. Computations are done on data which are either stored in a central filesystem or a database. In the MapReduce paradigm the systems of the cluster/grid can be divided into two types. They are :

- Master Node: As the name suggests this is the main node which directly interacts with the problem data and the program / algorithm to be implemented. This node is responsible to break the computation down and assign work to the worker nodes. Once the work is completed by the workers, the master node then does the reduce step where in the results are merged and the final output is obtained

- Worker Node: There are many worker node associated with each master. They receive instructions from the master node about the work they have to perform. Worker nodes are the nodes where the computation takes place. In some cases the worker nodes also perform the work of further breaking down of data and passing on its own workers.

These systems work with two types of functions. They are :

- Map function: In this function or step the master node breaks the problem or computation to many sub-problems and assigns these to its workers. The workers then do the assigned work and return the results back to the master. In certain cases (multi-level master-worker tree structure) the workers can further divide the work assigned to them and reassign them to their respective workers.

- Reduce Function: In this function or step, the master aggregates the results sent by the worker in a suitable fashion there by obtaining the final result.

The work flow of a MapReduce job is as shown in the figure 3.1. A problem along with the data from the filesystem/database is taken by the master node and broken down to multiple sub-problems. These sub-problems are then fed to the workers for its implementation. The workers do the required computation in parallel and send the result back to the master. Once the master receives the result from all its workers it then aggregates the result to form the final output.

3.2 Statistical Language R

R is an open-source programming software which is widely used among statisticians and data analysts to perform various statistical and data analysis. R provides variety of statistical tools, including linear and nonlinear modeling, classical statistical test, time-series analysis, classification, clustering and many more. R is extended to other functionalities through extensible functions and extensions. R offers a number of machine learning algorithms through the use of user submitted packages. Hence implementing algorithms without having to code the statistical details helps our cause.

R not only provides various algorithms through the packages but also supports distributed and parallel programming through a number of packages. There are packages to overcome the R's single threaded nature by spreading work across multiple CPUs or even multiple nodes in a cluster. R is also memory sensitive to its computation. Packages are also available for offloading work to multiple machines and to access data through files and databases to address R's memory barrier. The packages we used in this work are as follows:

- Snow package [27]: The package snow (an acronym for Simple Network Of Workstations) provides a high-level interface for using a workstation cluster for parallel computations in R. Snow relies on the Master / Slave model of communication in which one device or process (known as the master) controls one or more other devices or processes (known as slaves). Snow package contains a lot of functions which are parallel implementation of basic functions of R. Hence snow can be extensively used to parallelize the integral part of the codes without having to change them much. Snow supports three types of underlying message passing system. They are:
  - Raw Sockets - (does not require any additional packages)
  - Parallel Virtual Machines(PVM) (uses rpvm and PVM as additional packages)
  - Message Passing Interface(MPI) (uses Rmpi with an MPI setup back end)
- foreach package [1]: the foreach package provides a looping construct for the parallel execution of code. The parallel nature is seen when the foreach is used on multi-processor computer or a cluster of computers. The foreaqch package in general is not just used for parallel computation. The foreach package in this work is mainly used for the parallel implementation of random forest. This will be detailed in the next chapter.
- doMC package [2]: The doMC package is a parallel backend for the foreach package. It provides a mechanism needed to execute foreach loops in parallel. Using this package we register a backend of the required number of processors. This number is nothing but the number of worker processes running in parallel when the execution takes place. The foreach package must be used in conjunction with a package such as doMC in order to execute code in parallel. The doMC package acts as an interface between foreach and the multicore package [28]. The limitation of doMC is that it runs only on one machine with multiple cores. Hence to distribute the data onto multiple nodes, a package with higher abstraction such snowfall which is a wrapper around the snow package is used.

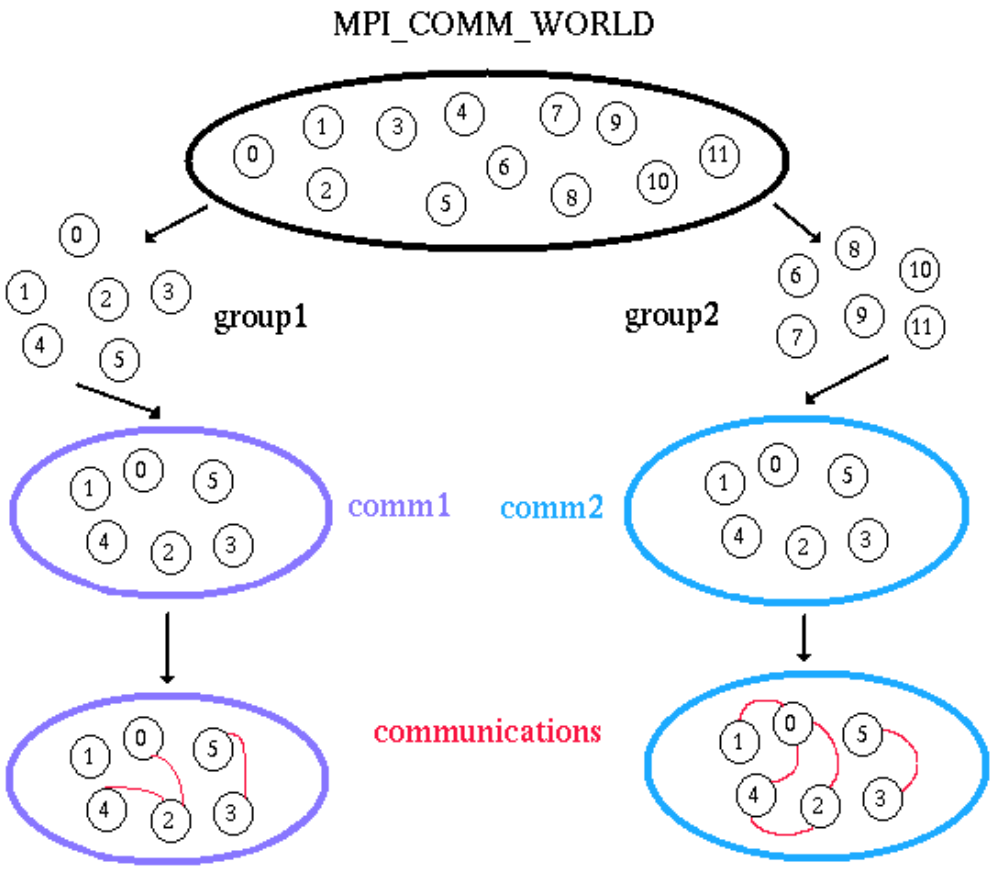

Figure 3.2: Message Passing Interface (MPI)

As written earlier there are 3 types of message passing communication methods available for using the snow package. In this research, MPI method for communication between nodes was used.

MPI[24][7] is a language-independent communication protocol which supports both point-to-point and collective communication. MPI is a message passing model or an application interface which contains semantic specifications and protocols about its behavior in any implementation. MPI is preferred by users because it is very simple to use. MPI usually work with multiple processes. Each core of a multi-core processor is assigned with a process and inter-process communications take place for two processes to interact. MPI also at any point has information about the total number of processes/ cores, information about which process assigned to its respective core. The communication between two processes is done through exchanging messages and storing these messages in each processes queue. The process then fetches the information from

its queue and follows the required steps 3.2. This work uses the package snowfall [15] which is a wrapper package around snow. Snowfall gives inbuilt functions which helps us create a cluster with user defined number of CPUS. It also gives option to define the backend communication method between the three described earlier. MPI was chosen as the communication paradigm and made use of the package Rmpi [32] to support it in R. Hence the package and in-built functions take care of the message passing and synchronization. No explicit coding for passing the messages from one process to another was done. Hence this makes it a lot easier to build the algorithm without worrying too much about the internal details.

Chapter 4

Scalable Data Analysis : Approach

This chapter describes the various machine learning algorithms considered with brief explanation and its features. It then describes two algorithms which are designed to implement ensemble learning. It further describes the features of these individual algorithm which makes them suitable for parallel implementation. A description of the approach for parallelizing the two ensemble learning algorithms and packages required are then listed. These algorithms and approach will be tested with two use case datasets and the results will be discussed in next chapter.

4.1 Algorithms in scope

This section will briefly explain the various algorithms we have used for the machine learning process. It will also describe their features and how they work.

4.1.1 Random Forest

Random Forest is a machine learning algorithm developed on the concepts of decision trees. It is an ensemble of many decision trees (forest) and its output is the mode of the class output of these individual trees. The method combines Breiman's "bagging" idea and the random selection of features, introduced independently by Ho and Amit and Geman in order to construct a collection of decision trees with controlled variation.

A decision tree takes several input variables to predict a target variable. Each node of the decision tree corresponds to one of the input variables. The children of these nodes are a logical outcome of the value check at the parent node. Each leaf represents the value of the target variable given the values of the input variables represented by the path from the root to the leaf. A decision tree looks like the one shown in fig 4.1.

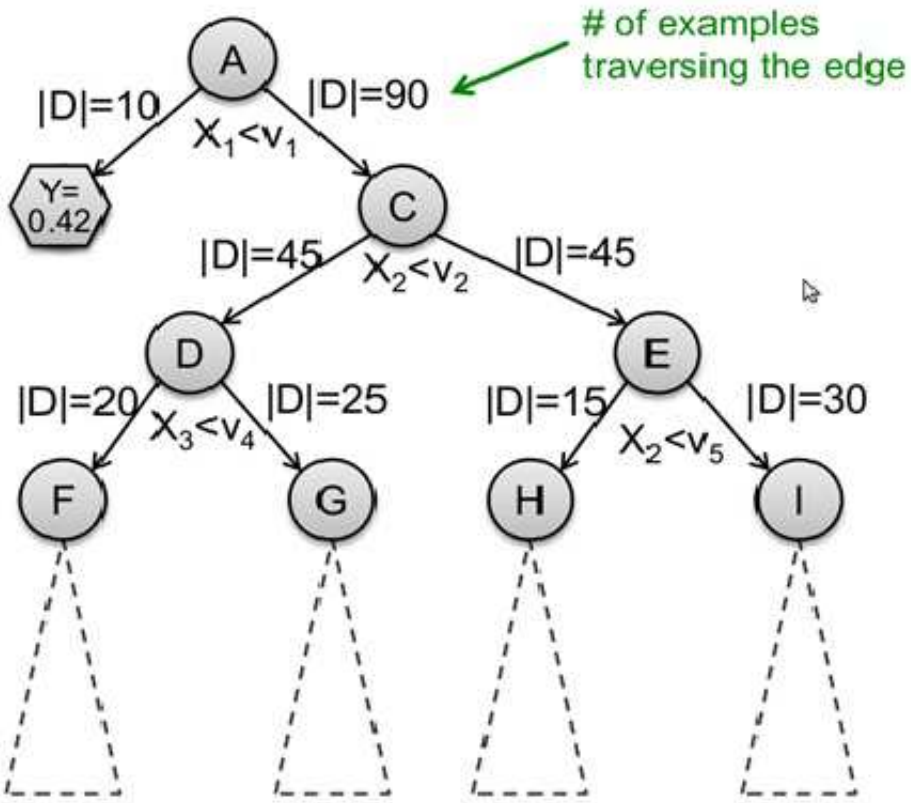

Figure 4.1: Example of a Decision Tree

At each node a random variable X from the set of variables is chosen and is compared with the split value v and depending on the outcome, the data point goes either to left child or right child. The leaf nodes are the final class associated with the path the data-point traverses. The tree is learned using the recursive partitioning process and follows the top-down induction of decision trees. If there is a dataset with N records and M descriptors/variables, a brief algorithm description of random forest can be described as follows:

- For each tree choose m out of M such that m¡M randomly to determine the decision at the node of a tree.
- Calculate the best split for these m variables and build the tree.
- Take a boostrap sample from N records and use the others for extimating the error of the tree by predicting classes. Hence for a tree all the N records are needed to build it.
- Each tree is fully grown and not pruned.

For the prediction of a test sample, the sample is trickled down the tree and is assigned the label of the sample in the terminal node it ends up. This is done over all the trees and the mode vote of all the tress is then taken as the final prediction. Since an ensemble of different models is considered, new models can be created by varying some of the features of the random forest. These features include the number of trees in the forest, the value m which is nothing but the number of observations considered per node. The package in R for random forest [**?**] provides other additional options which can be varied to obtain different models of random forest.

4.1.2 Gradient Boosting Machines

Gradient boosting is a boosting approach that resamples the data set several times to generate results that form a weighted average of the resampled data set. This work mainly deals with the gradient tree boosting where a series of decision trees together form a single model. A tree in the series is fit to the residual of the predictions from

earlier trees. This residual is represented in the form of a loss function. The loss function for a squared error loss is the difference between the actual and the predicted target value in a specific target interval. Each time data is used to grow a tree, accuracy is computed. The successive samples are then adjusted to compensate for the inaccuracies. This approach is also called stochastic gradient boosting. There is no assumptions made on the distribution of the data and in the given interval of input, the model depends only on the ranks of the values. Boosting is less prone to overfit of the data as compared to decision trees and the prediction accuracy only gets better and improves the fit. The CRAN package 'gbm' [21] offers excellent implementation in the R language for the gradient boosting algorithm. It also provides the various algorithm parameters which can be varied to create different models. Few of the GBM parameters which were used to create a variety of models are as follows:

- Number of trees: the total number of trees to generate. This is equivalent to the number of iterations and the number of base learners which make up the final additive expansion.

- Interaction.depth: The maximum depth of variable interactions. 1 implies an additive model, 2 implies a model with up to 2-way interactions, etc.

- N.minobsinnode: minimum number of observations in the trees terminal nodes. Regularization is implemented using this parameter and it helps to reduce variance in predictions at leaves.

- Shrinkage: a shrinkage parameter applied to each tree in the expansion. Also known as the learning rate or step-size reduction. This parameter plays an important role in the computation time for learning and querying. Its found empirically that lower values of shrinkage yields improvement in model's generalization and requires more number of iterations.

4.1.3 Generalized Linear Model

GLMNET is an algorithm developed for the estimation of generalized linear models. The models include linear regression, two-class logistic regression, and multinomial regression problems while the penalties include ‘1 (the lasso), ‘2 (ridge regression) and mixtures of the two (the elastic net). The algorithms use cyclical coordinate descent, computed along a regularization path. The algorithm provided is an extension of Friedman's work [11] in developing fast algorithms for fitting generalized linear models with elastic-net penalties. In particular, glmnet models include regression, two-class logistic regression, and multinomial regression problems. Lasso procedures are frequently used in domains with very large datasets, such as genomics and web analysis. Since the focus of our work was scalability and big data analytics, this algorithm was chosen for its stability to work on big data and also the speed. Many models of the glmnet are created by changing one of the main parameters, alpha - the elasticnet mixing parameter. This parameter helps to strike the right balance between the lasso (alpha=1) and ridge regression (alpha=0) analysis. Ridge regression is known to shrink the coefficients of correlated predictors towards each other, allowing them to borrow strength from each other. The ridge penalty is ideal if there are many predictors and all have non-zero coefficients. Lasso, on the other hand, is somewhat indifferent to correlated predictors, and will tend to pick one and ignore the rest. Hence the lasso penalty favors a distribution of predictors with many close to zero and a small subset which has large non-zero values.

4.2 Model of Models - A Constructive Approach

Motivated from the work of Sill et al. [23], Weiss et al. [30] in ensemble learning, different techniques of implementing the same were designed. Few of the ways included the Feature-weighted Linear stacking, bootstrap aggregating, Bayesian model averaging, etc. But something easier to implement and at a higher abstraction level was required. Further deliberation over new ways of creating models gave two interesting facts.

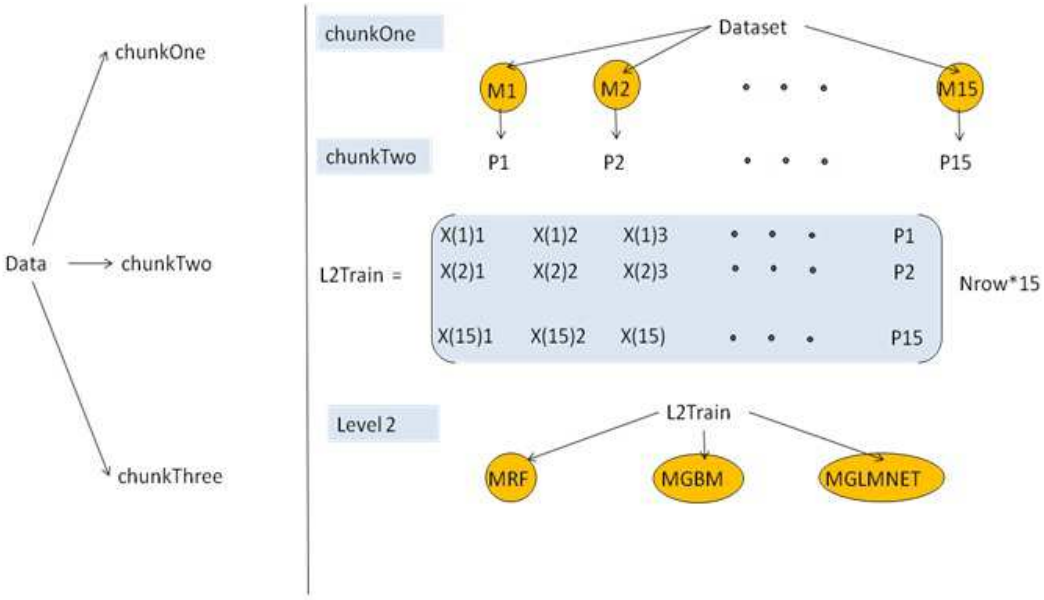

Figure 4.2: Ensemble Method One : Averaging by Learning

- Simple averaging, with equal weights assigned to each of the models in the ensemble, is a very common methodology adopted to do ensemble learning. The predicted values are given equal weights and are averaged to get the mean root mean square error.

- A model which is generated to fit a particular type of distribution of data-points tends to fit a different set of data-points in a similar fashion. Hence learning the inaccuracies while generating a model for a distribution could further be used to fit a new function with respect to the inaccuracies.

Working on the above two ideas, following two techniques of ensemble learning were devised.

4.2.1 Averaging by Learning

This method makes use of the simple averaging of the models at the learning level. What that means is that, rather than doing a mathematical average of all the predictions of different models, the different predictions are learned together along with its descriptors to form a bigger learning model which is later used to make the final prediction. This can be understood better with the help of an example. Consider the diagram 4.2.

For this method, the data has to be split into 3 chunks. Splitting in three chunks is basically splitting the number of records into 3 equal divisions. 15 different models are generated and are tested. The predictions thus obtained are then concatenated with the descriptors to form a new bigger training dataset. These 15 different data models can be divided as 5 models each of - Random Forest, Gradient Boosting Machines(GBM), and Lasso-Ridge Regression Generalized linear models(GLMNET). These 5 models are differentiated by the parameters used to generate them.

- Random Forest: 5 models are generated by varying the number of trees the algorithm has, and the number of descriptors per node used to define a tree.
- GBM: variable parameters include number of iterations or trees, number of observations in the terminal nodes, tree depth, and the shrinkage parameter.
- GLMNET: the only variable which we varied here was the alpha which gives the balancing factor between the algorithm favoring Lasso to Ridge regression.

The following is the stepwise explanation of the algorithm:

- The dataset consisting of descriptors and the final class to be predicted is taken in the form of a csv file and fed into the machine. (this dataset is already preprocessed to have desired descriptors present)
- This data set is then sub-divided into three parts where each part has the same number of records. The three divisions are called as - chunkOne, chunkTwo and chunkThree.
- Using the sub-dataset chunkOne, 15 different data models (M1 through M15), 5 each from individual algorithms as described above are generated.
- The sub-dataset chunkTwo is then used to test these models giving 15 set of predictions (P1 through P15).
- Predictions obtained in step 4 are then column concatenated with descriptors of chunkTwo to form 15 new data subsets as shown in the figure. These new datasets are the individual rows in the L2Train dataset.

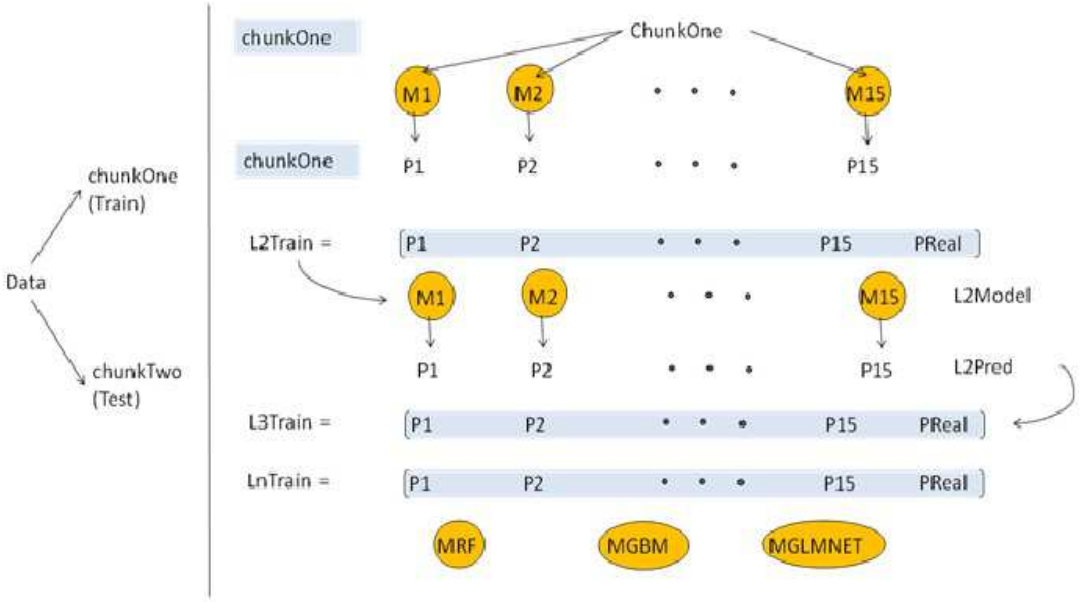

Figure 4.3: Ensemble Method Two : Recursive Learning of Prediction

- They are then row binded (add these 15 dataset row wise) to form a bigger L2Train(Level 2 train) dataset.
- This new dataset is then used to create 3 standard models, one each of the individual algorithms which serves as the final model.
- ChunkThree is then tested on the models generated in Step 7 to get the final prediction.

4.2.2 Recursive Learning of Predictions

This second method follows the idea that similar datasets with similar distribution will be fit in a similar fashion by algorithms. Hence if the datasets originate from similar source, a fit on such datasets would follow a similar pattern on the inaccuracies as well. These inaccuracies can be learnt separately to guide the learning towards the actual value. This can be explained better with the help of following example.

As can be seen from the diagram 4.3, the dataset is mainly divided into 2 parts. Also the models generated are in a similar fashion as done in the Averaging by Learning algorithm. The difference is in the way the predictions made are learnt. The 15 set of

predictions are column-binded along with the real value to form the successive level of training data. The step-wise algorithm for the same is as shown below:

- The dataset is divided into two parts - chunkOne and chunkTwo- with a distribution of chunkOne being 2/3 of the records and chunkTwo the remaining part of the records.
- chunkOne is then used as the training dataset and 15 different data models are generated as explained earlier
- Next the same chunkOne which is used to generate the models is used as the testing dataset and 15 set of predictions are made. One for each model.
- These 15 sets of predictions are then column-binded with the real value of the class to form a new data subset. Let us call this as L2Train dataset. (Level 2 Train)
- The L2Train dataset is then taken and 15 new models are generated with the same parameters as done in level 1. This is the step which helps us learn the inaccuracies/accuracies assigning them to the actual value. Hence 15 Level 2 models are generated.
- These Level 2 models are then again tested on chunkOne to get 15 sets of Level 2 predictions.
- Step 4 to Step 6 are repeated recursively to form a multi level model structure which consists of different models at each level. These models have to be preserved for later use.
- After a certain number of iterations, a standard set of parameters are used to generate the final model.
- For testing, chunkTwo is tested with all the intermediate models at each level in the same order and the final model gives the final prediction.

4.3 Scaling the Algorithms

The above mentioned algorithms are very computation intensive. Through experimentation it could be seen that they require a long time for execution. Hence to reduce this techniques were devised to distribute the computation. We demonstrate scalability at two levels. The first level is at the individual algorithm level and the second one at the ensemble level. A clear understanding of the algorithm's features is required for it to be parallelized. Random forest is based on the decision trees and each of the decision trees prediction is taken into consideration. Hence each tree is different than the other and there is no inter-dependency between them. This gives us the advantage of generating trees parallel across multiple cores or nodes and then integrating them to form a model with large number of trees. But at the same time, each tree is built on the entire dataset. Hence each node where the tree is built should have the exact copy of the dataset in use. Gradient Boosting Machines (GBM) also use decision trees in their approach. But successive trees are fit to the residual of the current tree. This residual is represented in terms of the loss function whose mean square error is given by the difference between the real and predicted value. Hence the trees are dependent on each other and cannot be processed in parallel. Hence at the individual algorithm level only different models of the algorithm can be generated in parallel. Lasso-Ridge regression Generalized Linear Models (GLMNET) works on the entire dataset. Hence the parallelization of this algorithm works a lot like GBM where different model generations can be done in parallel having different parameters.

4.3.1 Scalability on Mahout

Mahout has implementation of many classification and clustering algorithms. Random forest implementation on Mahout was carried out which successfully scaled on a 32 node local dell cluster with mahout. As mentioned earlier, mahout uses hadoop as its backend to support distributed programming. MapReduce is the paradigm which hadoop follows. The dataset was stored in the hadoop data file system in a CSV format with all the preprocessing done. While using the API for random forest we could define

the number of nodes to be used. Hence different models can be generated but not all at the same time. The API offers parallelism by breaking down the number of tree generation and also the data. Another important parameter which plays a major role in understanding the scalability is the split size or the number of splits. As the number of splits increases the utilization of each node increases as more work can be done in parallel. But at same time more splits to the data means each set of trees generated in a split has lesser data to build the model. Hence the accuracy may be comprimised. Details about the run time and the dependency of the same with different features of random forest are explained in the next chapter.

4.3.2 Scalability on R

R provides us with various user-built packages which help us scale a number of algorithms. With the help of these packages above mentioned algorithms were scaled both at individual and ensemble level.

Random Forest: Random Forest algorithm was implemented using the package 'randomForest' [**?**]. This package gives options to specify the various parameters which included the number of trees, number of variables at each node, if the variables have to be replaced, etc. Variables of interest in this work were the number of trees and the number of variables at each node. To parallelize the tree generation, the 'foreach' [1][31] package with the '

rf ¡- foreach(ntree=rep(250, 4), .combine=combine, .packages='randomForest')

Would generate 250 trees, each on 4 CPUs parallel, for a total of 1000 trees. In order to use the foreach package in the way shown above, a parallel backend has to be registered which informs the existence of a number of CPUs which could be used in parallel. This is done using the doMC package [2]. It helps define the number of CPUs we are dealing with. In the above case the number is 4. Any number of CPUs can be registered. Each node of the local cluster consisted of 16 cores. Hence a maximum of 16 cores could be defined using the doMC package. But doMC package only works at multi-core level. That is, doMC will not support parallelism on a network with multiple nodes. For extending the parallelism to multiple nodes another package called

'snowfall' [15] which is a wrapper package around 'snow' package [27] was used. With the help of this package the backend was defined with the required nodes. The type of communication could also be defined which exists between the nodes, which in our case is MPI. Snowfall provides functions with the help of which a cluster of specific nodes can be created. 'Rmpi' package [32] needs to be installed on all the machines in the cluster to use mpi as the communication interface. Once the cluster is created, it offers functions to load the required libraries across the cluster and to transfer data to all the nodes. A function which invokes a function in all the nodes of the cluster and gathers results from each is called from the master node. Hence two levels of parallelism is followed. One is at the muli-node level where data is distributed to individual nodes, individual functions are distributed to the nodes, required libraries are loaded and the functions are then invoked. A unique set of parameters is passed to each node such that each node generates unique models. At level 2, that is at node level, distribution of generation of decision trees across all the cores of that node is done. This reduces the execution time which will be demonstrated in the next chapter.

Gradient Boosting Machines: As parallelizing the generation of trees is not an option, parallelization is observed to create multiple models across many cores/nodes. The parallel version of the lapply function in R is used for the same. There are 2 variants of the lapply :

- Mclapply: this function is supported by the 'multicore' package [28] in R and helps us distribute the computation over the available cores. This again does not help our cause of running code on multiple nodes.
- sfLapply: This is function which is the parallel version of lapply offered from the snowfall package. This function basically helps us to invoke functions on multiple nodes.

So in the case when generated models are greater than the number of cores available, generation of models can be distributed over multiple nodes. For simplicity one model per node is generated. Hence sfLapply function was used. sfLapply function takes in a list and passes the list variables in sequence to the nodes to which it caters. Hence an

array of list can be generated which can contain the different parameter sets to generate different models. The elements of this array, which in itself is a list, can be passed to different nodes to create its own unique model.

GLMNET: GLMNET algorithms are not as compute intensive as random forest or GBM, which can be seen in the following chapter. Hence GLMNET models can be generated much faster using the normal apply. But creating many nodes still requires a sequential long time. Hence use of sfLapply as discussed in above helps us distribute the work. There are other constructs such as the 'foreach' package which can also be used. But the overhead for 'foreach' compared to the compute time is high. Hence it makes it better not to introduce 'foreach' for the GLMNET computations. Next chapter provides the use case for parallelizing the GLMNET algorithm.

Scaling the Ensemble Approach: The individual algorithms in the ensemble are scaled as explained above. This section describes the way each of the model is simultaneously generated across multiple nodes. The procedure for doing this is as given below.

- Load the dataset and make the necessary divisions as given in the algorithm before.
- Install snowfall, rmpi, randomForest, gbm, glmnet, foreach, doMC all the other supporting packages on all the nodes of the cluster in scope.
- Create functions for individual algorithms - randomForest using the foreach package, gbm and glmnet using the arguments of the function as the parameters in the algorithm implementation.
- Load the all the required packages. In particular load the snowfall package.
- Create the cluster of required number of nodes using the built in function provided by the snowfall package. Use MPI as the communication interface.
- Export the data required for the computation to all the nodes. Also load the other packages in all the nodes.

- Export the functions for the individual algorithms to all the nodes.
- These individual algorithms are written in such a way that depending on the argument of the function, a particular function will be called at that node.
- 15 models are distributed as 1 model per each node. The first 5 will be randomForest, next 5 GBM and the last 5 will be GLMNET.
- Invoke appropriate functions in the respective nodes by calling the main function in the master node and adjusting the arguments to those functions.
- Hence 15 models are simultaneously executed on 15 nodes and they return the model object back the master node.
- In the master node, all the model objects are stored a list of objects and can be used later for further testing.

Chapter 5

Scalable Data Analysis : Implementation

Preceding chapter introduced 2 new algorithms which were developed (Averaging by learning and Recursive Learning of Predictions) to implement ensemble learning. A brief description of the algorithms in scope and the method of parallelizinf them were also listed. This chapter presents two of the use cases for the system developed. They are:

- Heritage Health Prize: Is a medical dataset downloaded from an online machine learning competition website called Kaggle
- KDD2012: is a social media dataset downloaded from kdd official website.

5.1 Heritage Health Prize(HHP)

Problem Statement: To determine the number of days a patient will be admitted in the hospital next year based on the claims made by the patient this and the previous years. This would help the hospitals to reach out to patients in advance and prepare themselves accordingly.

5.1.1 Data preparation and pre-processing

The data downloaded was in the raw format and required a lot of preprocessing before it could be used by our algorithms. The dataset was downloaded from the kaggle website which hosts a number of machine learning events and provides downloadable datasets for the same. The Dataset consisted of 5 main tables.

- Claims Table: This is the main table which gives information about a patients activities over the last 2 years. It has fields explaing the variuos claims made by

the patient for various procedures and primary conditions.

- DaysInHospital Table: Gave the number of days the patient was admitted to hospital for the last two years
- Primary Condition Group: A look up table for different primary conditions a patient which are referenced in the claims table.
- Procedure Group Table: A look up table for different procedures undergone by a patient which are referenced in the claims table.
- Members Table: Provides information about various members such as age, gender, day since first service.

Pre-processing involved extracting useful information from this and representing it as a CSV file. The data was prepared using the milestone paper [25] as a reference. Though the dataset was not exactly recreated, the descriptors were designed as per the paper. A number of experiments were conducted on this dataset which will be discussed in the following few sub-sections.

5.1.2 Implementation on Mahout (Random Forest)

- The first experiment conducted was on the Mahout framework to check the strong scalability of the random forest. Strong scalability analysis provides a measure of execution time by doing same amount of work on a range of processors. As described earlier Mahout supports scalability using Apache hadoop with Map - Reduce paradigm. The local cluster was setup as follows:
  - Number of nodes : 16 dell nodes were used to create a local cluster.
  - Each node consists of 2 processors, each having 8 cores. Hence an over all of 256 cores were available for parallel computation.
  - One node serving as the master with 15 worker nodes. The process is triggered from the master node and work gets distributed. The config files were edited appropriately and were distributed to the cluster for its smooth functioning.

For implementing the process the following parameters were considered:

- No of Trees : For strong scalability analysis, an over all of 800 trees were generated which was kept constant for all the strong scalability experiments.
- Split Size: This attributes to the number of chunks data is divided into. This plays a evry important role in the eveluation. The split size determines the number of data chunks avaliable for parallel computation. Hence if there are lower number of data chunks as compared to number of parallel computation units (nodes/cores), the extra units may not provide additional scalability improvement. Note should also be made that increasing the number of chunks means each of the chunk size is small, hence accuracy of the models generated may be affected. We have kept this constant to have 10 chunks.
- The number of nodes: This value is increased linearly to evaluate the execution time performance of the algorithm.
- Number of descriptors per decision tree node: This defines the different attributes considered for developing the model.

The above diagram 5.1 shows strong scaling analysis using mahout. The graph shows that there is an initial improvement in the execution time when more than 1 node is used but as and when the number of nodes increases to 10 the graph flattens. This is because we have kept the splitsize constant which gives 10 parallel processes. Hence using more than 10 nodes for 10 parallel processes does not give improve the execution time any more.

- The weak scalability evaluation is the measure of execution time performance when equal work is done on each node or computation unit. The evaluation is done by increasing the number of computation unit which in-turn increases the over all computation as each unit does same amount of work.

For the Weak scalability we generated 50 trees on each node as part of the computation and kept the split size constant to have 10 splits. Weak scalability analysis is as

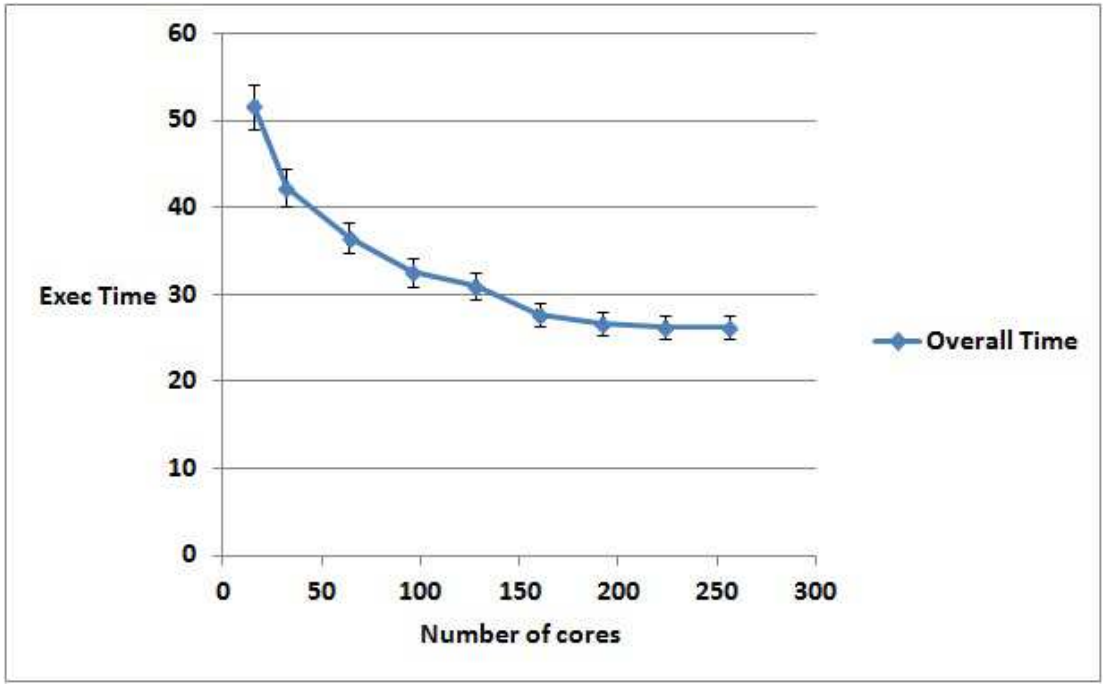

Figure 5.1: Results of strong scaling in Mahout platform on HHP Dataset

shown in the graph 5.2.

5.1.3 Implementation on R

Random Forest

Random Forest: was parallelized on the number of tree generation as discussed earlier. Two types of scaling were observed. Strong Scalability was observed similarly with following steps:

- 'snowfall' package is used to register a cluster of nodes.
- Functions provided by the package are used to distribute the function and the parameters required to each node of the cluster. Similarly the data is also distributed to each node of the cluster.
- 'doMC' package is then used to register a backend at each node level. This package gives us the flexibility of using each core of a node for computation.

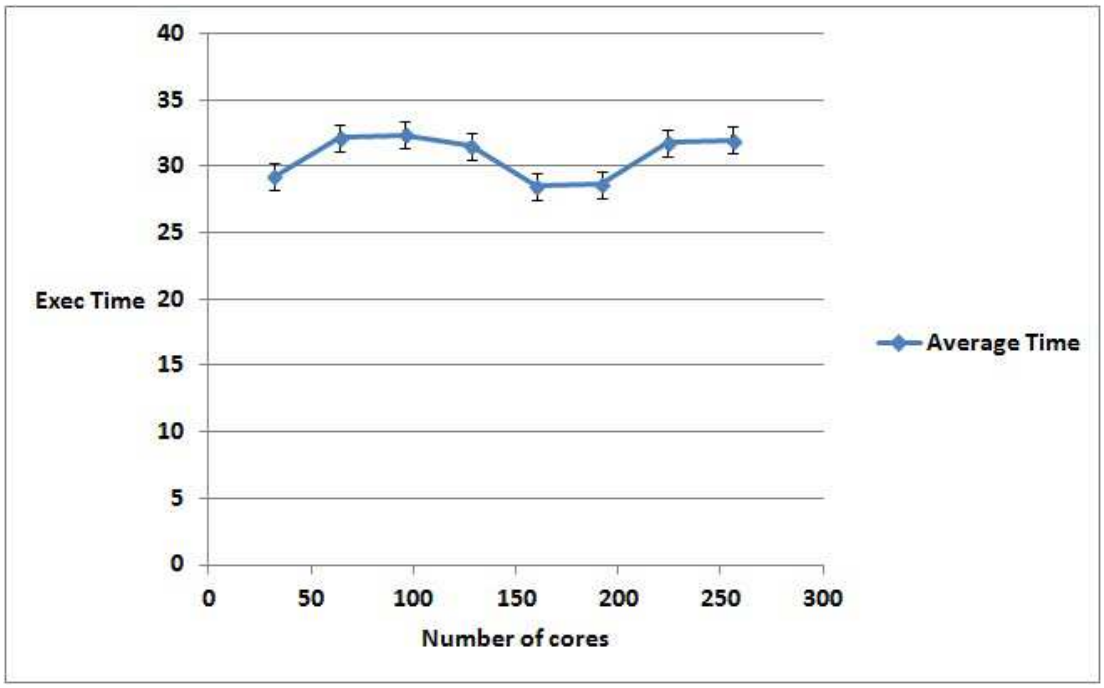

Figure 5.2: Results of weak scaling in Mahout platform on HHP Dataset

After making these configuration of required packages, we trigger the run from the master node. For random forest we generate 800 trees over all for strong scalability where each core contributes for generating a part of number of trees. The graph between the execution time and the number of cores is as shown below. For this setup the following parameters of random forest were used:

- Total number of Trees : 800
- Number of descriptors per node of the tree : 5

It can be seen from 5.3 that as the number of cores increase the decay in the execution time is exponential. For weak scalability, 100 trees were generated on each core with the parameters kept the same. The measure of weak scalability is as given below. In the above graph 5.4 there are 2 series. Overall Time is the graph of the elapsed time which constitutes both the execution time as well as the communication time between multiple nodes. The results show that as the number of nodes increase the communication time increases linearly.

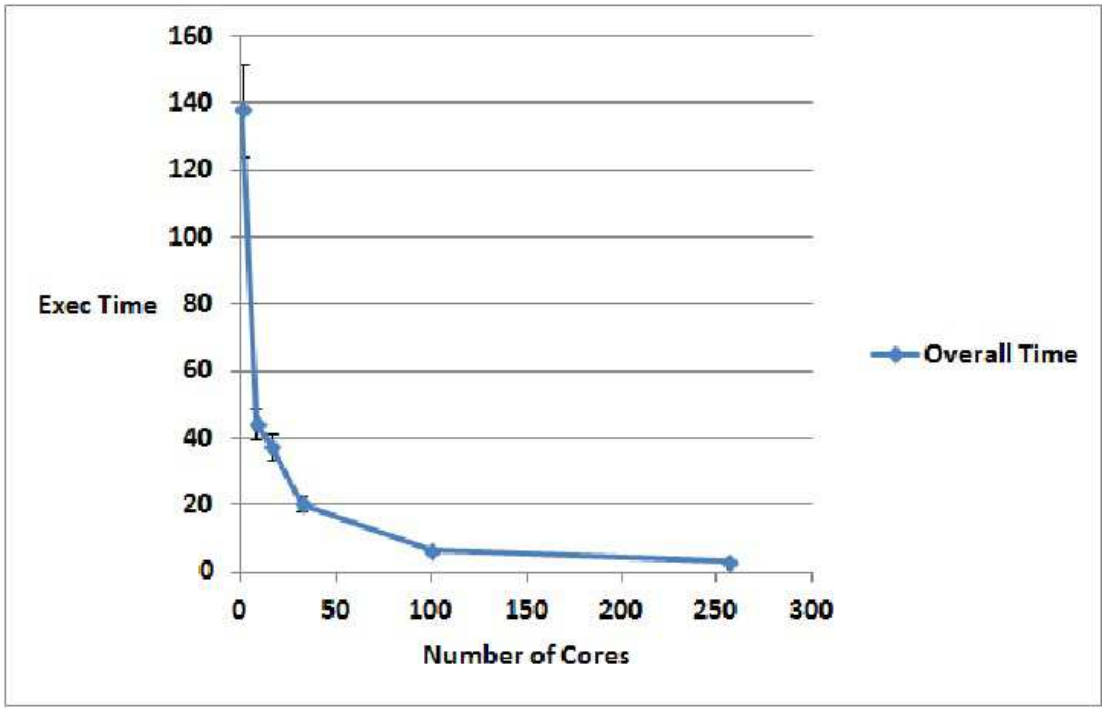

Figure 5.3: Random Forest implementation demonstrating strong scaling on HHP dataset in R

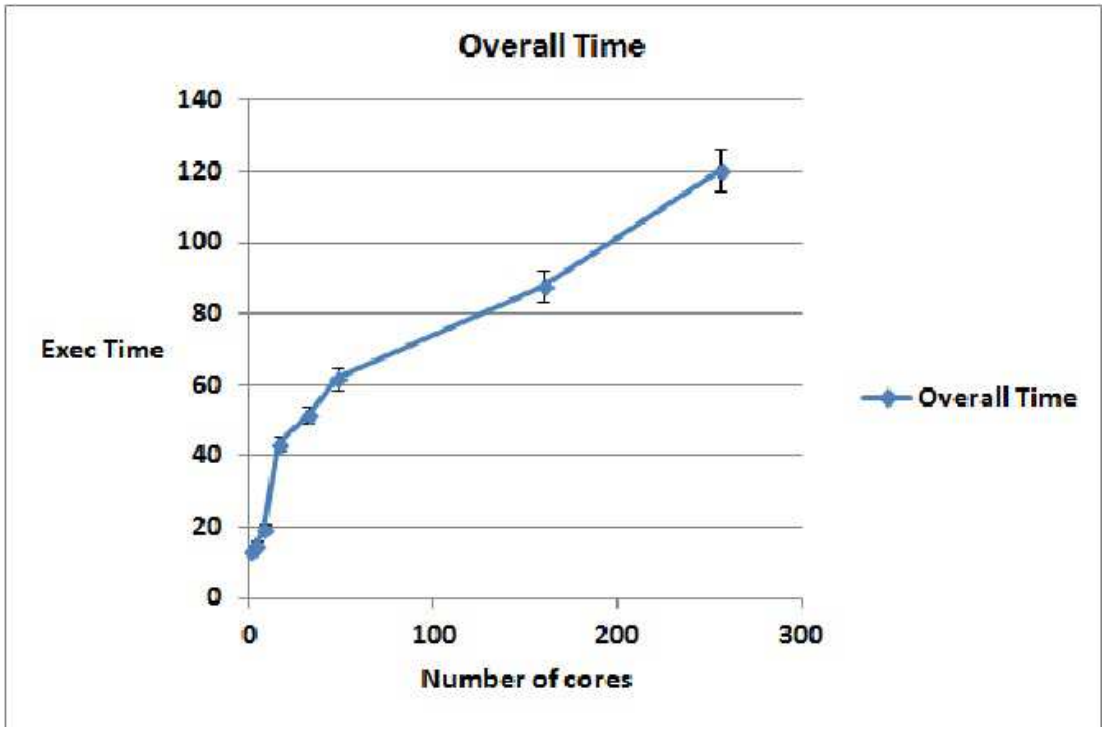

Figure 5.4: Random Forest implementation demonstrating weak scaling on HHP dataset in R

Gradient Boosting Machines (GBM)

GBM also generates decision trees and the final model is built on a number of weak models by integrating all of them. Since consecutive trees are generated by taking the gradient or the residue of the previous tree, in short is dependent on the previous trees, parallelizing the tree generation was not a good idea. This issue was tackled by simply generating more number of models and observed weak and strong scalability on them. The process of implementation is as given below:

- Set up the same configuration as done for random forest.
- Used the same set of packages to distribute the data.
- Generate an over all of 256 models on a range of cores. Hence fixed amount of work is done. An execution time evaluation of such a set up is used for measuring the strong scalability.

For weak scalability :

- 1 model per core was created.
- The number of models was then increased with linear increase in the number of cores.
- The parameters were kept the same as the ones for strong scalability and an over all of 256 models were generated.

Following are the set of parameters used for GBM evaluation

- Number of Trees : 100
- Shrinkage Parameter : 0.005
- GBM depth : 4
- Number of observations per node : 10

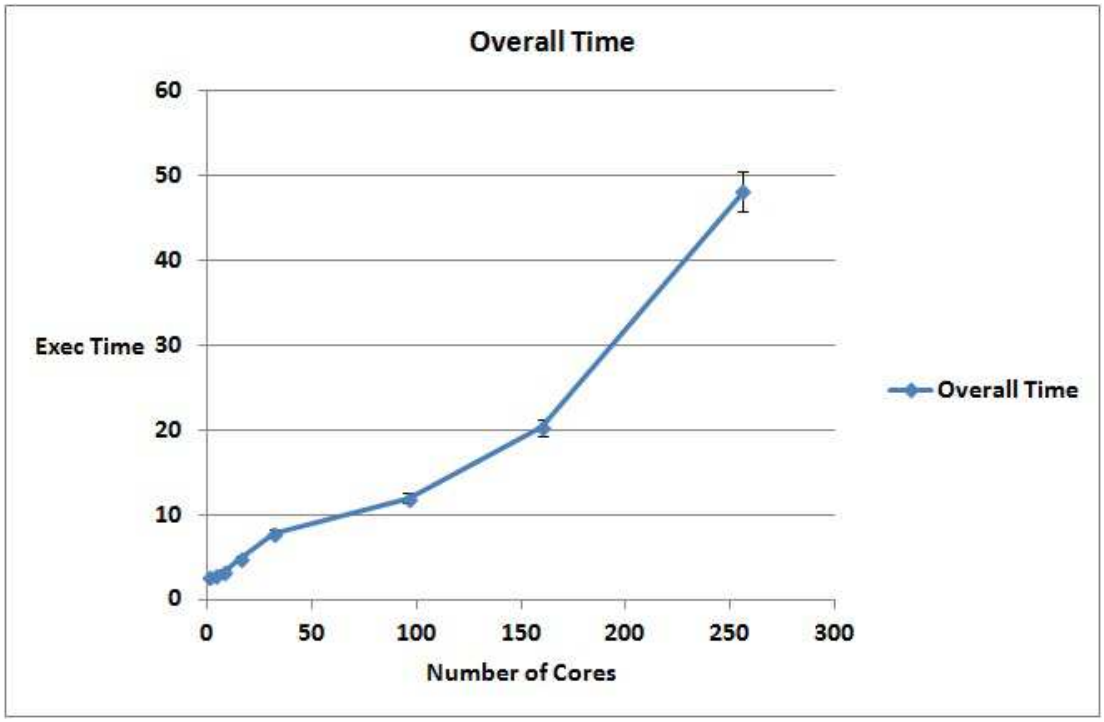

Figure 5.5: GBM implementation demonstrating weak scaling on HHP dataset in R

As can be seen in 5.5 the results of weak scaling are similar to the ones obtained for Random Forest. The Overall Time shows the execution time which includes the communication time involved in dealing with multiple nodes. The Average Time series is nothing but the execution time alone averaged over the number of nodes used for that computation. An analysis of strong scaling is shown in the graph 5.6. It follows a similar trend as for random forest with varied times. For implementing strong scaling we generated 256 models on a range of number of cores. The parameters were kept the same as mentioned above.

Generalized Linear Model - Lasso and Ridge Regression (GLMNET)

GLMNET does not use decision trees for generating the models. To parallelize the generation of individual model, the source code in the package had to be changed. That would not only be a tedious job but also would require a lot of time to create a new modified package. Without having to deal with modifying the package, scalability was obtained just the same way as in GBM. The process of evaluating strong scalability

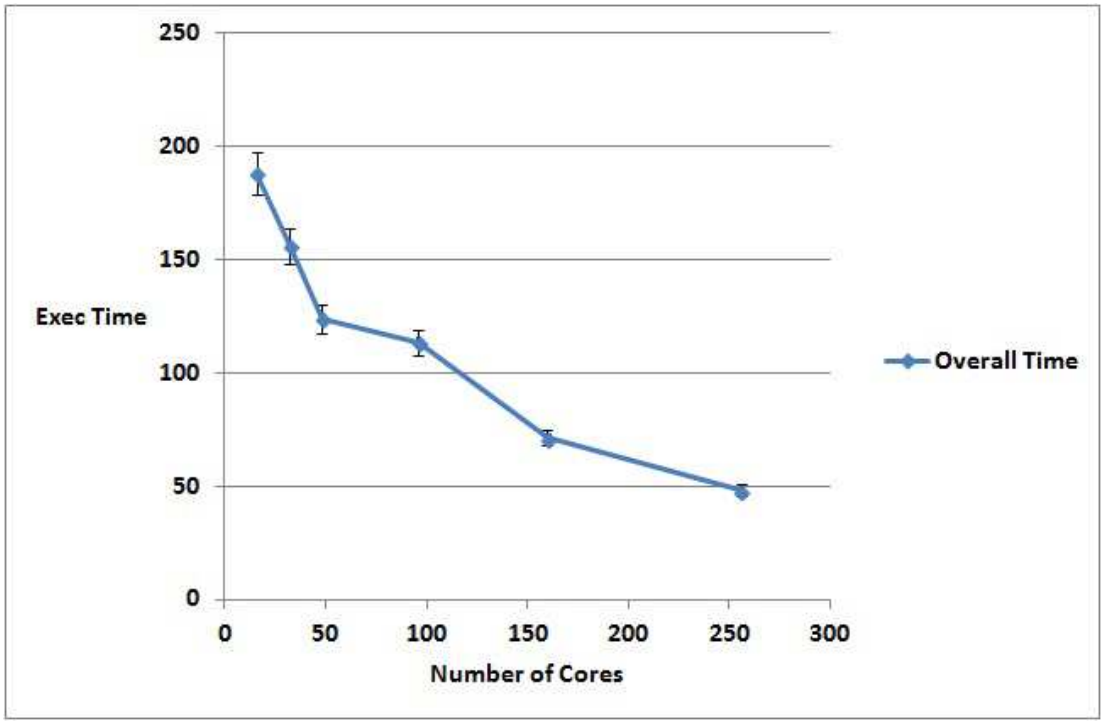

Figure 5.6: GBM implementation demonstrating strong scaling on HHP dataset in R

was similar to GBM where a total of generating 256 models was consdered as the final work. This was generated over a range of cores and an execution time evaluation was done. Weak scaling was measured by generating individual models on each core. Hence the number of cores determines the total work done and they have a linear relation. Hence the plot of weak scaling is as given in 5.7 As seen earlier the Overall time is the execution time which includes the communication delay between multiple nodes and Average Time is the plot of the average time on individual nodes/cores. The strong scaling analysis is as shown in 5.8 . It can be seen that the initial reduction in the execution time follows a steep curve but after a certain set of nodes the curve shows slower reduction in execution time.

Averaging by Learning (EnsembleM Learning Method 1)

As described earlier, averaging by learning is a mechanism in which 15 different models are generated. The predictions made by these models are then row-binded and learnt again to obtain the average of the predictions. The feature which was exploited to

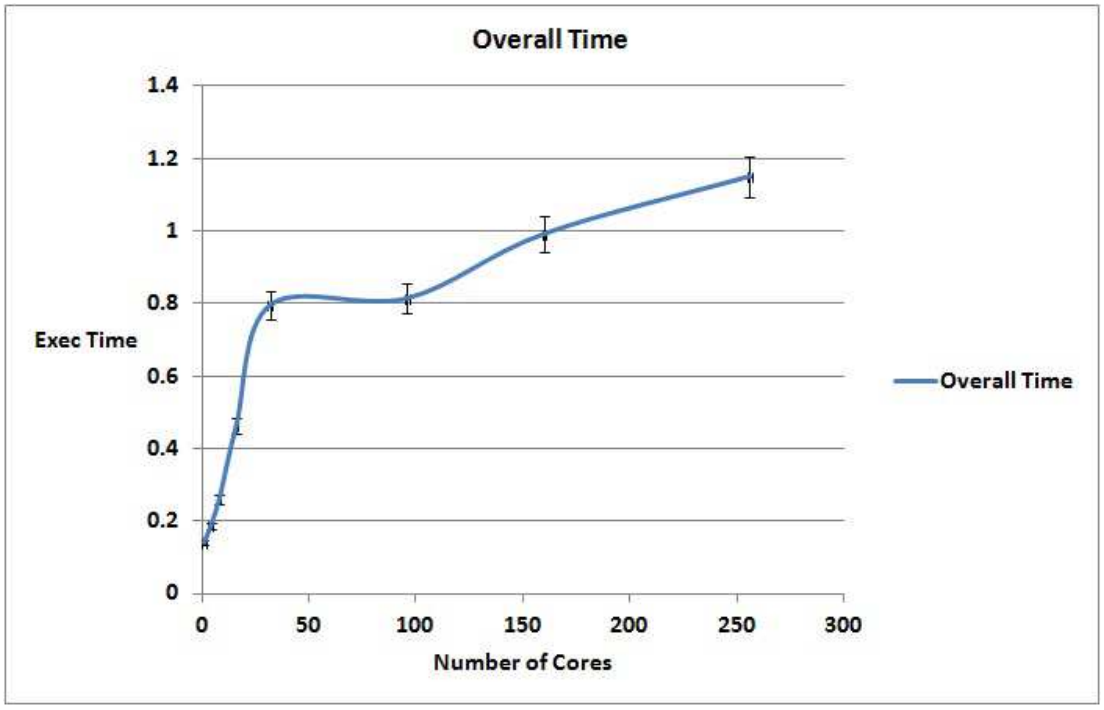

Figure 5.7: GLMNET implementation demonstrating weak scaling on HHP dataset in R

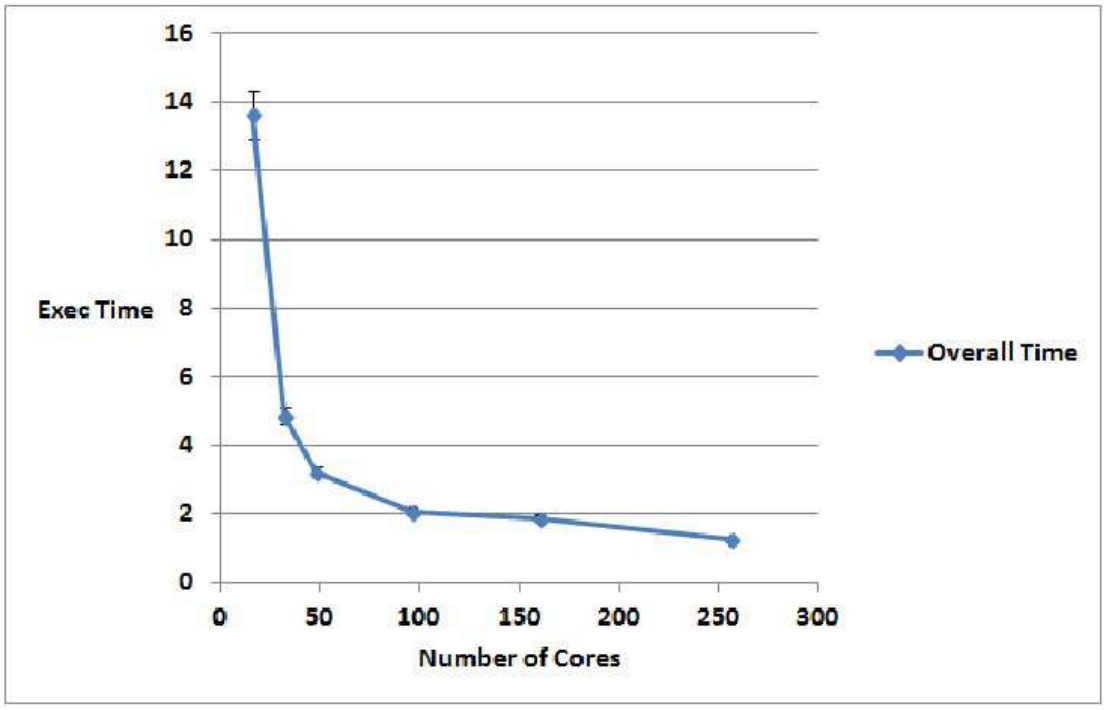

Figure 5.8: GLMNET implementation demonstrating strong scaling on HHP dataset in R

parallelize this mechanism was the generation of 15 independent models. These models were generated in parallel. The procedure to do so is as given below:

- Install necessary packages on each of the worker nodes of the cluster.
- Use snowfall to register a cluster of required number of nodes. Use MPI as the communication layer between them.
- Distribute the required data to each node and load the packages, in particular the doMC package.
- Register a backend of 16 cores using the function which gets distributed by the 'snowfall' package. Hence this function will carry information about what model will be generated on each node.
- Create an array of lists where each element of the array consists of the parameters of the 15 different models. The snowfall distributes this array in such a way that the first list goes to the first node, second goes to the second node and so forth.
- Hence the lists are carefully filled to create the required models.
- The return of these functions is then stored in a list of objects where each element represents a model.
- These indiviual models are then used to make predictions on the same data. Hence 15 sets of predictions are made. The process of 15 predictions are also distributed accordingly on the cluster.
- These predictions are later rowbinded and 3 models , one each from Random Forest, GBM and GLMNET are created as the final model, which is built over this ensemble of data and predictions.

An observation of the execution time in generating these models parallely ranging between 1 to 16 nodes was made. The results of this experiment are as shown in 5.9 . As expected the execution time decreased from 250+ seconds to close to a min. The time taken by individual nodes varied a lot due to different models generated on each node. The above represented is the average time.

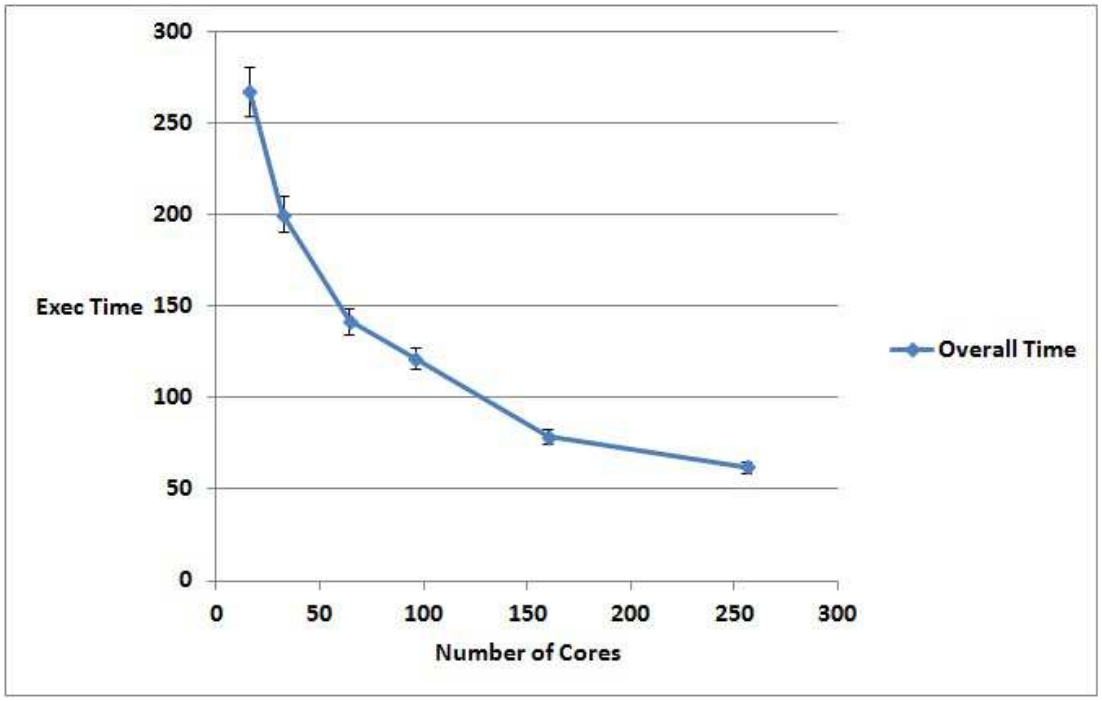

Figure 5.9: Averaging by Learning on HHP in R

Recursive Learning of Predictions(Ensemble Learning Method 2)

In this second method of ensemble learning, the procedure of generating the models parallely was the same as the previous method, but this also adds the prediction using those models to 'N' levels. We used N=3 hence 3 levels of learning and prediction were implemented. The execution time of individual learning and prediction is given as a separate graph. The following figure 5.10 shows the plot of each.

- Overall Time: It's the series where all the execution times are summed. It gives the graph of the complete procedure. It gives the complete experimental time for the ensemble learning procedure.

- Level1Learning: It's the graph of level 1 learning where the input training dataset is the dataset prepared from the HHP data.

- Level1Prediction: this series gives the execution time of the level 1 prediction taking the same training dataset used to generate the models as the testing dataset.

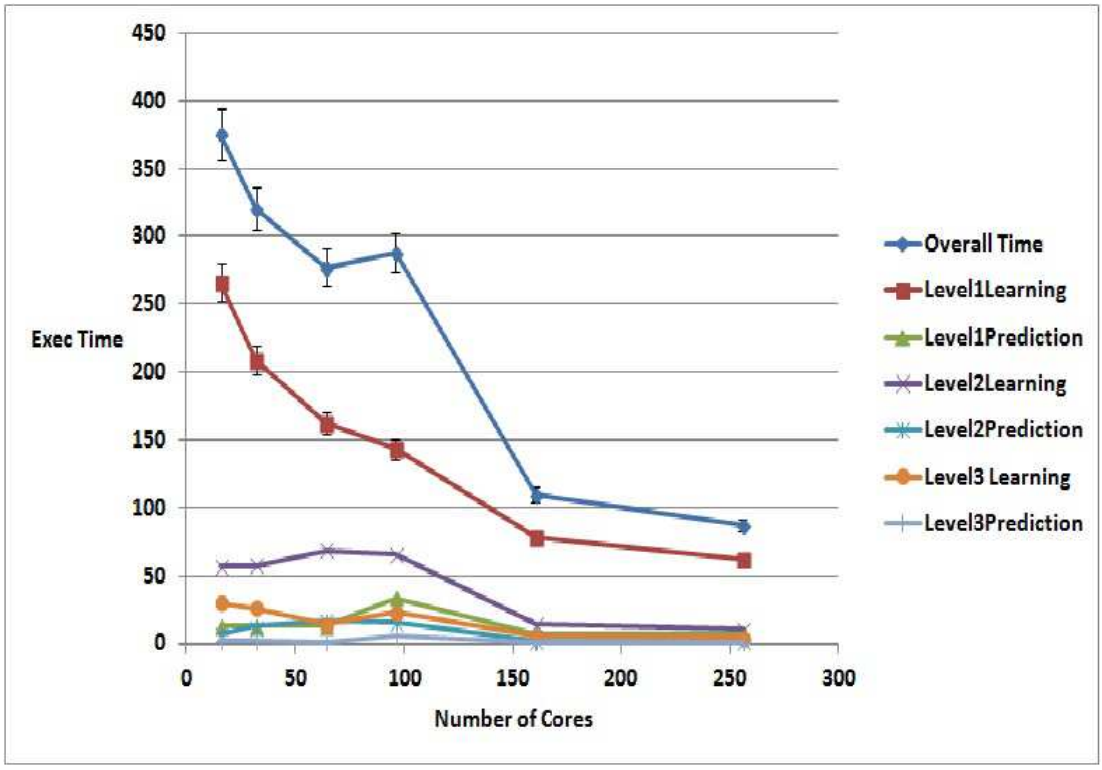

Figure 5.10: Recursive Learning of Predictions on HHP in R

- Level2Learning: This is the level 2 training where all the predictions are column-binded and are trained along with the actual value.
- Level2Prediction: this is level 2 prediction time graph using the same training dataset and the models generated in series 4.
- Level3Learning and Level3Prediction: are the level 3 learning and prediction graphs.

5.2 KDD 2012 DataSet

The second use case is the dataset downloaded from KDD cup 2012. Online social networking services have become tremendously popular in recent years, with popular social networking sites like Facebook, Twitter, and Tencent Weibo adding thousands of enthusiastic new users each day to their existing billions of actively engaged users. This dataset provides information about users, their profile and their history with respect what they follow and what they are interested based on the popular social site Tencent

Weibo.

Problem Statement: To predict whether or not a user will follow an item that has been recommended to the user. Items can be persons, organizations, or groups or entities on the social site.

5.2.1 Data representation and pre-processing

The data downloaded came in a set of 7 text files. Two files out of these are the recommendation and testing files. Remaining 5 text files constitutes information concerning each user, their interactions with other items, their interests and a measure of it, a file containing list of items each individual follows already and related information. In the data pre-processing stage, a training file is defined by taking the user and the item being recommended from the recommendation file and attaching information about this user and the item being recommended. This attached information will be a part of the descriptors for the final class of whether the recommended item will be followed by the user or not. This training file was then learnt on two platforms as mentioned earlier. The results and observations are as given in the following two sections.

5.2.2 Implementation on Mahout

- Strong Scalability measure of the KDD dataset can be explained in the diagram shown in 5.11. This is the measure by running random forest on the dataset over a range of nodes. The parameters set for this experiments was the total number of trees was kept a constant of 800 trees and split size is set such that 10 chunks of data is created. The procedure of experimentation was done in the same way as done for HHP. It can again be seen that the graph again flattens as the number of nodes increases more than 10 nodes for the same above mentioned reason.

- Weak Scalability: The procedure followed was same as done for HHP. The evaluations can be summed by the figure shown by 5.12. The parameters used were - 50 trees per hadoop node.

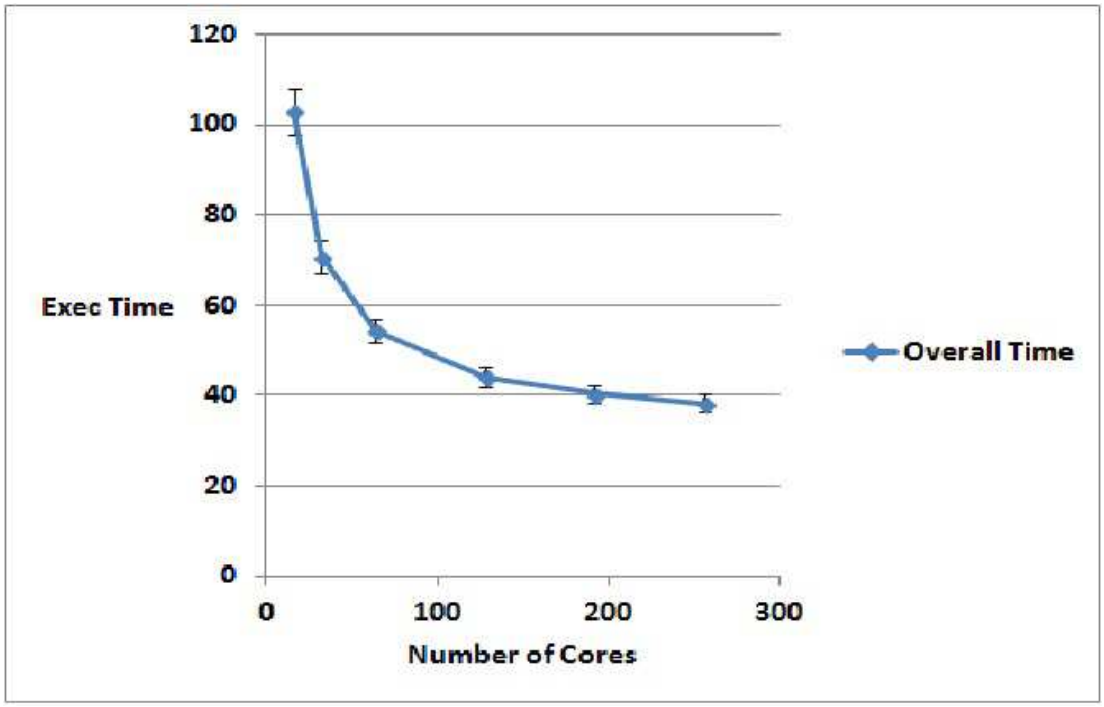

Figure 5.11: Random Foreest implementation of strong scaling in Mahout of the KDD dataset

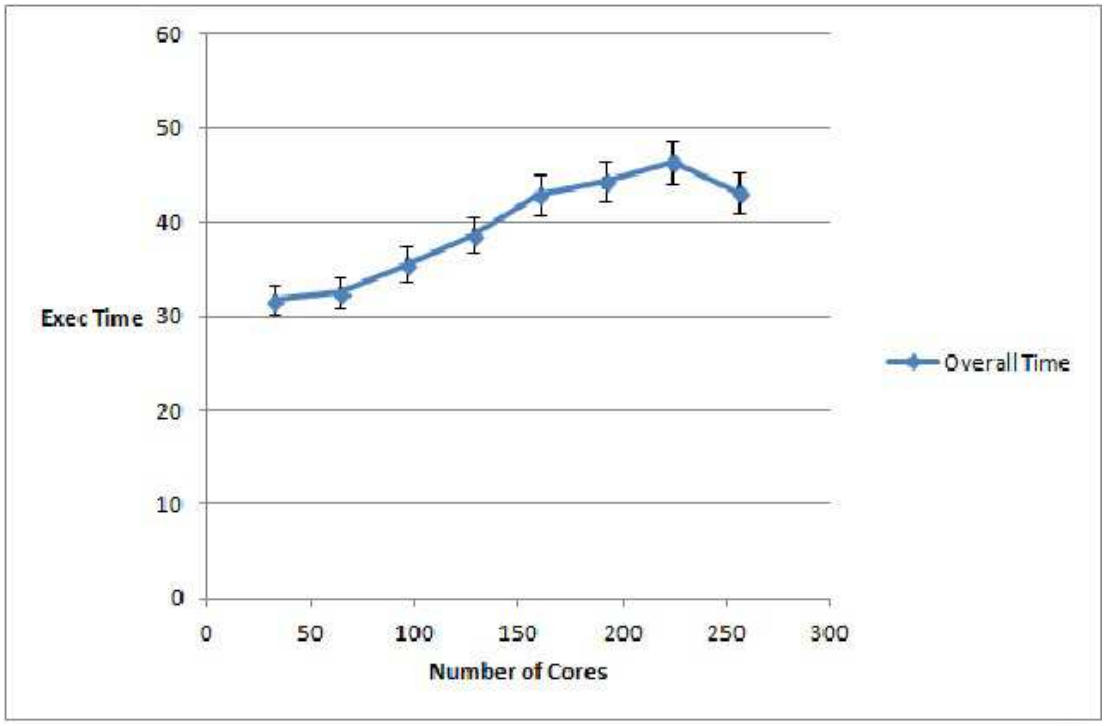

Figure 5.12: Random Foreest implementation of weak scaling in Mahout of the KDD dataset

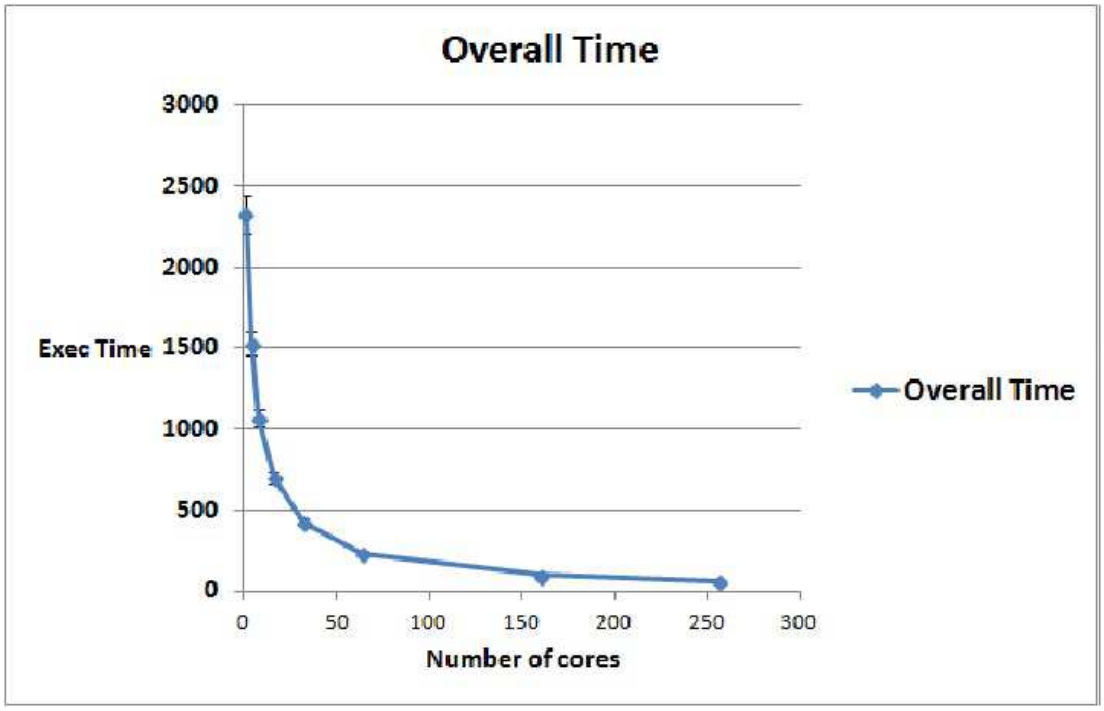

Figure 5.13: Random Forest implementation measuring strong scaling of the KDD dataset

5.2.3 Implementation on R

Random Forest

As tested in the first use case, the algorithm random forest was tested on the KDD dataset. Again testing for the strong scalability, 800 trees were generated on a range of nodes/cores. The other parameters were :

- Number of observations per node: 5
- Total number of trees: 800

As expected the execution time decreases exponentially with the increase in the number of nodes/cores 5.13. An important point on the graph which was noted was the execution time for generating this model on 16 cores or one node. It was important because later during the ensemble learning one model per node was generated and parameters were used of the same range. Weak scaling of the random forest has the following plot. Here 50 trees per core was generated and the execution time was measured as the

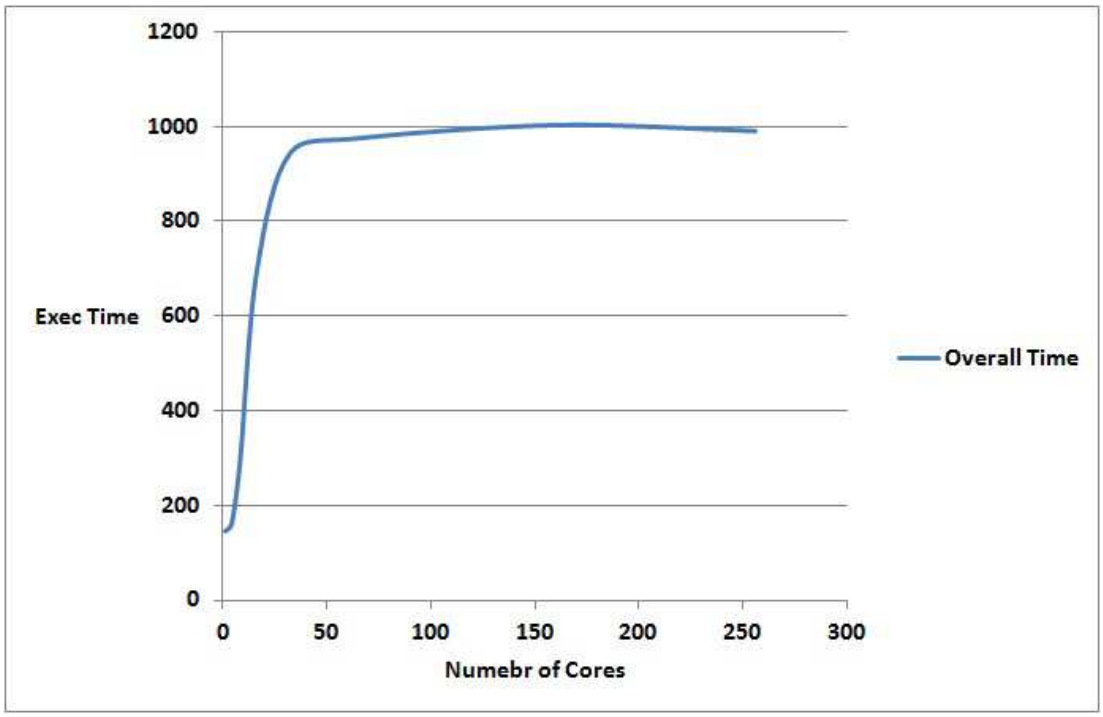

Figure 5.14: Random Forest implementation measuring weak scaling of the KDD dataset

number of cores increased. It can be seen in the diagram 5.14 that the execution time increases as the all of the cores in a node are used and increasing the number of nodes shows little increase in the execution time which basically is the communication time between nodes.

Gradient Boosting Machines (GBM)

Like in use case with the Heritage Health Prize dataset we measure the weak scaling by generating 1 model of GBM per core. It is important to note the difference in the execution time between datasets as the datasets have different size and number of descriptors. Same set of parameters were used as the previous use case with the only exception of changing the number of observations per node to 5.

- Number of Trees : 100
- Shrinkage Parameter : 0.005
- GBM depth : 4

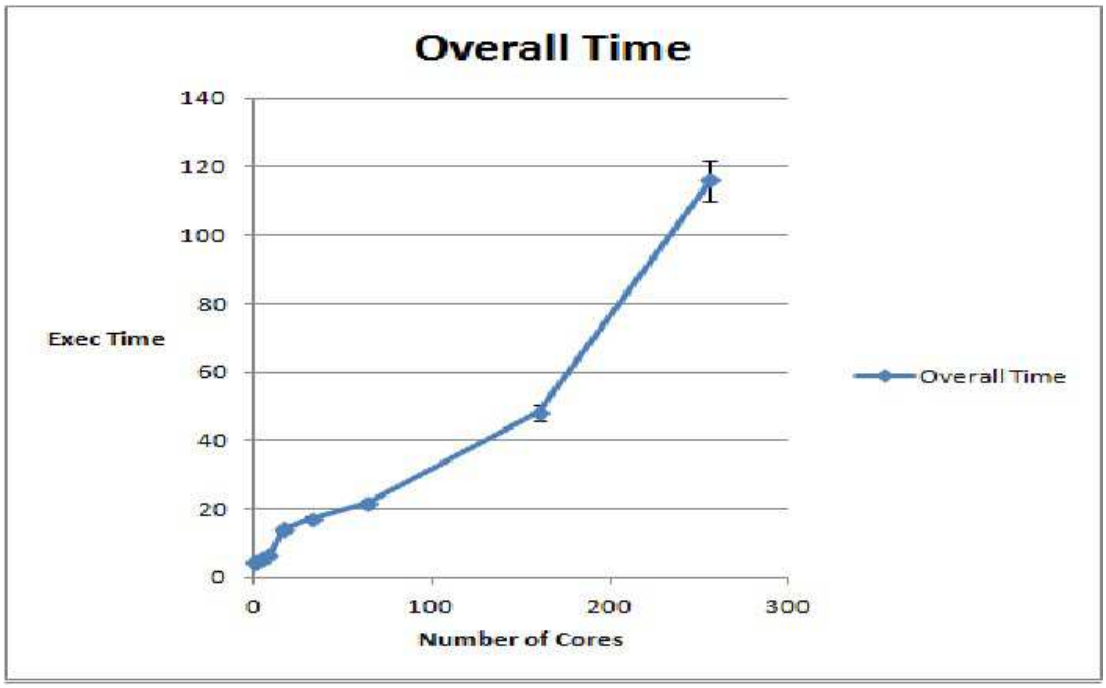

Figure 5.15: GBM implementation showing weak scaling on KDD dataset

- Number of observations per node : 5

Overall time in the figure 5.15 gives the graph of over all time for the generation of models. It includes the execution time as well as the communication time for all the results to be sent back to the master node in a multiple node cluster. Average Time is the average time of the execution time alone on each node averaged over the number of nodes. Strong scaling of the KDD dataset for the Gradient Boosting Machines is as given in 5.16. 256 models were generated with a standard set of parameters on a range of nodes/cores.

Generalized linear model (GLMNET)

The experimentation is similar to the first use case with HHP dataset. 5.17 depicts weak and 5.18 depicts strong scalability on the KDD dataset.

Averaging by Learning(Ensemble Learning Method 1)

In this Ensemble Learning method 15 different models are generated containing 5 each of random forest, GBM and GLMNET in parallel. Using these models predictions are

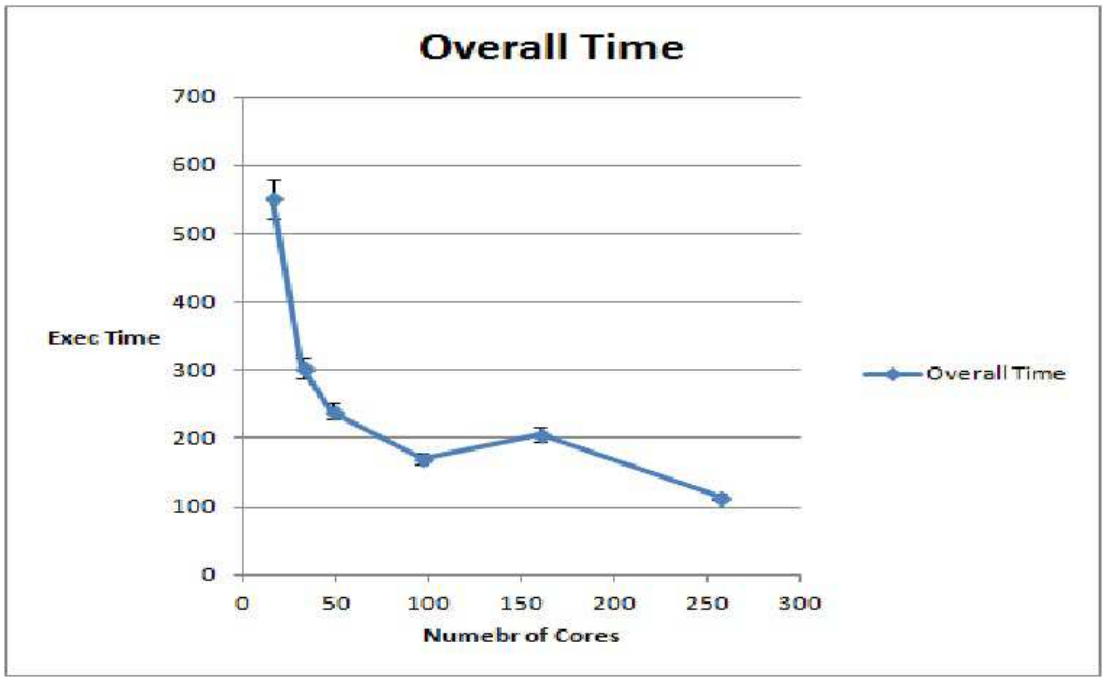

Figure 5.16: GBM implementation showing strong scaling on KDD dataset

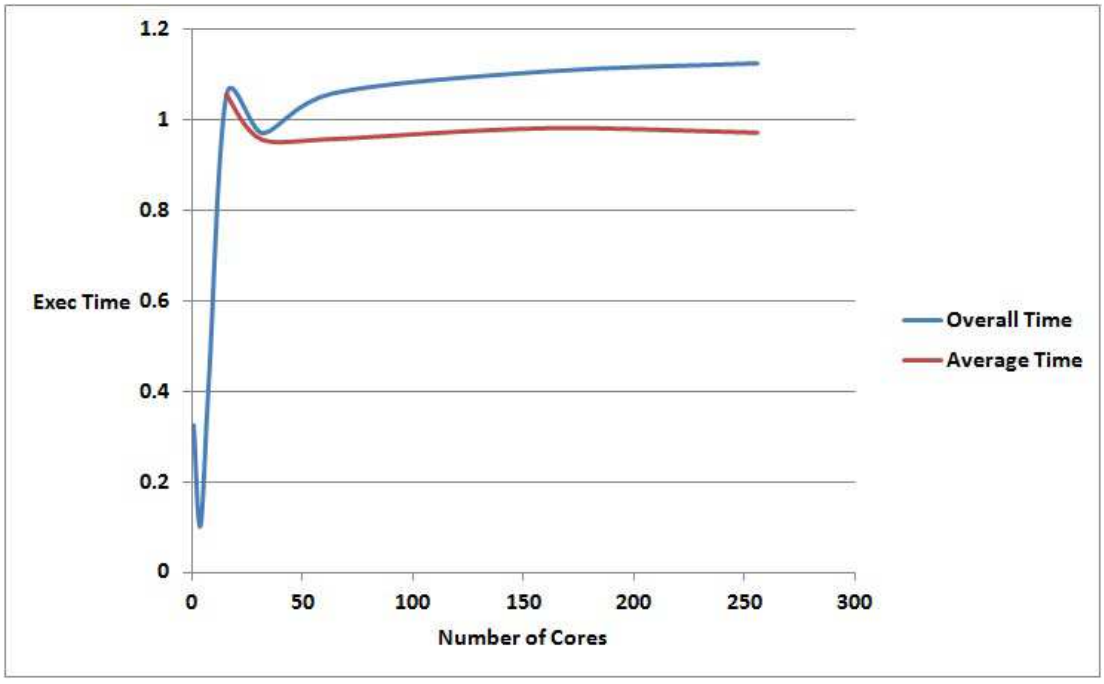

Figure 5.17: GLMNET implementation showing weak scaling on KDD dataset

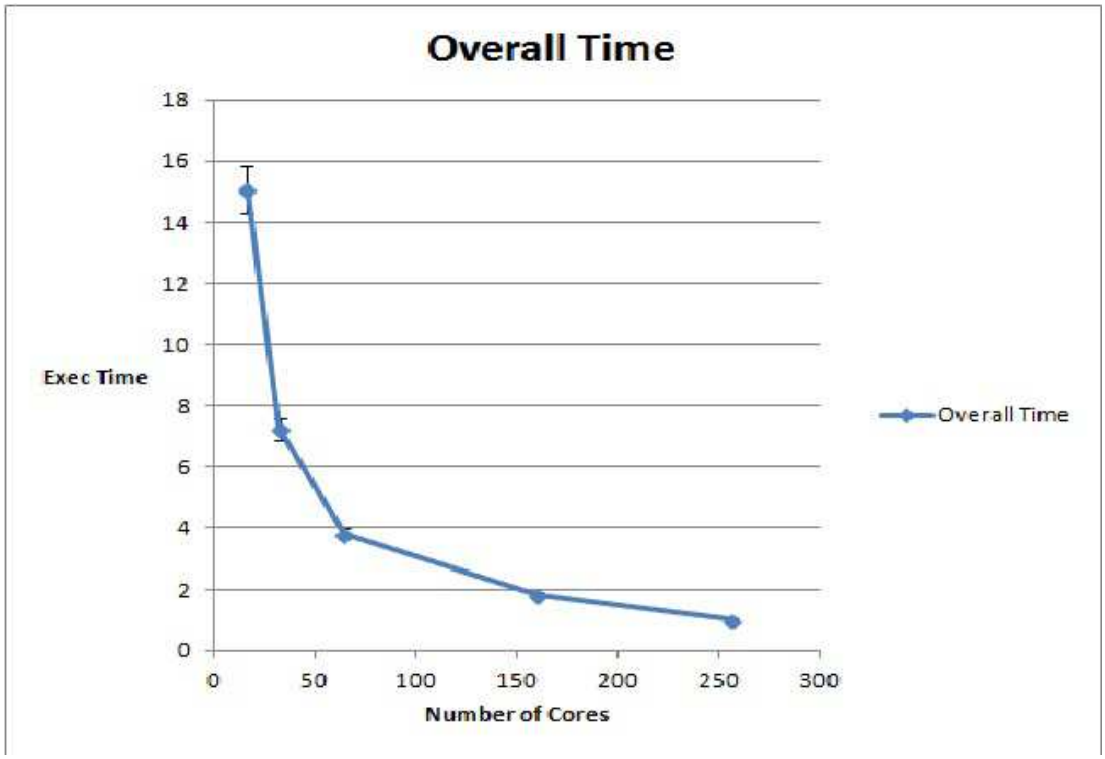

Figure 5.18: GLMNET implementation showing strong scaling on KDD dataset

made on a test data and row-bind the results with the same descriptors (Explained in Chapter 4). Then a second learning is done on this new dataset and generate the final models. The generation of 15 models in parallel are analyzed and the execution time is as given in the graph below. A point which has to be noted here is each node is limited by the model which gets generated in that node. Previous results show that random forest takes a lot more time than the other two algorithms. Hence the final execution time is driven by the node executing the longest random forest analysis. It goes without saying that each node takes a variable time in executing the computation assigned to it. Refer diagram 5.19

Recursive Learning of Predictions(Ensemble Learning Method 2)

In this method of Ensemble Learning, as explained earlier 15 models are generated in parallel. These models are then used to make predictions on the same data. These predictions are then column binded together with the actual value to make a 16 column level 2 training dataset. 15 of the Level 2 models are generated with level 2 training

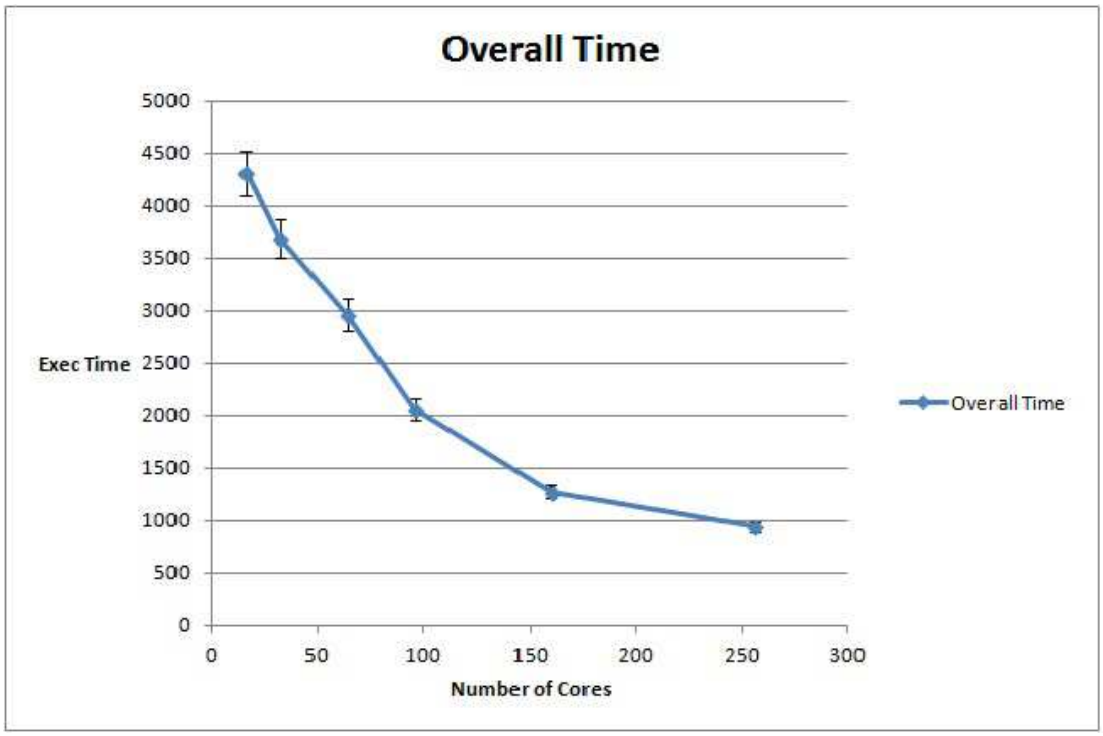

Figure 5.19: Averaging by Learning implementation of KDD dataset

dataset. Predictions are made using these level 2 models on the same dataset. This procedure is repeated to create 3 levels of training datasets, model and predictions. The graph 5.20 shows the execution time vs number of cores of generating all these models. Individual graphs are named as shown. All the series shows that the execution time is inversely proportional to the number of cores. The above use cases provide very vital and useful information about the algorithms from the scaling perspective. It also shows the scalability in simple ensemble learning. The following chapter draws some conclusions and future steps.

5.2.4 Key points of observation

Based on the above experimentation, following key points were drawn/observed:

- The Statistical Language and Tool "R" provides better flexibility and performance than Mahout for scaling up the algorithms on a small dataset (Few MegaBytes).
- Mahout, which uses Hadoop as the tool for scaling the algorithms, does not provide scalability at the core level granularity.

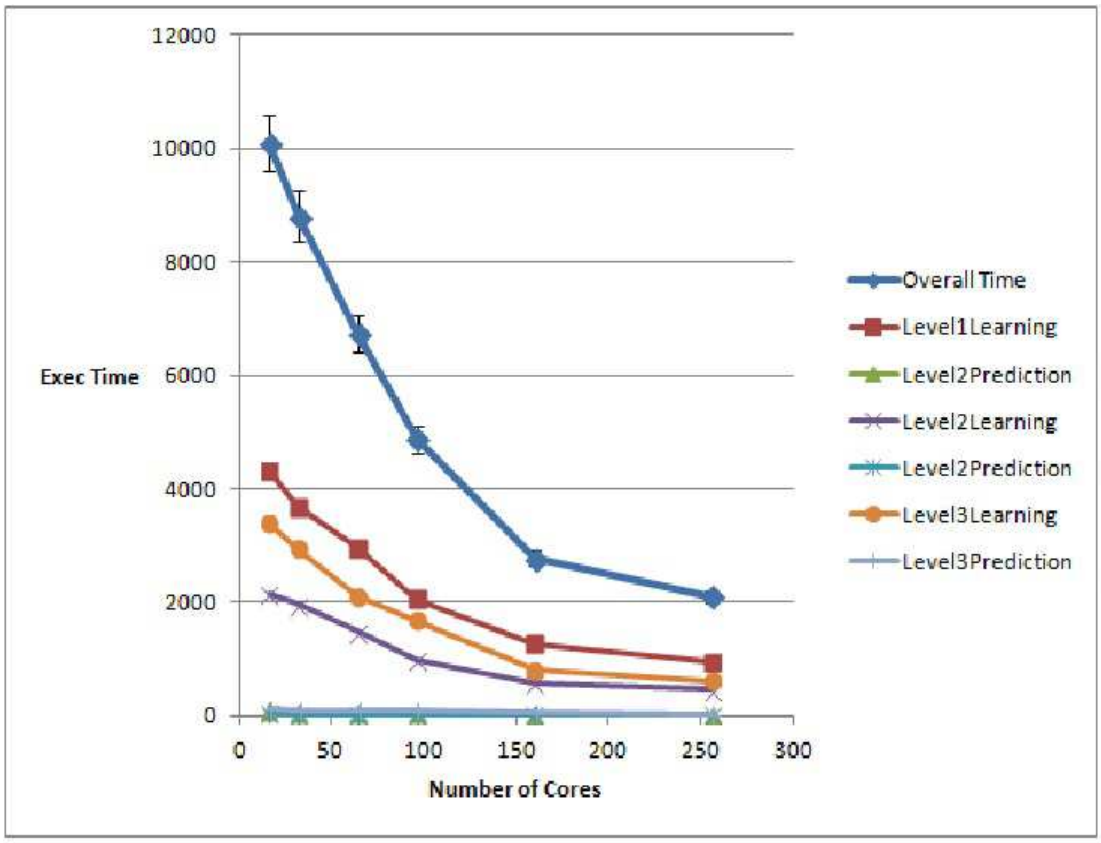

Figure 5.20: Recursive Learning of Prediction implementation of KDD dataset

- The data size impacts the execution time and the accuracy to a great extent.
- Lower the data size of each parallel chunk, overhead of communication time between nodes dominates in the over all execution time. Also the models generated will not have enough combinations of descriptors to define each output to the fullest.
- Though the packages in R (snowfall , do MC) provide good scalability results, the performance flatens to the end showing limitation in the execution time gain.
- Data and intermediate model vectors are stored in local memory in R. This limits the size of data with which we can work. With the integration of new packages which supports data on database and feed files for R , this limitation can be easily overcome.
- The split size variable of mahout plays an important role as it decides the size of parallelizable data chunks which are worked upon in parallel.
- The two new systems of ensemble learning provides great flexibility in modeling machine learning algorithms with different parameters.
- The two new system can be applied to varied set of datasets from various backgrounds and similar results both in terms of accuarcy and execution time can be expected.
- These systems, with minor tweaking, can potentially be scaled to 1000+ cores without compromising the accuracy.

Chapter 6

Conclusion

6.1 Thesis Summary

In this thesis work we have presented a distributed system for data analysis, particularly data classification and regression using ensemble learning and compared two platforms for the same. It was shown that distributing the computation involved in data classification and regression (machine learning) reduced the execution time to a great extent and showed its scaling stability to hundreds of cores. A detailed analysis of the packages used was also given, and how individual algorithm behaves to such a distributed setup. A presentation of evaluations which show the advantages of one platform over another for distributed data analysis was demonstrated. In addition this work validates the improvement in the execution time through two use-case analysis from datasets from two different domains.

A quick summary of the approach to parallelize individual algorithms and also a brief summary of our algorithm and its scaling performance can be listed in the following points:

- Random Forest: is based on the generation of many decision trees and the mode of all the decisions is taken as the final prediction. Hence these trees are independent and can be generated in parallel. Generating decision trees for a single model are done in parallel across multiple cores/nodes.

- Gradient Boosting Machines: is the method of machine learning where a multiple weak models are combined together to form a strong prediction model. Each model is dependent on its preceding model and its discrepancies. Hence generating decision trees in parallel was not a viable option. Hence multiple models were

created on a range of cores/nodes to observe the scalability.

- Generalized Linear Modeling-Lasso and Ridge Regression(GLMNET): Is the modeling where a balance between Lasso and Ridge regression has to be maintained to get the best prediction with respect to the dataset. The scaling is done on the number of models basis where many models are generated over a range of cores/nodes.

- Averaging by Learning(Ensemble Learning Method 1): in this method 15 unique models (5 each of the above algorithms)were generated, differentiated by the parameters, in parallel. These models were then used to make predictions which are later row-binded together with the predictors. This dataset is again learnt giving it an average of all the predictions.

- Recursive Learning of Prediction(Ensemble Learning Method 2): In this method we generate 15 unique models as in the previous method but the predictions are made on the training data itself. These predictions are then column-binded together with the real value and 15 new models are generated with this new dataset. This procedure is repeated 3 times and all the intermediate models are stored to get the final prediction on the test data.

The standard packages were used to implement the algorithms and also to scale the algorithms. Scaling was done both on individual core and node level using different packages at each level. Using the experiments the stability and usefulness of these packages were demonstrated.

The scalability of a particular algorithm Random Forest was analysed on two different platforms. Looking at the results conclusion could be drawn that the Java implementation on Mahout which uses the MapReduce paradigm and hadoop as its backend performs well when the split size is used intellligently. Keeping this constant flattens the strong scalability graph showing the limitation of scaling the implementation grater than a particular number of nodes. At the same time increasing the number of chunks by varying the splitsize helps in scaling but the accuracy gets comprimised as there is smaller data for each of the tree generation. Hence R provided better flexibility in its

implementation. But one major difference in the two models was that, mahout took the final result (class to be predicted) as a categorical value where as R took it as a numeric value. Hence the data analysis was data classification for Mahout where as it was data regression for R. Mahout used the hadoop data file system to store the datafiles and all the nodes accessed the same when implementing the computation. In R the data has to be exported to each node where the computation has to be done. Thus there exists a limit on the data size which can be used with R. But R provides standard packages for all the algorithms implementation which is used in this work as compared to Mahout where the Gradient Boosting Machines and Lasso and Ridge Regression implementation is not yet developed.

Key contributions of this work are 1.) A scalable data analysis tool with unique way of ensemble learning which scales to 200+ cores. 2.) A detailed analysis of the tools for 2 use cases with datasets from Medical and Social Network domain. 3.) A comparative analysis of two platforms Mahout and R for scalability and evaluation of its performance. Finally, our algorithm provides a foundation for further research in big data analytics and high scalability for machine learning.

6.2 Observations and Future Work

This research can be extended in many areas. It shows a scalability of a maximum of 256 cores. It can be extended for higher scalability of more number of cores. A potential extension could be its implementation on a huge cluster such as the Lonestar cluster at TACC. Lonestar supports 1000+ cores and it would be an interesting research to analyze the algorithms behavior on such a large cluster. That would indirectly mean experimentation could be done with more number of unique models which would constitute our ensemble for machine learning. The concentration in this work was mainly on the execution time and its variation with the number of cores or nodes. A potential research could also be towards measuring the accuracy and its variation when the computation gets distributed. This system can be tested on datasets from many different backgrounds and it can be molded appropriately to obtain maximum accuracy. A limitation of this system and its implementation was in the memory and the size of

the dataset we used. The training dataset was practically limited to the RAM of the system where the training was done. In the coming years storage and usage of data will reach an all new level and it will be imperative to look into this for future research. Some work is already being done in this area where packages for R are being written to access data directly from files or from the database. Another direction which can be explored is the inclusion of domain knowledge. Domain knowledge can be included in many ways - it could be in the way dataset is being processed, in the way the decision trees selects the descriptor while generating the tree, giving importance to certain set of descriptors. Domain knowledge will be very specific to the dataset being worked on. Finally, more complex and reliable ensemble learning can be tested with a lot more models without compromising the execution time to generate them.

References

[1] R. Analytics. Foreach looping construct for r. 1.4.0, Rev 22, 2012.

[2] R. Analytics. Foreach parallel adaptor for the multicore package. 1.3.0, Rev 12, 2012.

[3] S. Arthur. Some studies in machine learning using the game of checkers. IBM Journal, 2011.

[4] R. Bekkerman, J. Langford, M. Bilenko, and eds. Distributed machine learning.

[5] L. Breiman. Bagging predictors. 1994.

[6] L. Breiman. Random forests. 2001.

[7] J. Bruck, D. Dolev, C.-T. Ho, and R. Strong. Efficient message passing interface(mpi) for parallel computing on clusters of workstations.

[8] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 281–288. MIT Press, Cambridge, MA, 2007.

[9] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[10] T. G. Dietterich. Ensemble methods in machine learning. In *Proceedings of the First International Workshop on Multiple Classifier Systems*, MCS '00, pages 1–15, London, UK, UK, 2000. Springer-Verlag.

[11] J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. 33, Issue 1, 2010.

[12] J. H. Friedman. Stochastic gradient boosting. 1999.

[13] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: a review. *SIGMOD Rec.*, 34(2):18–26, June 2005.

[14] M. J. Kearns. The computational complexity of machine learning. MIT Press, Cambridge, MA, 2008.

[15] J. Knaus. Easier cluster computing (based on snow). 1.84-4, 2012.

[16] A. Liaw and M. Wiener. Classification and regression by randomforest. 2002.

[17] C. J. W. LUDMILA I. KUNCHEVA. Measures of diversity in classier ensembles and their relationship with the ensemble accuracy. abs/0911.0460, 2003.

[18] P. Ozer. Data mining algorithms for classification. 2008.

[19] B. Panda, J. Herbach, S. Basu, and R. J. Bayardo. Planet: Massively parallel learning of tree ensembles with mapreduce. *PVLDB*, 2(2):1426–1437, 2009.

[20] F. Provost and V. Kolluri. A survey of methods for scaling up inductive algorithms. *Data Min. Knowl. Discov.*, 3(2):131–169, June 1999.

[21] G. Ridgeway. Generalized boosted regression models. 2.0-8, 2012.

[22] I. Shamshurin. Data representation in machine learning-based sentiment analysis of customer reviews. In *Pattern Recognition and Machine Intelligence*, KDD '01, pages 254–260, 2011.

[23] J. Sill, G. Takács, L. Mackey, and D. Lin. Feature-weighted linear stacking. *CoRR*, abs/0911.0460, 2009.

[24] A. Skjellum and T. P. McMahon. Interoperability of message-passing interface (mpi) implementations: A position paper. 1997.

[25] P. Sollich and A. Krogh. Learning with ensembles: How overfitting can be useful, advances in neural information processing systems. Vol 8, 1996.

[26] W. N. Street and Y. Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '01, pages 377–382, New York, NY, USA, 2001. ACM.

[27] L. Tierney, N. L. A. J. Rossini, and H. Sevcikova. Simple network of workstations. 0.3-10, 2012.

[28] S. Urbanek. Parallel processing of r code on machines with multiple cores or cpus. 0.1-7, 2011.

[29] G. Valentini and F. Masulli. Ensembles of learning machines.

[30] G. M. Weiss. Mining with rarity: a unifying framework. *SIGKDD Explor. Newsl.*, 6(1):7–19, June 2004.

[31] S. Weston. Getting started with domc and foreach. 2012.

[32] H. Yu. Interface (wrapper) to mpi (message-passing interface). 0.6-1, 2012.