Why Naive Ensembles Do Not Work in Cloud Computing

Wenxuan Gao, Robert Grossman, Yunhong Gu and Philip S. Yu
Department of Computer Science
University of Illinois at Chicago
Chicago, USA
wgao5@uic.edu, grossman@uic.edu, gu@lac.uic.edu , psyu@cs.uic.edu,

Abstract. **Cloud computing systems that use are designed to scale to large numbers of loosely coupled commodity computers (or nodes) are growing more common. In this paper, we consider several different ways that tree-based classifiers can be used on cloud computing systems. The simplest way is simply to compute one or more trees independently on each node and then to combine these to create an ensemble. In a computer cluster containing 1000 nodes, computing 10 trees per node generates a 10,000 tree ensemble that may over fit. In this paper, we consider alternative approaches using random trees.**

Keywords: cloud computing; ensembles of classifiers; random trees; classification and regression trees

I. INTRODUCTION

Recently, cloud computing systems have been developed for processing very large datasets [18, 19, 20]. Often they include a software framework that provides a simple programming interface for large data processing using clusters of commodity computers. Hadoop [21] is the most common such system. Sector/Sphere [1] is another such system.

Given a cluster of computers, it is common to compute one or more trees on each node in the cluster (this can be done independently), to combine these into an ensemble, and then to score data (i.e. apply this ensemble to make predictions) using the ensemble. For small to medium size clusters, this is an extremely effective technique. On the other hand, the computer clusters used in cloud computing can contain a thousand or more computers, which would produce an ensemble with 10,000 trees (if 10 trees per node in the cluster are computed) or 100,000 trees (if 100 trees per node are computed). Even for very large datasets, ensembles of this size can over fit the data.

In this paper, we consider two alternative approaches.

Both approaches make use of random trees [2]. In the first approach called Top-k, each compute node in the cloud builds a random tree using its local data. These trees are then evaluated on a common dataset and the k trees with the lowest error are used to form an ensemble.

In the second approach called Skeleton, a common skeleton that is built from the features but without training data is distributed to each of the compute nodes in the cloud. Each compute node then uses its local data to define a classification tree. All the local classification trees are then returned to the central node that merges them into a single tree (which can be done since all the trees share the same interior node structure). To build an ensemble, multiple skeletons are distributed.

In this paper, we introduce these two methods and perform several experimental studies to understand their cost, expected accuracy, and other properties. As we will see below, the Skeleton Algorithm is an effective method for efficiently computing ensembles on even very large clouds.

II. BACKGROUND

In this section, we give a brief introduction to the random decision tree algorithm and to the Sector/Sphere system.

A. *Random Decision Tree Algorithm*

The random decision tree was first presented by Wei Fan et. al. [2]. Recall that the basic algorithm to build a decision tree uses a portion of the data (the training dataset) and at each stage selects the feature and cut value that maximizes the information gain. In contrast, a random decision tree is built in two stages. First, independently of the training data, a feature and cut value is randomly selected. The resulting structure is

sometimes called a *tree skeleton*. This is repeated until the tree reaches the specified size. Often, there is a requirement that at each stage a new (previously unused) feature is selected. Second, the training data is used to specify the appropriate classifications (for a classification tree) or values (for a regression tree). In general, multiple tree skeletons are computed and scoring (i.e. prediction) is done by combining the different trees into an ensemble as usual.

B. *Sector/Sphere*

Sector is a distributed storage system that is designed to operate over large numbers of commodity computers [1]. Sphere is a parallel computing framework that can access data stored by Sector [1].

Sector/Sphere is broadly similar to the Google File System/MapReduce framework [19, 18].

Sphere can execute User Defined Functions (UDF) over Sector-managed data. For example, for example a user-defined Map UDF, followed by a system-supplied Shuffle UDF, a system-supplied Sort UDF, and a user-defined Reduce UDF can be used to provide the functionality of the MapReduce parallel computing framework [18]. Sector is open source and available through Source Forge.

In this work, we use Sector/Sphere to implement the parallel random decision tree.

III. RELATED WORK

There is a very large literature on classification and regression trees and it is not practical to review it here. Broadly speaking, there are several approaches to scaling classification and regression trees to large data sets.

Sampling. Data sampling is often used to reduce the size of data sets [9, 10]. With this approach, there is the assumption that adequate tree models can be built using a portion of the data. One of the challenges with this approach is that large data sets often consist of several different populations. In this case, the data must be first explored to determine the various populations and then stratified sampling must be used to extract representative samples from each of the different populations.

Ensemble-based methods. One of the most popular architectures for managing and analyzing large data sets are clusters of commodity computers. Ensembles of models have been used frequently in practice for building models over computer clusters consisting of loosely couple distributed systems, beginning at least as far back as 1995 [27]. Essentially the same approach works over geographically distributed systems [26]. Although there is a fairly large literature on methods for improving the accuracy of ensemble-based models, in practice, it can be challenging to find a method that is broadly applicable and consistently outperforms simple voting. Bagging and boosting are two popular ensemble methods that can help improve accuracy, but are usually used for small datasets that fit into memory. An approach for using boosting and bagging on large data sets in a distributed environment is described in [25].

Task-level parallelism. Several parallel algorithms have been developed to compute classification and regression trees using multiple processors (that typically communicate using a message passing protocols such as MPI) have been developed, both when the data is assumed to be in memory and when it is on distributed disks [12, 4, 8, 16].

Simplified parallel processing frameworks. In practice, it can be labor intensive to develop and code using MPI task-parallel versions of algorithms. Recently, frameworks that use data parallelism and provide relatively simple primitives for building parallel algorithms have been introduced. The best known such framework is the MapReduce model introduced by Google [18]. The Sphere model described above is another example of a simplified model for coding data parallel algorithms.

IV. PARALLEL RANDOM DECISION TREE

In this section, we describe two different approaches for building random decision trees in parallel over large clusters of loosely coupled commodity computers: the Skeleton Algorithm and Top-k Algorithm. See Table 2. Recall that for large clouds, ensembles that contain trees from each node in the cloud can contain too many trees to be effective. One of the goals of both methods is to reduce the number of trees in the ensemble.

Train: (S, X, N)
Data: S = $\{(x_1, t_1), \ldots, (x_n,t_n)\}$ is a training set and X = $\{F_1, \ldots, F_k\}$ is a set of features, where F is a feature descriptor. N is the number of random decision trees.

Result: N random trees $\{T_1, \ldots, T_n\}$.
begin
  for i ∈{1,…N} **do**
    BuildTreeStructure (T_i, x)
  end
  for (x,t) ∈ S **do**
    for i ∈{1,…N}
      UpdateStatistics (T_i, (x,t))
    end
  end
  return $\{T_1, \ldots, T_n\}$
end

Algorithm 1. Random Decision Tree

A. *System Architecture*

Before introducing the Top-k and Skeleton methods, we briefly describe the distributed system architecture that both methods assume

As Figure 1 shows, a central (or master) node maintains the metadata of the files stored in the system, and controls the running of all local (or slave/compute) nodes. The local nodes store the files managed by the system and process the data upon the request of the central node. The local nodes are usually organized into racks of computers that are physically located in one or more data centers. There is usually a network switch at the top of each rack that supports the communication within the rack. Multiple racks are then connected to a larger network switch that supports the communication between racks.

B. *The Random Decision Tree Algorithm*

Algorithm 1 is a simple summary of the training process of the random decision tree algorithm. The algorithm consists of two steps. The first step (BuildTreeStructure) generates the structure of each random tree. This is referred to as the skeleton since the leaves do not contain class distribution statistics. Building the skeleton does not require any training data, but instead only requires information about the features. In the second step (called UpdateStatistics), the training data is used to compute class statistics for the leaf nodes. Below we briefly describe the BuildTreeStructure step and UpdateStatistics step omitting implementation details [2].

The BuildTreeStructure step builds the trees by randomly choosing a feature for each internal node of the tree. Discrete features can only be chosen once, but continuous features can be chosen multiple times. As mentioned, this stage does not require training data. In particular, in the distributed case, it is not necessary to move any training data in this step.

The UpdateStatistics step is used to update the class distribution information each time a new training instance is read. Each leaf records for each class the number of instances that are "classified" through that node. Note that in the distributed case, each computing node can complete the UpdateStatistics step independently.

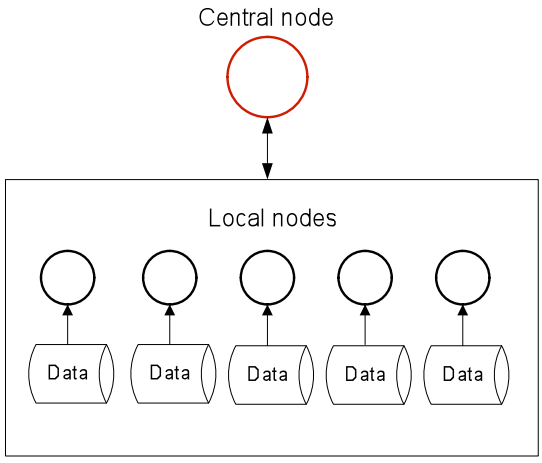


Figure 1. System Architecture

C. *Top-k Algorithm*

In the Top-k algorithm, each local node builds a random tree using local data. In this method, the

central node distributes a common dataset to each local node. Each local node then evaluates its local tree on this dataset and computes the error. Each local tree then returns to the central node the local tree and the corresponding error. In the final step, the central node forms an ensemble using the k random trees with the lowest error. The remaining trees are discarded.

The common dataset that is distributed to each local node is randomly selected from all the local nodes and thus represents (a sample of) the entire dataset.

A limitation of this method is that the k trees selected only represent part of the entire dataset – the part associated with the data on the corresponding k nodes – and this can reduce the accuracy of the method. The accuracy of this method improves if prior to building the trees some of the data from each of the nodes is scattered to the other nodes in the cloud. Sometimes this is called a shuffle. This is easy to implement with the Map portion of MapReduce. It is also easy to implement in Sphere. Perhaps surprisingly, as we will see below, in many cases a relatively small percentage of the entire dataset needs to be shuffled in this way in order to improve the accuracy significantly.

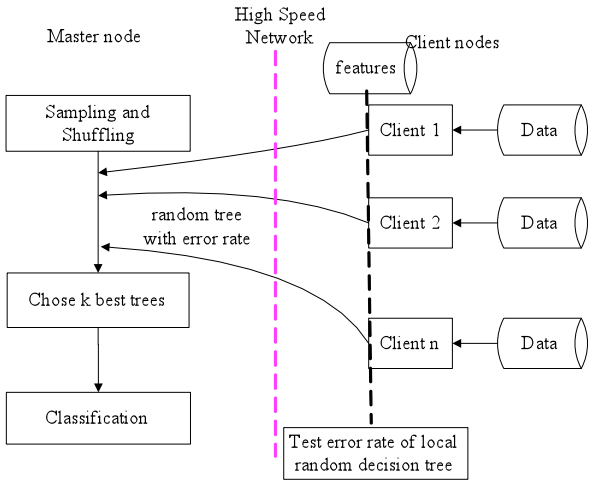


Figure 2. Top-k approach to parallel RDT

D. *Skeleton Algorithm*

In the skeleton algorithm, the central node builds the skeleton of a random decision and distributes this skeleton to all the local nodes. Recall that constructing a skeleton does not require any data except simply knowledge of the features. Each local node then scans its local data and uses the skeleton to build an actual tree. Each local tree is then returned to the central node. Since each local tree has the same topology it is straightforward to combine the statistics from each leaf node and to construct a single tree that reflects the entire dataset. See Figure 4. Of course, an ensemble can be built by simply scattering multiple skeleton trees to each local node.

Another alternative is to partition the local trees and to build a merged tree from trees in each of the partitions. In this way, an ensemble is produced with each tree in the ensemble reflecting data from the local trees in its associated partition.

Note that accuracy of the tree produced by the Skeleton Algorithm on a cloud is the same as the accuracy of a random tree built on the entire dataset, but much faster since the local trees can all be computed independently on each local node in the cloud.

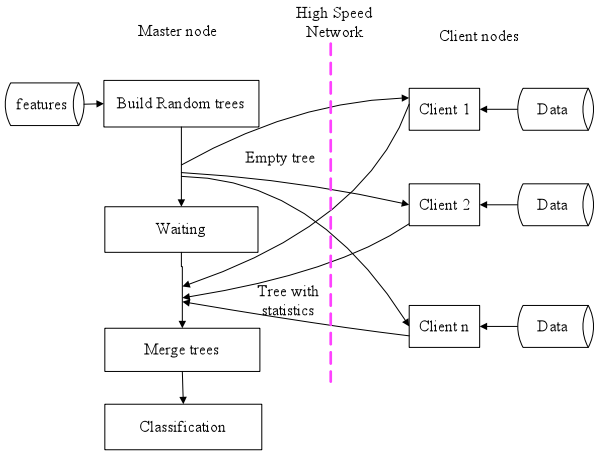


Figure 3. The Skeleton algorithm

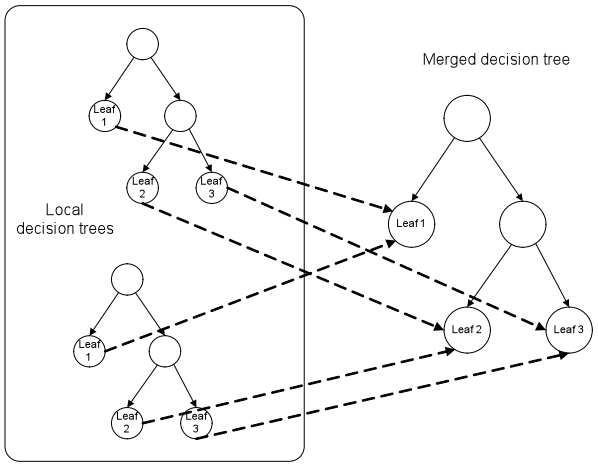


Figure 4. How trees in the skeleton algorithm are mergerd.

Since merging trees on the central node in the Skeleton Algorithm is more expensive than simply selecting the top k trees, the Skeleton Algorithm is more expensive to build than the Top k Algorithm. On the other hand, as we will see in the next section, it is usually more accurate.

V. Experiments Setup

In this section we describe several experimental studies we performed comparing the Top-k Algorithm and the Skeleton Algorithm. The experiments used two public datasets on a cloud consisting of four racks and evaluated the accuracy and execution time of the two algorithms as the number of trees, the number of nodes, and the proportion of data shuffled among the nodes varied.

A. Datasets

For these experiments, we used two datasets from the UCI KDD Archive. Since these datasets were small relative to our cloud, we replicated them for our experiments as described below. The first dataset was the 1999 KDD Cup dataset. We divided this data into training and test datasets. The raw training data was about 4 GB of compressed binary TCP dump data from seven weeks of network traffic.

Table 1. Data sets used (sizes are before data was replicated).

Datasets	**Training instances**	**Test instances**	**Features**
1999 KDD Cup	4,898,431 (742M)	311,029 (47M)	41
Census Income	199,523 (101M)	99,762 (50M)	40

We aggregated the packets into about five million records describing network flows. Similarly, the two weeks of test data yielded around two million flow records.

Our experiments were based on those flow records. The training dataset has 494,021 records with 22 attack types and the test dataset has 311,029 records with 37 attack types. The prediction task was to identify those 37 attack types in the test dataset.

We divided the original training dataset into 26 partitions determined by the protocol field in the record and assigned each partition to a node. In other words, we distributed the data over 26 nodes in a single rack and the data on each node had the same protocol field. We proceeded in this way in order to produce a dataset in which there were significant differences between the various local nodes. To increase the size of the dataset, we replicated the data on each node 100 times. After the replication, there was about 3 GB of data on each node. We then copied the data on the one rack to the other three racks. The final dataset was about 312 GB in size and had the same class distribution as the original dataset.

The second data set we used was a census income data set. The data set contained weighted census data extracted from the 1994 and 1995 Current Population Surveys conducted by the U.S. Census Bureau. The prediction task was to determine whether a person makes over $50,000 a year. The original data contained 40 demographic and employment related features.

We used a subset of records that was extracted by Barry Becker. Again, for the purpose of making the distributed dataset heterogeneous, we partitioned the training dataset into 26 subsets according to each individual's type of work and distributed the records over 26 nodes within a single rack. This time, we replicated the data on each node in the rack 10,000 times to generate about a 1 GB of data per node. As before, we then copied the dataset to the other three racks.

B. *Computing Environment*

The experimental studies were done using four racks on the Open Cloud Testbed [17]. Each rack has 32 nodes, including 1 NFS server, 1 head node, and 30 local or slave nodes. The head node is a Dell 1950, dual dual-core Xeon 3.0GHz with 16GB of RAM. The slave nodes are Dell 1435s, single dual core AMD Opteron 2.0 GHz with 4 GB of RAM and a single 1 TB disk. The 4 racks are located in JHU (Baltimore), StarLight (Chicago), UIC (Chicago), and Calit2 (San Diego). A wide area 10 Gbps network connected the four racks. For our experiments, we used 26 of the 32 nodes on each rack.

For the experiments, we used version 1.23 of Sector/Sphere.

C. *Evaluation Metrics*

We use the computation cost and error rate to evaluate the performance of the two algorithms. The computation cost is defined as the total training time required to build the ensemble of trees. For the Top-k Algorithm, the time to sample and shuffle the sample data is taken into account. The error rate is defined as the number of misclassified records divided by the total number of records.

VI. RESULT AND ANALYSIS

In this section, we report the computation cost and error rate for the Top-k and Skeleton algorithms as we vary the shuffle rate, the number of nodes in the cloud, and the number of trees in the ensemble. Since different random decision tree vary in accuracy, we ran each test 10 times and plotted the means of the 10 runs.

A. *Impact of Shuffle Rate*

In the first set of experiments, we vary the amount of data shuffled in the Top-k Algorithm and examine the impact on the accuracy and cost.

Accuracy. For the experiments measuring accuracy, we did not replicate the data in the 26 partitioned datasets. We used an ensemble of 10 trees. We first computed the error rate without any shuffling using the Top-k and the Skeleton Algorithms. Without any shuffling, the Top-k Algorithm has a high error rate for the both the 1999 KDD Cup dataset and the census income dataset. In contrast, the error rate of the Skeleton Algorithm is fairly low.

We shuffled the data by taking p% of the data from each local node and distributing it to all the other nodes. In this way, each node has a small amount of data from all the other nodes. We vary p and repeated the experiments 10 times. The results are plotted in Fig. 5 and Fig. 6.

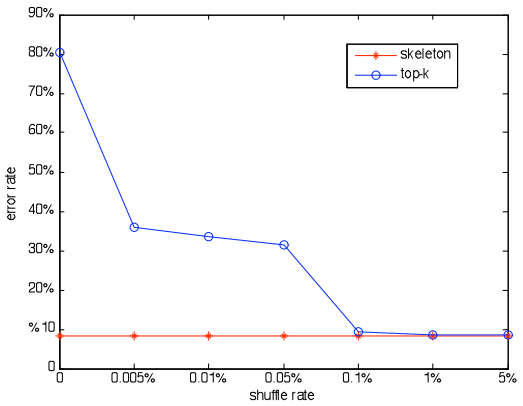


Figure 5. Error rate with KDDCup99 dataset by varying the shuffle rate

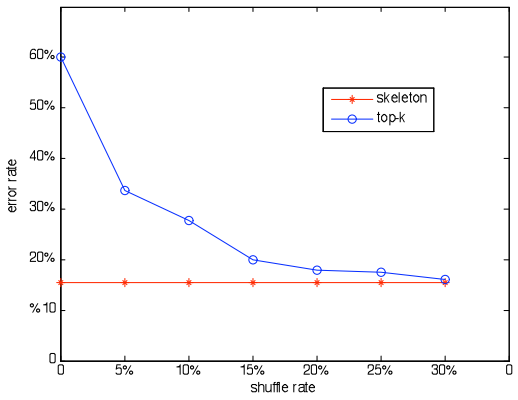


Figure 6. Error rate with census income dataset by varying the shuffle rate

Our experiments show that as the shuffle rate p increases, the error rate drops gradually. This is reasonable because as the shuffle rate increases, the data on each local node more closely resembles the entire dataset

For the 1999 KDD Cup dataset, when p is above 0.1%, the Top-k Algorithm achieves essentially the same accuracy as the Skeleton Algorithm. On the other hand, for the census dataset, a shuffling rate of 20% is required to achieve the same accuracy. Since the Skeleton Algorithm builds a tree using the entire dataset, no no benefit is obtained by shuffling.

Computation Cost. For the experimental studies involving the computational cost of the two algorithms, we replicated the data as described above: 100 times for the 1999 KDD Cup dataset and 10,000 times for the census dataset. The results are plotted in Figures 7 and 8. These figures also show the cost of the shuffle.

As expected, the computation cost of the Top-K Algorithm increases significantly as the shuffle rate grows.

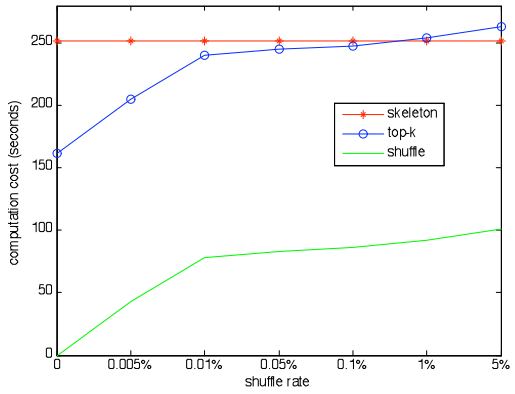


Figure 7. Computation cost for 1999 KDD Cup dataset as the shuffle rates varies.

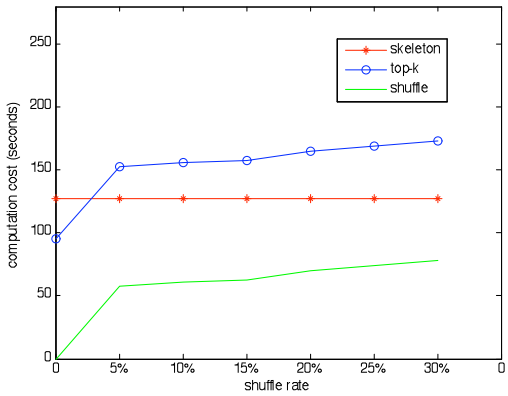


Figure 8. Computation cost for the census income dataset as the shuffle rates varies.

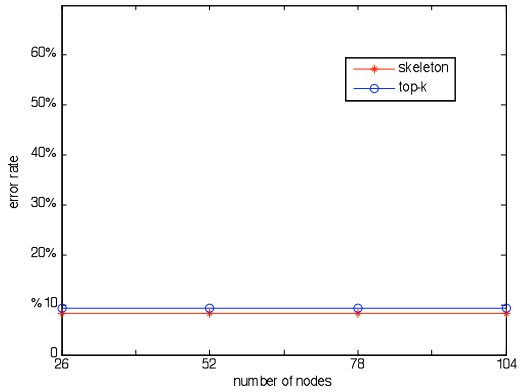


Figure 9. Error rate with KDDCup99 dataset by varying the number of nodes

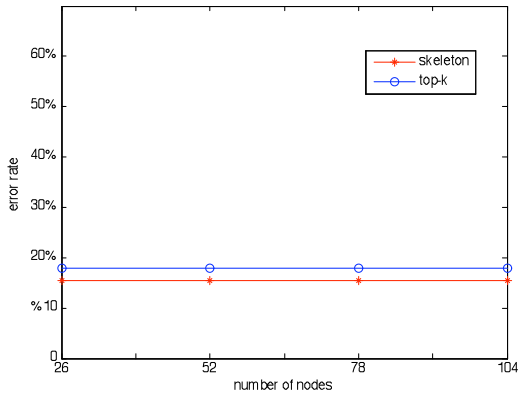


Figure 10. Error rate with census income dataset by varying the number of nodes

To summarize this set of experiments: when shuffling just 0.1% of the 1999 KDD Cup dataset, the Top-k Algorithm achieves almost the same accuracy as the Skeleton Algorithm while taking significantly less time. On the other hand, for the census dataset that requires shuffling 20% of the data to achieve the same accuracy as the Skeleton Algorithm, the Top-K Algorithm takes significantly longer.

B. *Impact of the Size of the Cloud*

In the second series of experiments, we compare the accuracy and cost of the two algorithms as the number of local nodes in the cloud increases.

Accuracy. For these experiments, we increased the size of the cloud from 1 to 4 racks. In these experiments, the data was replicated as described above. We fixed the shuffle rate at 0.1% for the 1999 KDD Cup dataset and at 20% for the census dataset. As before, we used an ensemble of 10 trees. The results are plotted in Figures 9 and 10.

From the figures, we see that as expected the accuracy stays almost the same as the size of the cloud increases from 1 to 4 racks. This is because simply replicating data from 1 rack to 4 racks does not change the statistical characteristics of the data.

Computation Cost. We next examined the impact of the increasing the size of the cloud on the computational cost of the two algorithms. The data was replicated as before. The results are plotted in Figures 11 and 12.

The figures show that the computation cost of both the Skeleton Algorithm and the Top-k Algorithm grow slowly as the number of racks increases. This is mainly due to the increase in communication overhead.

To summarize this set of experiments, there is very little impact on the cost or accuracy of the computation for either algorithm as the size of the cloud increases.

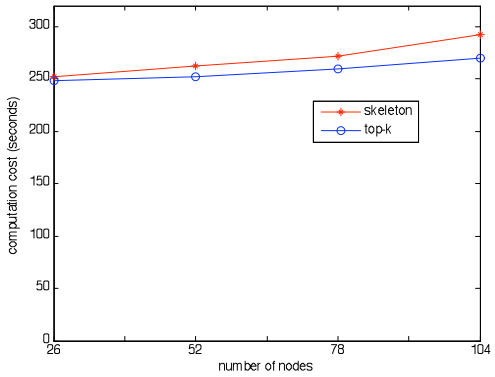
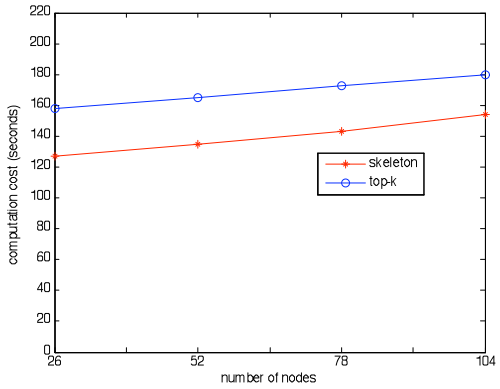


Figure 11. Computation cost for 1999 KDD Cup dataset as the size of the cloud increases.



Computation cost with cencus income dataset as the size of the cloud increases.

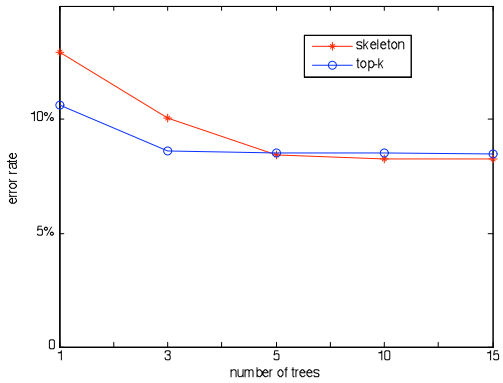


Figure 12. Error rate with 1999 KDD Cup dataset as the number of tree varies.

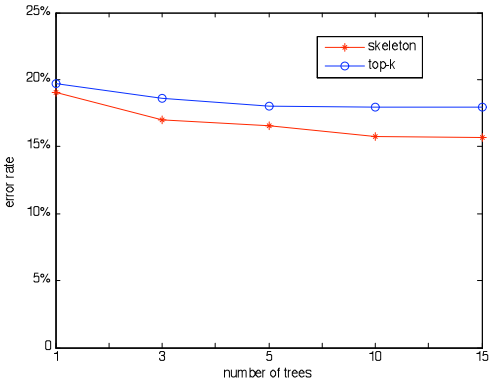


Figure 13. Error rate of the census income dataset as the number of tree varies.

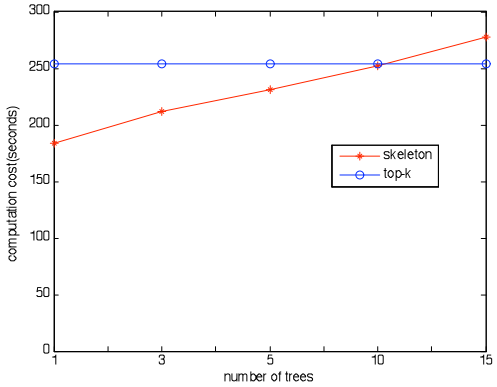


Figure 14. Computation cost 1999 KDD Cup dataset as the number of tree varies.

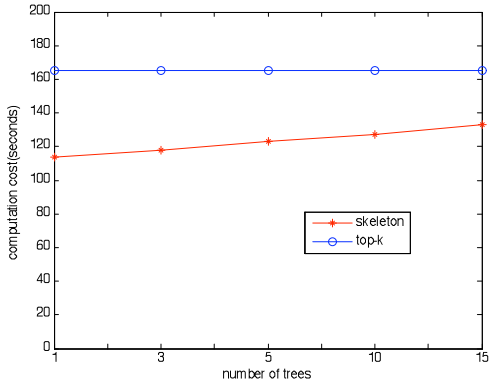


Figure 15. Computation cost with census income dataset as the number of tree varies.

C. *Impact of Number of Trees*

In [2], it is found that, in most situations, there is little gain in accuracy when more than 10 random trees are used for classification. In the third series of experiments we tried to verify this rule of thumb.

Accuracy. The settings in this series of experiments are the same as used for the accuracy

experiments described in section A of this section, except that we fixed the shuffle rate at 1% for the 1999 KDD Cup dataset and 20% for the census income dataset. The results are plotted in Figures 13 and 14.

Our results confirm that using more than 10 random trees provides very little improvement in accuracy. In Figure 13, we can see when the number of trees is less than 5, the Top-k Algorithm is better than the Skeleton approach. We argue that because the top-k approach always takes the best trees to do the prediction.

Computation cost. The settings in this series of experiments are the same as used for the computational costs experiments described in section A of this section, except that we fixed the shuffle rate at 1% for the 1999 KDD Cup dataset and 20% for the census income dataset. The results are plotted in Figures 15 and Fig. 16.

The results show that generally the training cost grows with the number of trees for the Skeleton Algorithm, but is relatively constant for the Top-k Algorithm. That is because only one random decision tree is built on each local node for the Top-k Algorithm, no matter how many trees are finally used in the ensemble.

To summarize this series of experiments, for both the Skeleton Algorithm and the Top-k Algorithm, increasing the number of trees to more than 10 does not significantly improve the accuracy. Also, as expected, the computation cost of the Skeleton Algorithm grows as the number of trees increases, whereas the Top-k Algorithm does not.

VII. CONCLUSIONS

As the size of cloud computing systems grow, building naïve ensembles by computing one tree for each local node in the cloud produces ensembles with so many trees that the ensembles are not effective classifiers. In this paper, we proposed and implemented two competing approaches: the Top-K Algorithm and the Skeleton Algorithm. We performed several experimental studies evaluating the accuracy and cost of these two algorithms as we varied the size of the cloud, the number of trees in the ensemble, and the amount of data shuffled prior to building the trees.

Our experiments show that the Skeleton Algorithm creates an ensemble that provides a consistently accurate prediction. For some datasets, the Top-k algorithm can produce an ensemble with comparable accuracy that is less expensive to compute, whereas in other cases, the Top-k Algorithm is more expensive to compute. The difference is due to the percent p of data that must be shuffled to achieve accuracy comparable to the Skeleton Algorithm. If p is known and small, the Top-k algorithm is preferable since it is less expensive. In all other cases, the Skeleton Algorithm is recommended.

	Top-k	**Skeleton**
What is distributed to the local nodes?	A common dataset is scattered to local nodes, which is selected from all the (distributed) data	A skeleton tree.
How is the tree computed on the local nodes?	Each local node builds a random tree independently. Different local nodes have trees with different topologies.	The central node builds skeleton that is distributed to all local nodes. Each local node uses its local data to compute a tree.
What is returned to central node?	Each local node returns a tree and its error on a common dataset.	Each local node returns a tree.
How are local trees combined?	Top-k trees with smallest error are selected to form an ensemble.	All trees are merged into single a tree. If an ensemble is desired, several skeletons can be scattered to local nodes.
Cost	Lower since just k trees with the lowest error need to be selected.	Higher since trees built from local data need to be merged.
Is the tree built on all or on part of the data?	On part of the data.	On all of the data.

Additional information	The data can be shuffled to improve the performance.	

Table 2. A comparison of the Top-k and Skeleton Algorithms.

REFERENCES

[1] Y. Gu and R. Grossman, "Sector and Sphere: The Design and Implementation of a High Performance Data Cloud", Phil. Trans. R. Soc. A 28 June 2009 vol. 367 no. 1897 2429-2445.

[2] W. Fan, H. Wang, P.S. Yu, and S. Ma, "Is random model better? on its accuracy and efficiency", Proc. 3rd IEEE Intl. Conf. on Data Mining (ICDM-2003), Nov 2003, pp.51-58.

[3] http://sector.sourceforge.net/

[4] M. Joshi, G. Karypis and V. Kumar, "ScalParC: A New Scalable and Efficient Parallel Classification Algorithm for Mining Large Datasets." Proc. 1998 Intl. Parallel Processing Symp., April 1998, pp.573-579.

[5] M.J. Zaki and C.-T. Ho (Eds.), Large-Scale Parallel Data Mining, LNAI State-of-the-Art Survey, Vol. 1759, Springer-Verlag: Berlin, 2000.

[6] H. Kargupta and P. Chan (Eds.), Advances in Distributed and Parallel Knowledge Discovery, AAAI Press/ MIT Press, 2000.

[7] P.S. Bradley,U. Fayyad and C. Reina, "Scaling Clustering Algorithms to Large Databases", Proc. 4th Intl. Conf. on Knowledge Discovery & Data Mining (KDD98), AAAI Press, 1998, pp9-15.

[8] M. Mehta, R. Agrawal, and J. Rissanen, "SLIQ: A Fast Scalable Classifier for Data Mining" In 5th Intl. Conference on Extending Database Technology, pp.18–32, 1996.

[9] R. Musick, J. Catlett, and S. Russell, "Decision theoretic subsampling for induction on large databases", In Proc. of 10th Intl. Conf. on Machine Learning, pp.212 – 219, Amherst, MA, 1993.

[10] F. Provost, D. Jensen and T. Oates, "Efficient Progressive Sampling", Proc. Fifth Intl. Conf. on Knowledge Discovery and Data Mining, pp.23-32, 1999

[11] C. Moretti, K. Steinhaeuser, D. Thain and N.V. Chawla, "Scaling up Classifiers to Cloud Computers", In 8th IEEE Intl. Conf. on Data Mining(ICDM08), pp.472-481, Dec 2008

[12] A. Srivastava, E. Han, V. Kumar, and V. Singh, "Parallel Formulations of Decision-Tree Classification Algorithms," Data Mining and Knowledge Discovery: An Intl. Journal, vol. 3, no. 3, September 1999, pp.237-261.

[13] E. Han, G. Karypis and V. Kumar, "Scalable Parallel Data Mining for Association Rules", IEEE Trans. on Knowledge and Data Engineering, Vol. 12, No. 3, May/June 2000, pp.372-390.

[14] V. Ganti, R. Ramakrishnan, and J. Gehrke, "Clustering Large Datasets in Arbitrary Metric Spaces", Proc. 15th Intl. Conf. on Data Engineering, p.502, March 23-26, 1999.

[15] R. Agrawaland J.C. Shafer, "Parallel Mining of Association Rules", IEEE Trans. On Knowledge and Data Eng., 8(6):962-969, December 1996.

[16] J. Shafer, R. Agrawal, and M. Mehta, "SPRINT: A Scalable Parallel Classifier for Data Mining", Proc. 22nd Int. Conf. On Very Large Databases, Morgan Kaufmann, 1996, pp.544-555

[17] http://www.opencloudconsortium.org/

[18] J. Dean and S. Ghemawat. "MapReduce: Simplied data processing on large clusters", In OSDI'04: 6th Symp. on Operating System Design and Implementation, 2004.

[19] S. Ghemawat, H. Gobioff, and Shun-TakLeung. "The Google File System", SOSP'03, October 19–22, 2003.

[20] Amazon Web Services, http://aws.amazon.com.

[21] A. Bialecki, M. Cafarella, D. Cutting, O. O'Malley. "Hadoop: a framework for running applications on large clusters built of commodity hardware", http://lucene.apache.org/hadoop/, 2005

[22] A.S. Das, M. Datar, A. Garg and S. Rajaram, "Google News Personalization: Scalable Online Collaborative Filtering", Proc. 16th Intl. Conf. on World Wide Web, 2007, pp.271-280.

[23] M.A. Bayir,I. H. Toroslu,A. Cosar and G. Fidan, "Smart Miner: a new framework for mining large scale web usage data", Proc. 18th Intl. Conf. on World Wide Web, 2009, pp.161-170

[24] S. Zhao, J. Betz, "Corroborate and learn facts from the web", Proc. 13th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, 2007, pp.995-1003.

[25] N. V. Chawla, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer, "Learning Ensembles from Bites: A Scalable and Accurate Approach", Journal of Machine Learning, 5:421–451, 2004.

[26] P. Chan, W. Fan, A. Prodromidis, and S. Stolfo, Distributed data mining in credit card fraud detection, IEEE Intelligent Systems, Volume 14, Number 6, pages 67-74, 1999.

[27] Robert L. Grossman, Haim Bodek, David Northcutt, and H. Vincent Poor, Data Mining and Tree-based Optimization, Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD 1996), E. Simoudis, J. Han and U. Fayyad, editors, AAAI Press, Menlo Park, California, 1996, pages 323-326.