Fault-Tolerant Distributed Random Forests

1.0 Background

Random Forests[1] is a classifier based on classification trees constructed using an algorithm developed by Leo Breiman and Adele Cutler. This classifier trains a 'forest' of decision trees and performs classification predictions by presenting an input vector to each of the trained trees in the 'forest', and taking the majority 'vote' as determined by the individual trees.

2.0 Motivation

As the cost of commodity PC hardware and network equipment drops, cluster computing becomes ever more popular. If previously sequential algorithms can be rewritten for distributed implementation, one can benefit from both reduced processing time, and the ability to accommodate extremely large working sets available today.

However, there are limitations to cluster computing. Network communication becomes a severe bottleneck, and there always exists the possibility of node failures within the cluster.

The Distributed Random Forests algorithm presented here specifically targets implementation in cluster environments and attempts to accommodate the inherent limitations and concerns of using cluster hardware by presenting an algorithm with sparse communication and fault tolerance.

[1] Random Forests (http://www.stat.berkeley.edu/users/breiman/RandomForests/)

3.0 Algorithm

Current implementations of Random Forests are sequential in nature. A forest of decision trees independently train on a training set, all running on a single processing element. Because each tree in the forest trains independently from all other trees, Random Forests is a strong candidate for parallelization.

A large cluster collectively has enough resources to accommodate most problems, but resources are distributed amongst many independent nodes. The data sets targeted for classification by algorithms such as Random Forests tend to be extremely large, and must be distributed amongst all the working nodes of the cluster.

In the distributed implementation of Random Forests, each cluster node is transferred an equal subset of the whole training data set from a source cluster node. Each computing node then trains a Random Forest cluster sub-forest on its training subset. After training, the Distributed Random Forest can accept vectors for classification. An input vector is presented to each node's cluster sub-forest, and each sub-forest returns with a vote. All votes are communicated to a result node that is responsible for tallying all incoming votes. The majority vote is the classification result.

While the training subset size can vary, each computing node only trains on the subset provided to it by the source cluster node. Computing nodes communicate only with the source node and the result node. Within this framework, the majority of communication occurs when transferring the training data subsets to the computing nodes, and thereafter all computing node communication is sparse, either receiving input vectors from a source node on the network, or sending its computed vote to the result node which then computes the final classification result.

There are several advantages to the distributed framework presented.

Fault-tolerance is a large concern when using commodity hardware in a cluster environment. Because computing nodes do not communicate amongst each other, no single computing node can become a source of catastrophic failure for the cluster in the case when a computing node either fails or becomes unable to communicate.

In the typical case, the result node will await responses from all computing nodes, but being built for fault-tolerance, the result node has a built-in time-out device. This time-out counter, reset by an input vector arrival, will, after a time-out period, abandon waiting for any nodes that have not yet returned their votes, and compute the majority vote with the tallied votes that have successfully arrived. This feature can be tuned such that it will abandon waiting for votes from nodes that have either likely failed, or may be subject to high latency network connections. In either case, the behavior of the result node can be tuned to suit the needs of the application.

Classification accuracy results presented will show that the loss of some fraction of computing node results will not critically impact prediction accuracy as long as sufficient nodes are in the cluster. Adjusting the subset size sent to each computing node is a possibility to accommodate for node failure within small cluster.

4.0 Methodology

A simulation of the Distributed Random Forests algorithm was implemented, along with a cross-validation framework. The simulation implemented sequentially performs the tasks of the source node, computing nodes, and result nodes to partition and distribute datasets, train sub-forests, vote, and classify.

All experimental results presented are obtained using a ten-fold cross-validation scheme

Parameters including training subset size, number of classifiers, and number of trees per classifier were varied to investigate their effects on classification performance.

Before dealing with subset size, we introduce the concept of "sample gain". Given a training set of size X, and a cluster with C computing nodes, a training subset of size X/C (i.e., the data set is equally distributed amongst all the computing nodes), refers to a sample gain of one; sample gain of G refers to a training subset of size G*X/C. Note that if sample gain is greater than one, training subsets will contain overlapping data.

Data partitioning is performed as follows:

1) the training dataset is first randomized, and
2) training subsets are then picked out from the original training set in contiguous blocks.

The training subset for the n^{th} classifier (of C classifiers) is a contiguous block of size G/C (where G=sample gain) beginning $(n\text{-}1/C)^{th}$ way down from the top of the source training set, while treating the source training set as a circular buffer. Figure 1 shows an example of data partitioning for 3 classifiers with sample gain=2.

Note that with this partitioning scheme, those training subsets with sample gain less or equal to one are effectively picked without replacement, and those with sample gain greater than one, are effectively picked with replacement because of the overlap of training subsets.

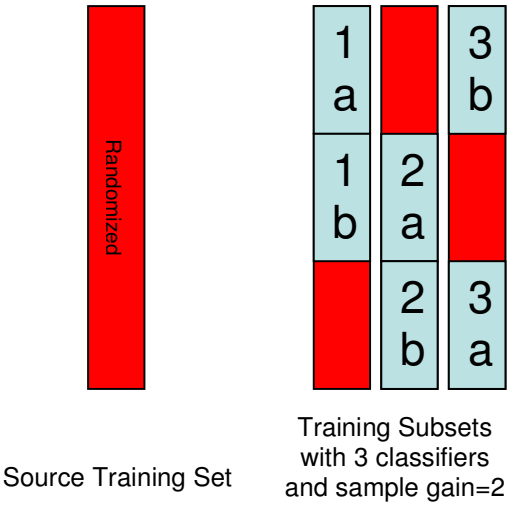

Figure 1: Data Partitioning Example

5.0 Results

Data sets from the University of California Irvine Machine Learning Repository were used to validate our results. Data sets presented here relate to diabetes data (768 data points, 8 features) and forest cover type data (495141 data points, 54 features).

5.1 Increasing Training Subset Size and Speedup vs Accuracy

Figure 2 deals with UCI training data relating to forest cover. The results presented show the sensitivity of accuracy performance as sample gain and number of classifiers are varied.

We can draw two conclusions from this graph:

a) Increasing the number of classifiers while maintaining the same sample gain is accompanied by a drop in classification accuracy. Basically, reducing the classifier training subset size is amenable to improving execution speedup time, but is accompanied by a drop in classification accuracy.

b) However, increasing the sample gain (possibly causing training subsets to be drawn with replacement) increases accuracy performance.

Specifically, we can see that while drawing without replacement at sample gain equal to one, there is some loss in classification accuracy compared to the baseline (the single classifier drawing with replacement from the entire dataset). Accuracy performance for the 100 classifier case with sample gain of 1 drops by 1.3% relative to the baseline. At sample gain equal to 0.1, the accuracy performance drops by 3.6%. However, by increasing the sample gain, accuracy performance can be recovered by increasing the sample gain. Note, however, that once sample gain exceeds 1, our data partitioning scheme moves from sampling without replacement, to sampling with replacement.

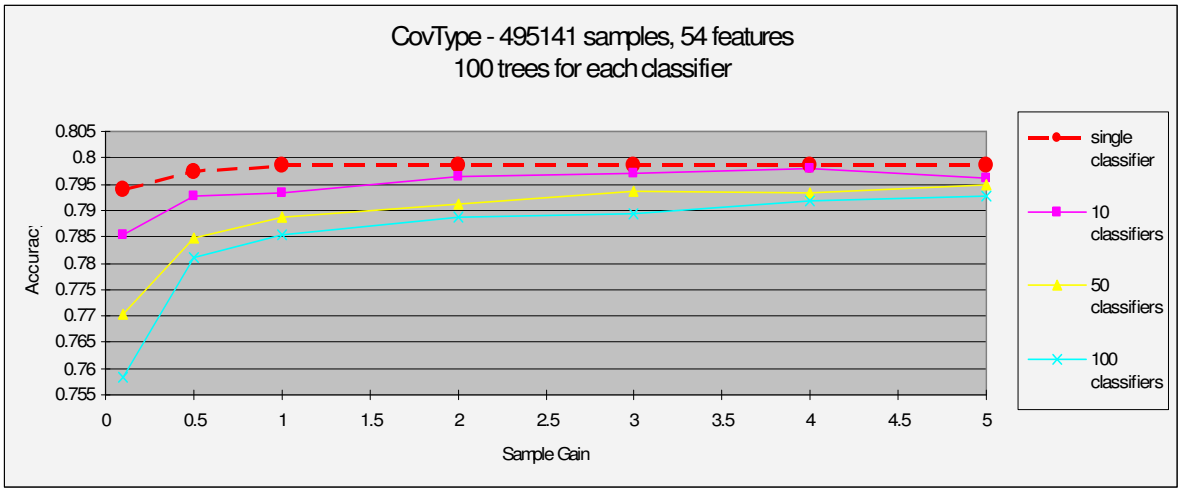

Figure 2: Accuracy vs. Num of Classifiers and Sample Gain – Covtype data set

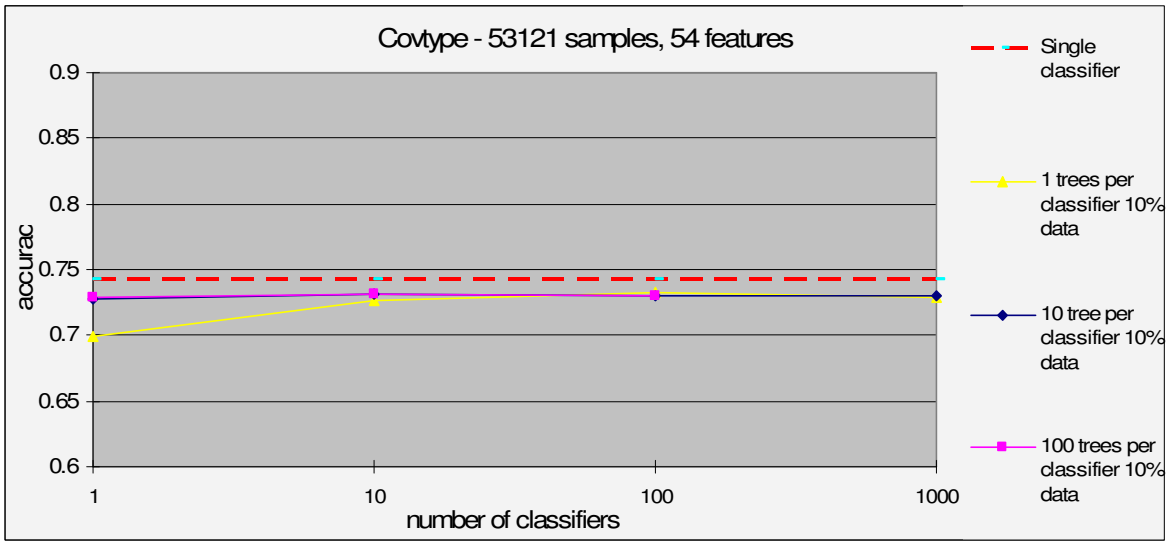

Figure 3: Accuracy vs. Num of Classifiers – Covtype data set

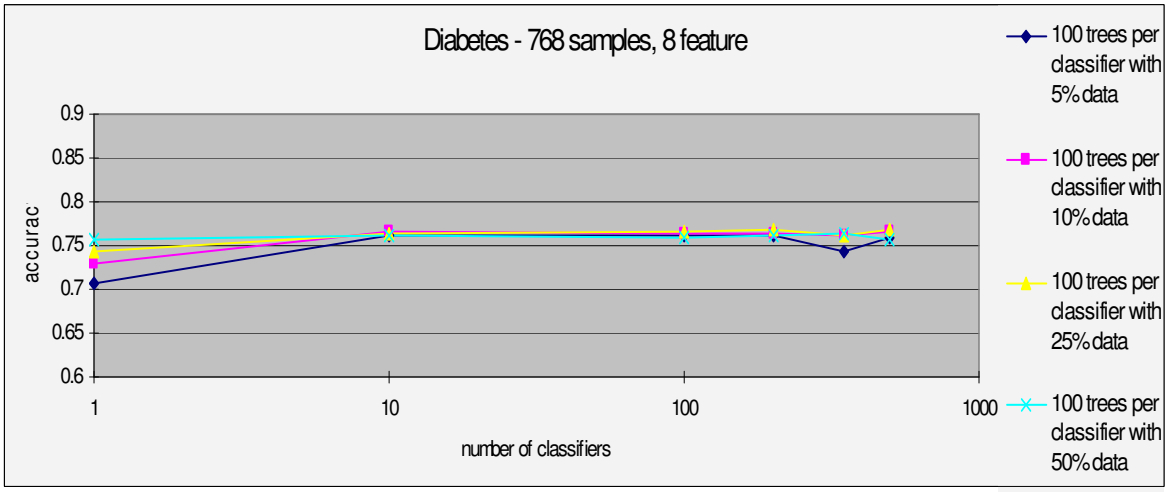

Figure 4: Accuracy vs. Num of Classifiers – Diabetes data set

5. 2 Accuracy Plateau and Fault-Tolerance

Figure 3 and 4 results show the effect of varying the number of classifiers on accuracy performance. We can draw two conclusions from these graphs:

a) Accuracy performance rises quickly when raising the number of classifiers while the number of classifiers is small, and
b) As the number of classifiers increases provided other variables are held constant, accuracy performance plateaus and stays approximately constant

It is the second observation stated that makes this distributed implementation of the Random Forests algorithm a good candidate for the cluster environment where fault-tolerance is necessary. The algorithm is capable of handling lost sub-forest votes and maintains prediction accuracy given a sufficient number of initial classifiers.

Beyond ten classifiers, accuracy performance on the Forest Cover data set in Figure 3 has plateaued and stays approximately constant as the number of classifiers increases into the hundreds. Suppose a cluster of hundreds of classifiers are operating on the Forest Cover data, and some computing nodes may have either failed or do not return their votes within a time-out period. The result node responsible for computing the majority vote may choose not to include the votes of those computing nodes in the final tally. However, given the observation that accuracy performance remains approximately constant for most cluster configurations containing at least 10 classifiers, one can expect that the effect of losing a small number of sub-forest votes on accuracy performance will be negligible.

6.0 Conclusion

The first major conclusion that can be made from this study is that some loss in accuracy is inevitable from parallelization. This loss in accuracy is directly attributable to the data partitioning, which causes each tree to be exposed to only a subset of the data. This is especially true when sampling without replacement. We must qualify this result, however, with the note that the loss in accuracy is dependent on the actual data set.

As a means to counteract the potential drop-off in accuracy, we showed that increasing sample gain beyond could help in recovering lost accuracy performance. We also note, however, that this is a tradeoff, since increasing sample gain results in increased execution time.

Accuracy performance rises quickly when raising the number of classifiers while the number of classifiers is small, and as the number of classifiers increases, accuracy performance plateaus and stays approximately constant, making Distributed Random Forests a good candidate for cluster environments, where fault-tolerance is an asset, due to its accuracy performance's insensitivity to lost sub-forest votes when sufficient classifiers are present.

7.0 Acknowledgements

The authors would like to thank Gary Bradski for his comments, suggestions, and continual support throughout the quarter. Without Gary's mentorship, this project would not have been possible.