Weka-Parallel: Machine Learning in Parallel

Sebastian Celis and David R. Musicant
Department of Mathematics and Computer Science
Carleton College
One North College Street
Northfield, MN 55057
celiss@carleton.edu, dmusican@carleton.edu

Abstract

We present Weka-Parallel, which is a modification to Weka, a popular machine learning software package. Weka-Parallel expands upon the original program by allowing one to perform n-fold cross-validations in parallel. This added parallelism causes Weka-Parallel to demonstrate a significant speed increase over Weka by lowering the amount of time necessary to evaluate a dataset using any given classifier. Weka-Parallel is designed for the researcher who needs to do intense cross-validation calculations and wishes to transparently and simply harness the power of multiple computers. All the details of cross-validation, result aggregation, and multiprocessor communication are completely handled by Weka-Parallel. The practitioner is freed from implementing these tasks, and need only ready the remote machines for incoming work requests.

1 Introduction

Cross-validation is a primary methodology in measuring the success of machine learning algorithms [3, 5]. Techniques such as classification, regression, clustering, and feature selection all make use of cross-validation techniques in predicting generalization success. Cross-validation is inherently parallelizable, as it requires repeatedly breaking the data into different segments, running an algorithm on some of the segments, and using the remaining data to test the results. The desire to do calculations in parallel has become popular in recent years due to the dropping prices of multiprocessor machines, as well as the increasing availability of networked commodity single-processor machines. We present Weka-Parallel, a modification to Weka [9], that allows one to easily implement cross-validation in parallel.

Weka is a flexible environment for running cross-validation on datasets using a variety of machine learning algorithms. Because it is written in Java, Weka proves to be a versatile tool that runs seamlessly on most platforms. This versatility also makes Weka an ideal tool for academic and research institutions.

Weka-Parallel fills a specific niche in the machine learning community. It is aimed at the researcher who has access to parallel processing capabilities and does not wish to be concerned with details of parallel programming. Java is an excellent language to use when writing a program that involves distributed computing as the java.net package is both simple to use as well as powerful. Weka-Parallel uses this package to achieve its distributed computing capabilities which in turn assists the machine learning practitioner.

There are a number of general purpose systems in place for automating parallel process management and communication. Beowulf [8] provides a mechanism for clustering a collection of individual computers to appear as one shared machine. Condor [4] runs silently on machines throughout an installation, and can take advantage of idle machines as requested. While both these environments would also provide the capability to do parallel cross-validation, both require significant setup costs. Moreover, a machine learning practitioner using either of these systems would still have to write code to handle partitioning of data for cross-validation. Weka-Parallel is relatively easy to install, as the user need only have Java and Weka-Parallel installed in a directory that each machine can see. Thus, the software can be either installed on a shared network drive, or it can be installed on each machine individually. Weka-Parallel handles everything else, including partitioning the data for cross-validation, transferring it out to the individual machines, and aggregating and reporting results.

We now provide an overview of the rest of the paper. Section 2 reviews cross-validation techniques. Section 3 gives a high level overview of how Weka-Parallel functions. In Section 4 we describe how a user would go about running Weka-Parallel. We show some experimental results in Section 5 indicating the effective integration and performance of Weka-Parallel. Finally, we conclude and provide thoughts for future directions in Chapter 6.

2 Cross-Validation Overview

While cross-validation applies to any number of machine learning problems, we will restrict ourselves to classification problems for purposes of this discussion. We therefore assume that we have a dataset with m points and k features, indicated by the $m \times k$ matrix A. We also have an $m \times 1$-dimensional vector y, each element of which contains the value that the corresponding row in A is to predict. For classification problems, the elements of y correspond to discrete classes.

In order to test how well a particular machine learning algorithm will generalize to future data, the most common technique is n-fold cross-validation. The matrix A, and the corresponding elements of y, are partitioned horizontally into n separate groups. For each of n iterations, a distinct group (or *fold*) of the data is *held out* from the processes as a *test set*, and the remaining data is used as a *training set*. The machine learning algorithm is run on the training set, and validated via the test set. This yields n estimates of the generalizability of the algorithm as illustrated by its performance on the test set, which are then averaged together to yield the overall test set accuracy. Most often, $n = 10$ (tenfold cross-validation) or $n = m$ (leave-one-out cross-validation).

We note that cross-validation serves a number of purposes in machine learning beyond measuring the expected success of a single technique. It is often used for comparing multiple algorithms, where statistical tests can be performed for determining whether there are significant differences between algorithms [1]. Cross-validation is also a common tool to use in conjunction with a tuning set for finding optimal parameter settings.

3 Weka-Parallel Philosophy

Weka-Parallel was designed for the sole purpose of drastically decreasing the amount of time necessary to run cross-validation on a dataset using any given classifier. This is particularly true when a computationally intense classifier is used, such as Weka's "J48" decision tree, as well as when the number of folds to be calculated is large. With the knowledge that each fold in an n-fold cross-validation is calculated independently, we set out to modify Weka so that different folds could be calculated on different computers. All of the information could then be aggregated on a single machine where the final results would be displayed to the user.

The first hurdle in creating such an application was determining the best way to get the machine running Weka to connect to the machines that will help with the computations. The most simple and elegant way that we found to do this is to use a simple connection established with the Socket class in the java.net package. Each contributing machine could start a daemon that would listen on a specific port. Once a connection between the machine running Weka and the remote machine was established, the socket could then be used to open both an ObjectOutputStream and a DataOutputStream to send all of the necessary information back and forth between the computers.

This architecture provides the capability to set up any number of computers (which we will call the servers) to run daemons that listen for incoming connections. Then, a computer running Weka-Parallel (which we will call the client) can connect to these servers to start distributing work through streams.

The next step is getting the remote machines to correctly use the data to calculate specific folds. We considered using Java RMI (Remote Method Invocation) [2] to have the client directly call the necessary methods on each of the servers. This would have reduced some of the complexity since the client machine would be in complete control by sending all of the necessary information, calling all of the necessary methods, and accessing all of the calculated results. Unfortunately, Java's implementation of RMI proved to be a lot of work to setup. For example, in order to use RMI, the server program must create and install a security manager as well as register the remote objects with the RMI remote object registry for bootstrapping purposes [7]. The registry requires additional setup by the user which would have been an unnecessary complication for anyone who wished to use Weka-Parallel.

The solution that proved simpler than RMI involved having the client transmit integer codes to indicate which method to run. This is accomplished simply by using the sockets that were already created in the opening of the connection. For example if Weka-Parallel is running through Weka's interactive GUI, each server must be instructed to generate graphical data associated with the folds being evaluated. In order to give the server the necessary information, the client sends out an initial integer that denotes whether Weka is being run from the command-line or through a GUI. Once the server receives this information, it immediately expects to receive everything else it needs in order to do the computations for the cross-validation. In this case, the server expects to receive a copy of the entire dataset, the name of the classifier that is being used, as well as the total number of folds that need to be calculated. The final piece of information sent to the server is an integer that represents the index denoting which fold that server should start running.

At this point, the client waits for the servers to finish the calculations and then send the results back. Each server has all of the information it needs to run a particular fold. While each server runs its fold, its connection with the client is left open for simplicity as well as for speed.

Once a server has finished sending back the results from its fold, the client does one of two things. If the client sees that all of the results have been returned and that no more folds need to be calculated, then the connection with this server is simply dropped. If there are still more folds that need to be finished, an index designating one of these folds is sent over the socket to the server. This continues until the n-fold cross-validation is complete. Also, to improve overall efficiency, the client machine itself acts as a server and works on calculating folds at the same time that all of the server machines are also doing calculations.

Which index the client should send to each remote machine is not immediately obvious. Weka-Parallel uses a round-robin algorithm. It starts by assigning the first fold, and then the second, and so on. Once each fold has been assigned, the program starts again at the beginning, assigning in turn each fold that has not already been completed. This continues until the cross-validation is complete. In an ideal situation, with all computers being the same speed and no computers crashing or losing their connection to whatever network they are on, no fold would have to be assigned more than once. But, computers do crash and networks do go down. Thus, the round-robin algorithm along with the client machine helping with the computations act as a safeguard, making sure that no matter what happens, as long as the client machine continues to run Weka-Parallel, the computations will get done.

4 Running Weka-Parallel

4.1 Initial Setup

First, install Weka-Parallel on every machine that is to take part in the distributed calculations. Do this the exact same way you would install Weka. These installation instructions can be found in the file README included with both Weka and Weka-Parallel.

A Weka-Parallel session is run on a single client machine and a number of distributed servers. Each server runs software in the background that listens for incoming requests placed by the client and then fulfills them. For each computer that is to be used as a distributed server, launch the software by entering the following line at a command line prompt on that computer:

```
java weka.core.DistributedServer <port number>
```

Every computer with the DistributedServer program running can act as a server and will listen on a specified port for all incoming work requests. Each request that the server receives is given its own thread, thus allowing many computers to connect to this one server at the same time. This threading will also take advantage of multi-processor machines. If a server has two processors, alter the configuration file described below by telling Weka-Parallel to connect to this computer more than once. This is done by entering the address on the configuration file twice.

This server program can be launched manually on each computer or can be placed in a startup script that runs when each computer boots. A log of all connections processed from the server software is automatically sent to the output screen but can be redirected to a text file if the user so chooses.

4.2 Configuration File

In order for WEKA to know which computers to distribute work to, a config file needs to be present at ~/.weka-parallel on UNIX systems and C:\WINDOWS\.weka-parallel on Windows systems. The first line of the config file should be:

```
PORT=XXXX
```

where XXXX is the port number on which each instance of DistributedServer is listening. The remainder of the config file should consist of the addresses of each computer running an instance of DistributedServer with one address per line.

The configuration file can be created manually or can be created through Weka-Parallel's GUI in the cross-validation options under the classifiers pane.

4.3 Running Weka in Parallel

When using a command line interface, to do cross-validation within Weka in parallel, simply add the -a tag onto a standard Weka command line. For example:

```
weka.classifiers.j48.J48 -t weather.arff -a
```

When using the GUI, after selecting a dataset and classifier, first press the button next to cross-validation to alter the cross-validation settings. Then check the box marked "Run in parallel" and then hit "OK". Finally, run the classifier and the cross-validation will occur in parallel.

5 Experimental Results

In order to demonstrate the effectiveness of Weka-Parallel, we ran the "J48" decision tree classifier included in Weka with default parameter settings on the waveform-5000 dataset from the UCI repository [6]. The waveform-5000 dataset contains 5300 points in twenty-one dimensions, and the goal is to find a classifier that correctly distinguishes between the three classes of waves. We ran 500-fold cross-validation on this dataset using up to fourteen computers from our instructional labs. We had exclusive access to these machines while running the experiments. Each computer has a single 2 GHz Pentium 4 processor with 512 MB of memory, and runs Redhat Linux 6.3. We performed our 500-fold cross-validation fifteen times, always using one of the machines for the main client process and varying the number of machines that were allowed to be used as servers to help with the computing. We also ran an unmodified version of Weka as a control with which to compare everything. The resulting running times are shown in Figure 1.

As is to be expected, significant savings in time can be seen as more computers are added to the task. It should be noted that the "zero remote machines" experiment timed above is a slightly different experiment from the rest, in that Weka-Parallel was told to run in parallel but was not given any remote machines to which it could connect. In all other experiments, the client machine simply connected to the specified number of remote machines.

As can be seen from the horizontal line denoting the amount of time Weka took to run the experiment, the overhead of the added networking code in Weka-Parallel is negligible. Thus, it takes approximately the same amount of time for Weka to produce results for any given query as it does for Weka-Parallel to produce the same results when Weka-Parallel has no remote machines to use. Then, as more remote computers are added, Weka-Parallel takes significantly less time to perform the same task.

6 Conclusions

Weka-Parallel is an easy to use, highly generalizable tool for performing n-fold cross-validation testing of machine learning algorithms. It scales ex-

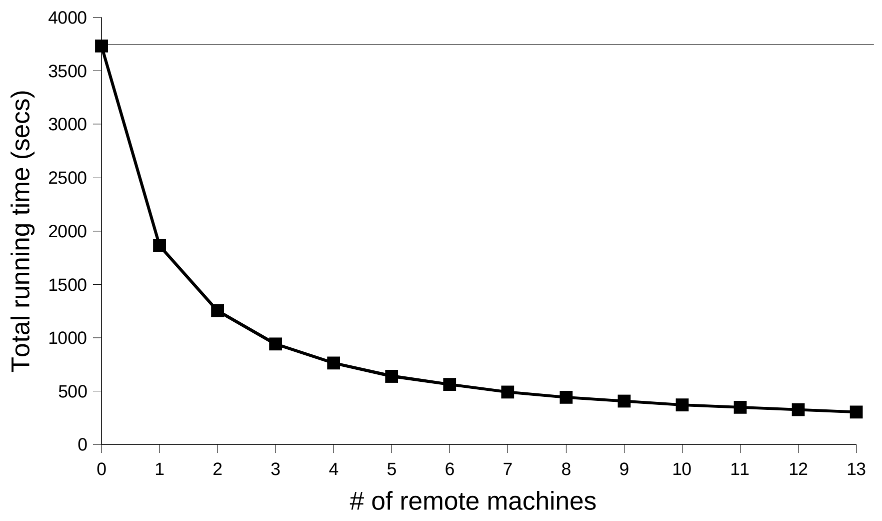

Figure 1: Speedup in parallel cross-validation on single processor 2 GHz Pentium 4 machines. The horizontal line indicates running time on a single processor using an unmodified version of Weka.

tremely well, and is easily run on any machine that supports Java. For future versions of the software, we hope to parallelize more aspects of machine learning that Weka supports.

Weka-Parallel is available for download at `http://www.mathcs.carleton.edu/weka`, and is free for research and academic purposes.

Acknowledgments

Research described in this report was supported by a grant from the Howard Hughes Medical Institute, and by Carleton College.

We thank Mike Tie, Technical Associate at Carleton College for his assistance in appropriating and configuring the machines.

References

[1] Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1924, 1998.

[2] William Grosso. *Java RMI.* O'Reilly, Sebastopol, CA, 2002.

[3] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In C.S. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence.* Morgan Kaufmann, 1995.

[4] M. Litzkow, M. Livny, and M. W. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 104–111, San Jose, CA, June 1988. IEEE Computer Society Press.

[5] T. M. Mitchell. *Machine Learning.* McGraw-Hill, Boston, 1997.

[6] P. M. Murphy and D. W. Aha. UCI repository of machine learning databases, 1992. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[7] Northeastern University, College of Computer Science. Java rmi tutorial. `http://www.ccs.neu.edu/home/kenb/com3337/rmi_tut.html`.

[8] T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranawake, and C. V. Packer. BEOWULF: A parallel workstation for scientific computation. In *Proceedings of the 24th International Conference on Parallel Processing*, pages I:11–14, Oconomowoc, WI, 1995.

[9] The University of Waikato. Weka 3 - machine learning software in java. `http://www.cs.waikato.ac.nz/ml/weka/`.