Sprint: A Scalable Parallel Classifier for Data Mining

William Martin
Math and Computer Science Dept
South Dakota School of Mines
Rapid City, South Dakota
William.martin@gold.sdsmt.edu

Øystein Bedin
Math and Computer Science Dept
South Dakota School of Mines
Rapid City, South Dakota
Oystein.Bedin@gold.sdsmt.edu

Abstract

This paper is a study and implementation of the data mining technique of building decision trees for classification. More specifically, this paper will concentrate on the SPRINT algorithm. The submission of this paper is for a partial completion of the requirements for the final project.

1.0 Introduction

The SPRINT implementation is a scalable, parallelizable decision tree classifier developed by John Shafer, Rakesh Agrawal, and Manish Mehta of the IBM Almaden Research Center in San Jose, CA. The goal was to develop a decision tree classification algorithm that was robust, scaleable, and parallelizable. This project was developed as a hodgepodge of ideas from other papers, some of which co-authored or authored by the same individuals as this paper. Our team will implement the serialized version of the SPRINT decision-tree classifier to satisfy the final project.

1.1 General Implementation Details

Our implementation of SPRINT was developed on the Windows XP Operating System, written in Visual C++ 6.0. The decision to use C/C++ was a wise choice, as if we had time to implement the parallel version, we could move it to the Linux Operating System and provide MPI (Message Passing Interface) calls for inter-process/inter-processor communication. After it was tested on Windows, we moved it down to Linux so that we could call it from our webpage.

Datasets

We used the datasets at http://ftp.ics.uci.edu/pub/machine-learning-databases/, which houses many databases for classification purposes. To give credit, it must be pointed out that some of the files are from a public domain project, STATLOG. This project and the datasets that belong to it are all part of the public domain. We chose the Shuttle and Heart datasets to train and display our results.

Assumptions are made about the datasets. Firstly, we minimized the number of input files by combining the property names and values into one file. This was done to minimize the time required for parsing the data files, because the inherent nature of the properties file was not friendly for tokenizing with c's built-in token function. The file is ASCII based, with characters in the normal range of 32d-122d (blank space through lower-cased z). The format of the input file consists of the first line as the list of property names, followed by the classification or outcome name. These values are all comma-delimited, with the last element

having a carriage return, to signify a new line. From the 2nd line all the way to end consists of the set of training data, listed in attribute form, with the last element being the classification. This too is comma-delimited. It should be noted that the files at the archive were not comma-delimited to begin with; we used Microsoft Excel 2000 to properly delimit the file.

1.2 Split Points

A split point is defined as the most optimal class at the junction in the decision tree to split on. The idea is to find the attribute that "best" divides the training records. If we are at the root node in the tree, we have all of the attributes to compute a "fit" value upon. As we move down the tree, we remove the attributes used previously but only for that nodes above the currently evaluated node, and only for nodes that "live" on the same branch.

The ID3 decision tree algorithm uses a method to evaluate to split points called the *information gain*. The SPRINT algorithm uses an idea, taken from statistics, called the GINI index to evaluate split points. These "goodness" measurements are a measure of the "purity" of the split. The goal of these methods is to discover the attribute that will provide the purest split. The GINI index is similar to the Entropy idea in the ID3 decision tree algorithm, with a difference being the calculation of the fitness of the split for a given attribute.

The GINI index is defined as: $gini(S) = 1 - \sum p_j^2$, where S is the data set, p_j is the relative frequency of class j in S.

1.3 Structures

The SPRINT algorithm requires many different data structures to complete its tasks. Here is a short list, with a description:

Structure Name	**Description**
Attribute Lists	An attribute list for each attribute in the data.
Histograms	Used for continuous attributes, with two elements: C_{above} + C_{below}
Count Matrix	Used for categorical attributes (class count for current node)

1.4 Detailed Implementation

The sorting routine used on the continuous attributes is sorted using a modified Shellsort routine. The routine had to be modified to adapt to our proprietary structures.

When using C (and not using any special vector libraries) we become responsible for creating memory and destroying it. This implies that we must define structures to be able to grow dynamically. Before long, we become overwhelmed in a mountain of pointers and multi-dimensional dynamic arrays to support large datasets with heterogeneous attribute types (continuous or categorical). Shellsort was chosen because the recursive Quicksort implementation does not work well with larger datasets, because we quickly run out of stack space. Given the time constraints, we needed a faster way of implementing a sort, with respectable time complexity.

There was a finitely, small amount of time to implement everything that was described in the paper. Thus, many items had to be decided upon based on the factor of necessity. We made a decision to implement a non-parallel version, with no disk swapping. These two elements were outside the parameters of the class, although they are the most important aspects of the SPRINT algorithm. We also decided to only implement the continuous implementation after noting that the categorical implementation wasn't working quite as well. This turned out to be of great advantage because most of the dataset's attributes are either homogenous or heterogeneous, with a larger number of datasets mainly in continuous form.

One more important detail is that datasets with attribute value-pairs that are "saturated" with zeros cause problems in the calculation of the GINI index. This has the added detrimental effect of causing a split value of 0.0, which produce erroneous decision paths in the final tree. If the data values do not include negative values, then our split value is meaningless and we build a heavily lopsided tree, which becomes lopsided starting at the first time we compute a 0.0 split value. We changed our implementation to return an error message when this case occurs. The data set specialist (i.e. end-user) will have to include their own workaround, such as "shifting" the values so that there isn't an over-abundance of zeros and other values being only positive. The "shift" would subtract an amount from the value so that negative and positive numbers would be available (which would change 0 to be a negative number), thus building a more balanced tree and eliminating erroneous split values.

1.5 Parallelization

The authors discuss parallelizing the SPRINT algorithm, which is of interest to Computer Science students because we will all encounter some form of parallelization in our careers. It might not be evident that you are parallelizing, because of the nature of libraries and abstractions, but it will take some form. However, we did not have time to implement this feature.

1.6 Other Features

SPRINT has the capability to train with larger datasets, based on the fact that it can "swap" some of it to disk. This would be a nice feature because it would allow us to load larger datasets. Future implementation will include this, however we did not have time to implement this yet.

1.7 Results

The SPRINT algorithm is a great algorithm for implementing decision trees. We had successful results with the amount of implementation we had done. When using data that is put into a dataset warehouse, the results are actually "real" values; the decision trees are usually much cleaner because they mimic the human decision thought process and hold recorded data from an actual event. We tested our trees by ading values that would cause larger, more complex trees (see figure 1.0 in Section 3.0). The dataset for that tree is:

age,salary,credit_risk
23,50000,high

17,30000,high
43,40000,high
68,50000,low
32,70000,low
20,20000,high
31,90000,low
65,35000,high
28,47000,low
29,59000,high
44,30999,low
55,22000,high
92,45000,high

We were not able to produce accuracy and timing results, as we didn't have any other programs to compare our values. We also didn't test larger datasets thoroughly. This was mainly because our algorithm handles only outcomes that have 2 classifications (i.e. high or low, black or white, 1 or 2, etc). It was very difficult to find datasets that are as specific as our needs are; no categorical data, attribute-values that are not saturated by 0s, and only two outcomes.

2.0 Sprint GUI

An important question that came up was: What's the best way to display our result, and at the same time quickly verify that the result from the algorithm is correct. The application itself produces a regular text file, which contains lines of words and numbers. Even for those who know exactly how this file is designed may have a hard time keeping track of what describes what. Another nice feature would be if the user were able to run a query, which produces a result that's easy to read. That's the basics for why we early in the project decided to make a GUI, graphical user interface, for our project. We started out with a discussion on the specifications – what's a good GUI and what do the user want. Some of the requirements we came up with were to make a GUI that was platform independent – to a certain degree. Also, the user should easily make a query and get the result quickly in an easy-to-read manner.

While working on our requirements, we looked into several possibilities for a GUI. We're not going to discuss all the different solutions in this paper, but Visual Basic, Macromedia Flash, C++ were some of the suggestions that came up. With the idea about making it platform independent, we started to think about a browser application. An Internet browser is something all platforms got, and if made a server-side application all existing browsers should be able to display it. We some knowledge in what possibilities that lays in this, we decided to make a PHP / MySQL server based application. The MySQL database should be used to store the data and information temporarily while working on a certain tree. The PHP application/GUI is the main interface against the user.

None of us had done any thing like this in PHP or similar language before. The first case we had to solve was to figure out how to display the graphical part of the tree. After some time searching for how to display graphic using PHP, we decided to use JpGraph. JpGraph is an OO class library for PHP 4.1 (or higher). The only requirement for the server is that the PHP installation needs to have the GD library setup correctly. (in most cases it is) The GD library is an ANSI C library for the dynamic creation of images. GD

creates PNG and JPEG images among other formats.

JpGraph makes it easy to draw graphs with a minimum of code and complex professional graphs, which require a very fine grain control. JpGraph is released under a dual license, QPL 1.0 (QT Free Licensee), for non-commercial, open-source and educational use and *JpGraph Professional License* for commercial use.

Even though the graphical objects are easy to get displayed, we had to develop a great amount of code to get the tree displayed correct: The layout of the nodes, the text and the query part. Our goal was to feed the application with raw data let the application draw the tree. This is how we do this: The SPRINT application is written in C++, which is made to run as a console application. When the user "uploads" a raw data file, the PHP code executes the C++ application, which generates the tree-data file, from which the PHP draws the tree. In this way we made a great combination of both "modern" browser and "regular" console applications.

What is required to run our application, is a web server with PHP / MySQL setup correctly, a C/C++ compiler which is used to compile our main application, and correct user rights to store/execute the files. During our development we were using Linux Redhat 8.0 with Apache 2.0 web server, PHP v.4.2.2 and MySQL v. 3.23.54, JpGraph 1.12.1

An example of our application is shown in the figure below (see figure 1.1). Here a file is uploaded and the decision tree for "credit risk" based on age and salary is drawn. As shown in the figure, the user has done a query – and the result is highlighted (blue nodes) in the tree. In this case the query is the age 65 and salary of 50000 – which cause the result to be "credit risk equal to low".

We also implemented ability to resize the tree and the nodes. If the tree grows big, it will be hard to look at the nodes further down in the tree. This makes it available for the user to scroll around to look closer at each part of the tree.

3.0 Conclusion

Implementation of the SPRINT algorithm was extremely enjoyable, with a great amount of enthusiasm to every success we had along the way. We gave SPRINT a personal flavor by extending the basic algorithm to provide a graphical user interface that was platform independent. This, we believe, was the most powerful aspect of our implementation because any user, whether they operate on Macintosh, Windows, Linux, or some other variant can build decision trees with our software.

We increased our strengths and knowledge in how decision tree algorithms are employed both in academia and industrial. We also discovered several pitfalls of understanding a paper, but not immediately seeing the problems inherent in data that can be noisy or weighted towards a particular value.

3.0 Figures and Screenshots

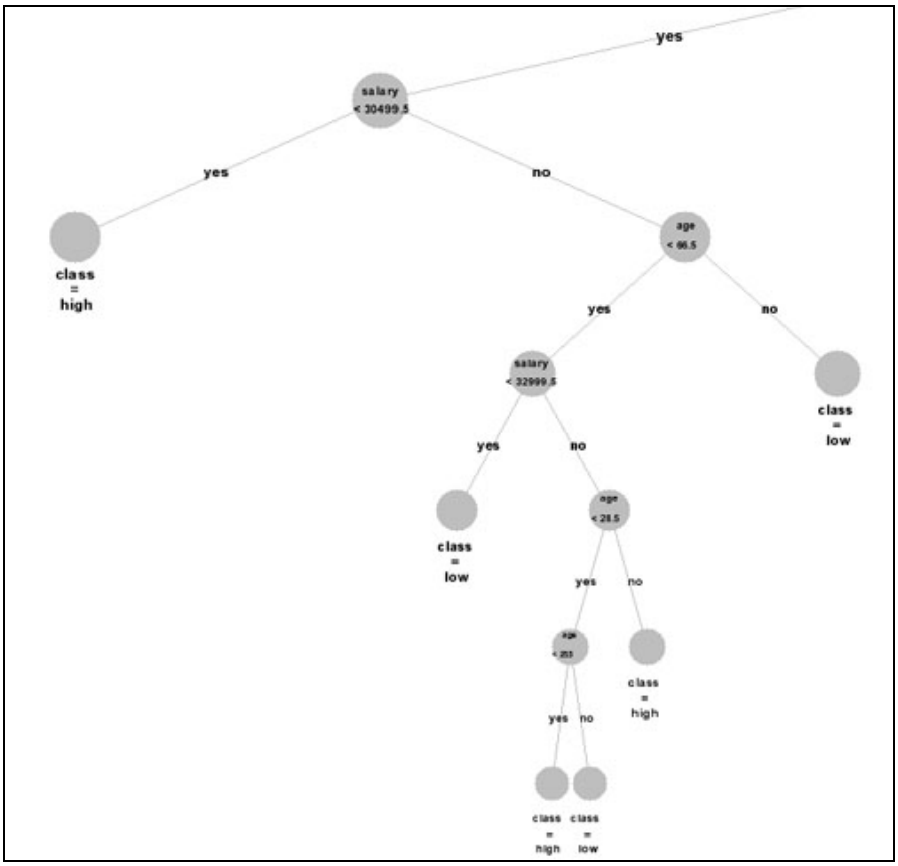

Figure 1.0: More complicated tree

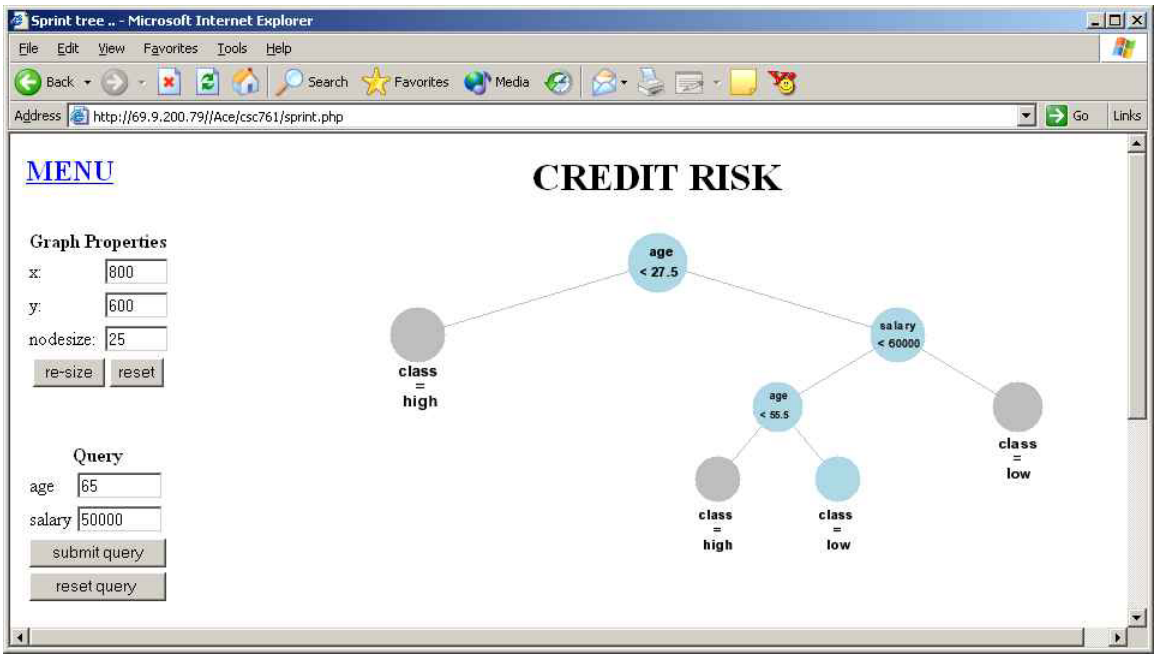

Figure 1.1: Screenshot of the SPRINT GUI.

4.0 References

JpGraph class library
http://www.aditus.nu/jpgraph/index.php

GD Graphics Library
http://www.boutell.com/gd/

J. Shafer, R. Agrawal, M. Mehta., "***SPRINT**: A Scalable Parallel Classifier for Data Mining.*", In 22nd VLDB Conference Data Mining. In 22nd VLDB Conference, Sept 1996.

M. Mehta, R. Agrawal, J. Rissanen., "***SLIQ**: A fast scalable classifier for data mining.*", In 5th Intl. Conf. on Extending Database Technology mining. In 5th Intl. Conf. on Extending Database Technology, March 1996.

M. Mehta, J. Rissanen, and R. Agrawal., "***MDL**-based decision tree pruning. based decision tree pruning.*", In Int'l Conf. on Knowledge Discovery in Databases and Data Mining (KDD-95), Montreal, Canada, Aug. 1995.

Penaloza, Manuel Dr., "***CSC761:** Advanced Artificial Intelligence"*, South Dakota School of Mines and Technology, Spring 2003.

M. V. Joshi, G. Karypis, and V. Kumar, "***ScalParC**: A New Scalable and Efficient Parallel Classification Algorithm for Mining Large Datasets*", in "Proceedings of the Joint Twelfth International Parallel Processing Symposium and Ninth Symposium on Parallel and Distributed Processing", 1998.
http://citeseer.nj.nec.com/joshi98scalparc.html

University of California, Irvine, School of Information and Computer Science, "*Machine Learning Database Repository*", http://ftp.ics.uci.edu/pub/machine-learning-databases/

Karypis, George, "ScalParC A New Scalable and Efficient Parallel Classification Algorithm for Mining Large Datasets", Powerpoint Presentation, http://www-users.cs.umn.edu/~karypis/publications/Talks/ScalParC/sld009.htm