NISS

Classification of Partially Observed Data with Association Trees

Michael Last, Sandro Fouche, Alan F. Karr, Alex Orso, Adam Porter and Stan Young

Technical Report Number 164
November 2006

National Institute of Statistical Sciences
19 T. W. Alexander Drive
PO Box 14006
Research Triangle Park, NC 27709-4006
www.niss.org

Classification of Partially Observed Data with Association Trees

Michael Last MLAST@NISS.ORG
National Institute of Statistical Sciences
19 T.W. Alexander Dr. Box 14006
Research Triangle Park, N.C., 27709

Sandro Fouche SANDRO@CS.UMD.EDU
Department of Computer Sciecne
University of Maryland
College Park, Md. 20742

Alan F. Karr KARR@NISS.ORG
National Institute of Statistical Sciences
19 T.W. Alexander Dr. Box 14006
Research Triangle Park, N.C., 27709

Alex Orso ORSO@CC.GATECH.EDU
College of Computing
Georgia Institute of Technology
Atlanta, Georgia,30332

Adam Porter APORTER@CS.UMD.EDU
Department of Computer Science
University of Maryland
College Park, Md. 20742

Stan Young YOUNG@NISS.ORG
National Institute of Statistical Sciences
19 T.W. Alexander Dr. Box 14006
Research Triangle Park, N.C., 27709

Editor:

Abstract

Classification methods have troubles with missing data. Even CART, which was designed to deal with missing data, performs poorly when run with over 90% of the predictors unobserved. We use the Apriori algorithm to fit decision trees by converting the continuous predictors to categorical variables, bypassing the missing data problem by treating missing data as absent items. We demonstrate our methodology in a setting simulating a distributed, low-overhead, quality assurance system, where we have control over which predictors are missing for each observation. We also demonstrate how performance can be improved by the introduction of a simple adaptive sampling method.

1. Introduction

We describe a unique science problem. We are trying to predict when a complex computer program fails, or crashes. This information will be used to find and fix faults in the program. We have a large number of possible debugging statements used to instrument the program, but if we turn them all on, the program runs too slowly to be useful for end-users. We cannot test all the possible paths through the program during normal quality-assurance testing as there are too many paths and configurations for a limited quality-assurance budget. We are working on distributed testing methods to evaluate the program in the field by turning on a relatively small number of debugging statements in each fielded instance. When a program is installed, a central server will be queried, indicating which debugging statements will be active. This set of active statements can be different for each program instance. All the other debugging statements are turned off so we do not have information on those parts of the state of the program.

This plan gives rise to rather unusual data sets, where more than 90% of the predictor variables for an installed instance of the program are missing. How do we design which statements to turn on and how do we analyze the resulting data set when 90% of the data for each record is missing? Conventional imputation techniques will not solve our missing-data problems. There are no methods which give reliable results when 90% of the data are missing. We assume that the computer is essentially deterministic. If we run exactly the same program the computer will respond in exactly the same way. If we run a path that gives a crash, running the same path will always give a crash. Our methods can be adapted to a non-deterministic setting, but, for the sake of clarity, these modifications will be discussed as asides.

The predictors that we are working with are counts of how many times different parts of the program are called. These predictors take on a range of integer values. Some predictors take on only one value across all executions, whereas others can take on several hundred values. A more thorough discussion of this problem can be found in Haran et al. (2005).

This paper deals with how to analyze these partially observed data. We run our initial experiments with a random design, and then will refine our design by using an adaptive sampling strategy. Our key assumption is that whether our predictors are observed is independent of the outcome. We can enforce this independence by turning on debugging statements at random. So long as this assumption is met, our proposed method generalizes to other settings. For instance, when there is a network of sensors where, for power reasons, only a limited number can be queried at a given point in time, or, where the sensors can be moved, so that at any time only a chosen sample of the possible measurements are made, our algorithm should be of help. While we do not consider categorical predictors, our algorithm can easily be modified for this situation.

2. Definition of Problem

We consider a problem where there exists a large number (e.g., thousands or more) of predictors, of which only a small fraction (less than 10%) are observed for each data point. Each training observation belongs to one of {pass,fail}, which we represent as {0,1}, and each test observation is assigned a prediction from {0,1,"can't predict"}, where "can't predict" represents the observations we think there is insufficient information to classify. We want to predict which class the new observations belong to. In the trials considered in this paper, we are working with tens of observations per predictor, and failure rates ranging from about 8% to about 50%.

We are interested in the decision rules for locating bugs in computer programs. We want to find as many correct rules as possible, to best characterize the failure mode(s) of the program. Rules for predicting both failures and successes are of interest for this.

The rest of the paper is organized as follows. Section 3 presents a review of algorithms in the literature which we use or which are similar to what we propose. Section 4 introduces our proposed Association Trees Algorithm, which uses the Apriori algorithm to find decision trees.

We also present a refinement of the proposed Association Trees Algorithm, where we use adaptive sampling to dynamically select more informative predictors. In section 5 we provide the results of an experiment on data simulated by sampling from fully instrumented runs of a subject program, and what we learned about the effects of the tuning parameters involved in our algorithm. Finally, in section 6 we discuss our findings.

3. Review of Existing Algorithms

In this section we briefly review several classes of algorithms that are potentially useful for this problem. Decision trees are used as a baseline for evaluating our methods. They are fast, scale to large problems, and are designed for interaction detection. Given complete data, we are able to train perfect decision trees (Haran et al., 2005). However, when we simulate cases with large amounts of missing data, to represent real-world conditions, conventional decision trees did not perform as well. For instance, Classification and Regression Trees, or CART (Breiman et al., 1984), using the rpart function in R with default settings, had an error rate of 8.5% on the *training data*. The Apriori algorithm is an efficient method to find frequent shared traits in large data-sets, and is useful for finding traits which predict given outcomes. We use it in the development of our decision rules. It would be accurate to describe our algorithm as a method by which the Apriori algorithm is used to build a decision tree in the presence of partially observed data. Finally, Logical Analysis of Data considers a similar class of problems, but does not scale with the number of predictors we are considering.

3.1 Decision Trees

Decision trees are a common class of prediction algorithms. They classify new observations by a hierarchical series of rules, typically forming a binary tree. Each node has two daughter nodes; a true/false statement about one of the predictors (typically of the form "Var>C") is evaluated to determine which daughter node to use. Leaves are the final classification rules. There are several algorithms, differing on how to generate the true/false statements about the predictors, and in how many nodes to generate. The effect of these trees is to divide the decision space into rectangles upon which class membership is predicted. The most common implementation in the statistical literature is CART, described in Breiman et al. (1984). Another implementation is Partitionator, based on the FIRM algorithm of Hawkins and Kass (1982), www.goldenhelix.com, first described in 1979.

3.2 The Apriori Algorithm

The Apriori algorithm was first described by Agrawal and Srikant (1994). Each observation is characterized by the presence or absence of a collection of items. For instance, observations could come from transactions at a grocery store, where we record what items were bought at the same time. The goal of the Apriori algorithm is to find items that occur together in groups, and, given a subset of the items purchased, be able to predict what other items the consumer has purchased (or may be interested in purchasing).

The Apriori algorithm finds these frequently occurring itemsets. Do we have eggs? Milk? Bread? Butter? We need to specify how many times an itemset occurs for it to be of interest to us. This minimal value is usually expressed as a proportion, called the support. We also specify how often an inferential rule needs to hold, and call this the confidence. A typical rule would be {bread, jelly} $\rightarrow$ peanut butter. The key insight of the Apriori algorithm is that if an itemset appears with at least some specified frequency (the support), then all of its subsets also appear at least that number of times. Thus, when checking for itemsets of size n, only unions of itemsets of size $n-1$ need to be checked. Assuming that there are a limited number of frequent itemsets, this provides a major improvement on the worst-case exponential search-time (all possible itemsets). As originally described, the Apriori algorithm only works with 0/1 data (presence or absence). The extension to categorical data is done by marking the presence or absence of each category. In our problem, we will be dealing with continuous predictors (counts). The extension to continuous data remains an open question. Fukuda et al. (1996) propose a method by which the Apriori algorithm can be extended to one continuous variable and any number of categorical variables. Brin et al. (2003) extend this work, allowing for two continuous variables. However, allowing arbitrary numbers of continuous variables is still an open problem.

We simplify the issue of dealing with continuous predictors by only splitting each continuous variable into two regions, instead of allowing for a split into an arbitrary number of regions. We also do not (at present) consider interactions when deciding where to split continuous variables to get categorical variables. This is sufficient for our current application.

3.3 Logical Analysis of Data

Logical Analysis of Data (LAD), described in Boros et al. (2000), takes a similar approach to the one we propose, although it is not adapted to dealing with large datasets or with frequently missing predictors. Boros et al. want to find logical rules for predicting an outcome from some set of predictor variables. They also deal with the issue of how to convert continuous predictors into sets of binary predictors, and can handle some level of missing data. However, their proposed method creates binary predictors covering every possible interval of a continuous predictor. This does not scale to the size problem we are considering. Boros et al. deal with missing data by requiring a proposed rule to be accurate when we can observe all the predictors, and for it never to describe another outcome for any assignment of the missing values. While this may work when predictors are rarely missing, if any rule requires an interaction between predictors, it will not work when most of the predictors are missing.

4. Our Algorithms

Our basic Association Trees algorithm:

1. In this first stage, the algorithm transforms the predictors into items that are considered either present or absent, in two steps. *First*, it screens each predictor and checks that the Spearman rank correlation between the predictor and the observed outcome exceeds a minimum threshold (Spearman's rank correlation is similar to the traditional Pearson's correlation, except that the actual value of each variable is replaced with a rank, i.e., 1 for the largest value, 2 for the second largest value, and so on, which makes the correlation measure less sensitive to extreme values.) The goal of this step is to discard predictors whose values are not correlated with outcomes, since they are unlikely to be relevant for the classification.

   Second, the algorithm splits the distribution of each remaining predictor into two parts, such that the split point is the point that minimizes the p-value of the t-test for outcome. Ideally, after the split all runs with one outcome would have values above the split, while all runs with the other outcome would have values below it. If the p-value is below a maximum threshold (we used .0005 in the experiments below), the algorithm creates two items: the first item is present in a given run if the predictor's value is below the split point; the second item is present if the value is above the split point. Neither item is present if the corresponding predictor was not being measured during the run. We also represent outcomes as separate items (i.e., "pass" and "fail" for the experiments reported in this paper).

2. After the algorithm completes Stage 1, the original set of training data has been transformed into a set of observations, one per run, where each observation is the set of items considered

present for that run. The goal of the second stage of the algorithm is to determine which groups of frequently occurring items are strongly associated with each outcome. To this end, it applies the Apriori data-mining algorithm (Agrawal and Srikant, 1994), described in section 3.2. Choice of appropriate supports and confidences is discussed below in section 4.1 Finally, to reduce computation times, we may want to limit the length of association rules. In the experiments reported below, we limited the length to three. This has the downside that, if four items must be present to perfectly predict an outcome, our algorithm will not be able to find the corresponding rule.

3. Given the association rules, the third and last stage of the algorithm is the one that performs the outcome prediction. Given a new run, the algorithm finds the rules that apply to it. If all applicable rules give the same outcome, then it returns that outcome as the prediction. If there is disagreement, or there are no applicable rules, the algorithm returns a prediction of "unknown". We chose this unanimous voting scheme in order to be conservative. One might also decide to use simple majority voting or a weighted voting scheme (if the penalties for incorrectly predicting different outcomes are unequal).

4.1 Parameters

As currently implemented, for the problem of describing bugs we have nine parameters to set, described in table 4.1. Note also that we are including three items such as the **number of blocks** and the **block size** as parameters, even though they are better described as experimental design issues, since they determine the sample size. Notice also that, of the six remaining parameters, several may be set based on theoretical considerations such as a desired false positive rate, so the tuning problem becomes more tractable.

There are two parameters that affect how predictors are cut before checking for association rules. The first is a **minimum correlation** between the predictor and the class. The lower it is set, the more rules will be found; however, there will also be an increase in the memory and processor requirements. The other parameter at this step is the critical **maximum p-value** for the t-test for making a cut, which can be set based on controlling the family-wise error rate, or the False Discovery Rate. While this makes the minimum correlation redundant, we find that initial screening on correlation results in faster executions of the algorithm.

The other parameters are the **minimum support** and **minimum confidence** required to declare a rule for each of our classes. For the supports, there are two main considerations. Supports that are too small find too many frequently occurring itemsets, and require too much memory to hold the intermediate calculations. Further, small supports might find rules that only exist by chance. Finding a minimum support to address the latter concern can be done by testing the hypothesis that the group has an observed prevalence of the minimum confidence, but a true prevalence of the overall population. Set a minimum support to reject this hypothesis at any desired confidence level. Finally, too high of a support will find too few rules. In our experiments, we chose a minimum support to satisfy the hypothesis test minimum, and then checked several values around it.

The minimum confidence should be set based on how well we think it is possible to separate the classes. In our example, where we look at crashes of computer programs, the underlying process is deterministic, so we required supports of 1. Support will, of course, vary with the application.

Finally, we have three parameters that may be better described as experimental design decisions, since they describe how much data we have. We have one for the **number of predictors measured** per observation. Another parameter is our **block size**. For each block of size b, choose the same predictors to observe for each observation in the block. The final parameter in this group is the **number of blocks** we sample, which will in turn determine our sample size.

For the bug-detection system we describe below, we considered maximum p-value and the required confidences for support rules as values we could set by theoretical considerations.

Parameter:	Description
Minimum Correlation	The minimum correlation between a predictor and class membership. Used to keep the number of rules manageable. Recommend starting with a high value and searching down for faster run times
Maximum P-value	What is the maximum p-value for the t-test to to determine if a split should be used?
Number of Blocks	How many blocks are sampled. Appropriate value depends on application, computing resources, and availability of data
Block Size	How large of a block do we sample the same predictors for? In the extreme cases, either all the data has the same predictors sampled, or every predictor has a different set of predictors sampled
Number of Predictors Measured	How many predictors are measured for each sample?
Minimum Support Successes	What proportion of the time do we need to see a rule that predicts a successful execution to use it?
Minimum Support Crashes	What proportion of the time do we need to see a rule that predicts a crash to use it?
Minimum Confidence Successes	What proportion of the time does a rule predicting successful execution need to be correct to use it?
Minimum Confidence Crashes	What proportion of the time does a rule predicting a crash need to be correct to use it?

Table 1: Description of parameters

4.2 Adaptive Sampling

We found that most of the predictors that we sampled gave us no useful information in our predictions. This suggested that, if we could detect which predictors were useful, and preferentially sample these predictors, we could increase the performance of our Association Trees. This requires that data be collected sequentially, so that we can run a preliminary analysis on our already-collected data before we choose what data to observe in the future. This method is also usable in fewer settings than the method described above. If which predictors are observed is determined at random, then the adaptive sampling techniques described below cannot be used, whereas the technique described above can be.

We start our adaptive sampling procedure by assigning a weight to each predictor, which can either incorporate pre-existing knowledge, or, as in the case in the experiments reported below, simply be set uniformly to 1. After each round of data collection, we look at the observed correlation between each predictor observed during the round of data collection, and the outcome. When the observed correlation is above some minimum threshold τ (in absolute value), we increase the weight of that predictor by 1. As the amount of data collected tends toward infinity, we will observe each predictor increasing numbers of times. Thus, the observed correlation will tend towards the true correlation for each predictor. The predictors that are not correlated (have correlations below τ) with the outcome will eventually stop having their weights increased. Thus an ever-increasing proportion of our measured predictors come from the set of highly correlated predictors.

4.3 Proof of Asymptotic Correctness of Adaptive Sampling

Definition: Let τ be the minimum threshold we set for the correlation between a predictor x and the outcome Y to increase the sampling weight on the predictor. We say a predictor x is highly correlated ($x \in \mathrm{HC}$) if $|\mathrm{cor}(x, Y)| \geq \tau$.

Let S_k denote the set of predictors sampled in the k'th round of sampling with Adaptive Sampling Association Trees. Let p of P predictors be sampled in each round.

Theorem: Let W_x^k be the weight assigned to predictor x after round k. Then,

$$W_x^k \to \infty (\text{a.s.}) \text{ iff } x \in \mathrm{HC}$$

Proof: By the strong law of large numbers, the estimated correlation of any predictor with the outcome will converge to the true correlation a.s. as the number of times it is observed increases to infinity. Unless the true correlation is τ, eventually we will correctly classify it as either highly correlated or not each future time we observe it. So, if $x \notin \mathrm{HC}$, after some number of observations, its empirical correlation will be less than τ for all future times we observe it. It follows that for $x \notin \mathrm{HC}$, its weight will not increase above some finite value.

To prove that the weight of all highly correlated predictors tends to infinity, we will first show that the number of times every predictor is observed tends to infinity. If we observe p of P predictors each round of testing, then the probability on the k'th round of testing that we observe a predictor is bounded below by $p(P + kp)^{-1}$, since the weight of the predictor is at least 1, the total weight of all the predictors is at most $P + kp$, and we have p draws. Rewriting this as $(P/p + k)^{-1}$, we see that this is bounded below by the tail of the harmonic series (start at the least integer greater than P/p). Thus, as the number of observations tends to infinity, the number of times we will see a predictor will tend to infinity. Since the empirical correlation of all predictors converges to the true correlation, after some number of times observed, with the exception of those predictors such that $|\mathrm{cor}(x, Y)| = \tau$, every additional time we observe a highly correlated predictor we will increase its weight. In the case of those predictors x such that $|\mathrm{cor}(x, Y)| = \tau$, the empirical correlation will converge to a normal distribution centered on τ. By the law of iterated logarithms, it follows that in this case, half the time we sample x we will increase its weight. Thus, as $k \to \infty$, the number of times we observe each predictor tends toward infinity, and we will assign a weight tending toward infinity to all the highly correlated predictors. $\blacksquare$

Remark: Note that the convergence in the theorem cannot be made almost sure, since we will see every predictor infinitely often. This is a virtue of our weighting scheme. Without it, we would admit the possibility, if there are more highly correlated predictors than we sample at each step, that there would be highly correlated predictors that we would miss by chance, if they were not sampled enough in the early rounds of testing.

Corollary: If there are more than p highly correlated predictors, then as $k \to \infty$, for each predictor x,

$$\mathrm{P}(x \in S_k) \longrightarrow 0 \text{ if } x \notin \mathrm{HC}.$$

Proof:Let $W_{\sim HC}$ be the total weight assigned to all the predictors that are not highly correlated, and W_{mHC} be the minimum weight assigned to any highly correlated predictor. Then,

$$\mathrm{P}(x \in S_K)\text{for some } x \notin \mathrm{HC} < 1 - (1 - \frac{W_{\sim HC}}{W_{mHC} + W_{\sim HC}})^p$$

Since $W_{\sim HC}$ is finite, and $W_{mHC} \to \infty$, the right hand side of this inequality converges to zero.
■

Corollary: If there are p or fewer highly correlated predictors, then as $k \to \infty$, for each predictor x,

$$\mathrm{P}(x \in S_k) \longrightarrow 1 \text{ iff } x \in \mathrm{HC}.$$

Proof: Let there be $q \leq p$ highly correlated predictors. Using the notation of the previous corollary, we get a lower bound on the probability that all the highly correlated predictors are in S_k of

$$\Pi_{x \in S_k} \frac{W_{mHC}}{W_{mHC} + W_{\sim HC}}$$
$$= (\frac{W_{mHC}}{W_{mHC} + W_{\sim HC}})^q \to 1$$

since the probability that a highly correlated predictor is selected is bounded below by $W_{mHC}/(W_{mHC} + W_{\sim HC})$, and that assuming the q highly correlated predictors will be selected as the first q will underestimate the true probability. For $k > K$, for all $x \notin \mathrm{HC}$, W_x has a maximum. Since there are $p - q$ additional predictors sampled for k large, $\mathrm{P}(x \in S_k) < 1 - (1 - W_x/W_{\sim HC})^{p-q}$ for all $k > K$.
■

5. Experimental Data

As a subject program for our experiments, we used JABA (Java Architecture for Bytecode Analysis),[1] a framework for analyzing Java programs that contains 60,000 lines of code. We have multiple versions of JABA, each containing one or more known faults, and a test suite of 707 cases. We obtained the set of data for our experiments by running the 707 test cases on a fully-instrumented version of JABA. (Haran et al. (2005) contains a more detailed description of JABA and how it was instrumented.) Among the versions of JABA available to us, there are 14 versions with failure rates of at least 8%. The instrumentation counting the number of times each method was called produced approximately 1240 possible predictors (there were slight variations from version to version). From this complete data set, we simulated the desired amount of partially observed data by, for each observation, selecting m of the measurements to be observed, and setting the remaining $1240 - m$ to Not Available (NA). We took a block-sampling approach. Each block sample of size b test cases had the same m predictors measured. Which inputs (test cases) and which predictors were included was

1. `http://www.cc.gatech.edu/aristotle/Tools/jaba.html`

determined by a simple random sample for each block; that is, b unique inputs would have m unique measurements taken from the data of fully-instrumented runs of JABA. The remaining $1240 - m$ measurements would be NA.

For the initial Association Trees experiments, we simulated 18,000 observations, with block sizes of 6, 12, and 24 (3,000, 1,500, and 750 blocks respectively). When we ran experiments with adaptive sampling, we used only a third as many observations. The first half of the data was used as a training set, and we tested on the remaining data. We tried both 50 and 100 observed predictors per observation.

We tried various settings in an exploratory analysis on one version of JABA, with a goal of seeing how varying the parameters would affect the quality of the prediction. We set the minimum confidence to 1 for both crashes and successful executions of the program, since we consider the process to be deterministic. We ran extensive tests of Association Trees on one version of JABA, selected because it had a 12.2% failure rate (on the test cases), to determine the effects of tuning, and to determine which tuning parameters had the biggest impact. We tested minimum supports of .0016, .0033, and .0066 for rules predicting success, and .0005 and .001 for rules predicting crashes.

We tried minimum correlations of 0.4, 0.3, and 0.2. In some cases we could get successful runs with a minimum correlation of 0.1, but in other cases we ran out of memory on a machine with 1 GB of RAM. We set a minimum p-value for the t-test of .0005. In our initial tests to figure out where in the parameter space to run these experiments we found negligible dependence on the minimum p-value. We conducted ANOVAs to analyze the affects of tuning on misclassification rate and coverage.

We then tested Association Trees both with and without adaptive sampling on the 14 different faulty versions of JABA having failure rates above 8%, and compared what we considered the best tuning for each version. The goal of this analysis was to explore how Association Trees would work for detecting bugs for a variety of programs. For both methods, we observed 100 predictors per observation. With adaptive sampling, we observed blocks of a random size (following a Poisson(12) distribution), where the first half went into the training data, and the second half into the test data.

5.1 Results

The analysis of the tuning parameters detected many high-order interactions, so presenting the results of an ANOVA would barely compress the data. In Tables 2 and 3, we show the estimates from running an ANOVA with no interactions to provide a coarse picture of the effects of the various tuning parameters. We focused on two responses: **coverage**, which we define as 1-P("can't predict"), and the **misclassification rate**, which we define as the proportion of predicted crashes/successes that are incorrect, ignoring the predicted responses of "can't predict." In some of our studies, we further refined the misclassification rate, looking at both the **false positive** and **false negative** rates. We observed coverages ranging from around 8% to over 90%. In general, the higher the coverage, the higher the misclassification rate. The best coverage settings had error rates from 2-4%, whereas the best error rates (perfect classification) had coverage rates below 10%. We had two preferred settings for the compromise between accuracy and coverage. In one of our preferred compromise values, we had 80% coverage and a 1% misclassification rate with 375 samples of blocks of 24 in both the training and the test set, 100 predictors observed, a minimum correlation of 0.2 between each predictor used and class membership, and a support for the rules predicting successful execution of the input program of .0016 (.0033 had 73% coverage and .7% misclassification). In the other, we had 50% coverage and a 0.3% misclassification rate with 1500 samples of blocks of 6 in both the training and test set, 100 predictors observed, a minimum correlation of 0.3, and supports for the rules predicting successful executions of the input programs of .0033 and .0016. We have summarized these results in Table 4.

These settings generalize across similar data. Another version of JABA with a 15% failure rate was tested, with a minimum correlation of 0.2, and supports of .0016 and .001 for successes and

Parameter	Estimate	t-ratio
intercept	0.0078	19
MinCor = 0.2	0.00056	0.87
MinCor = 0.3	-0.003	-4.74
nblocks = 750	-0.000077	-0.14
nblocks = 1500	0.0004	0.80
npredictors = 100	0.0025	6.52
SupportSuccess = .0016	0.003	5.69
SupportSuccess = .0033	-0.0006	-1.16
SupportCrash = .0005	0.001	3.14

Table 2: Main effects from model on one version with no interactions for misclassification rate - $R^2 = 0.62$

crashes respectively, a block size of 24, and 375 cases for both training and testing. This run found so many rules that the processing computer ran out of RAM, so the minimum correlations of 0.3 and 0.4 were tested. Surprisingly, the minimum correlation of 0.3 had the better coverage and misclassification rates, 94% and 1.5% respectively, as opposed to 89% and 2.1%.

For some cases, Association Trees had significantly more difficulty. One version of JABA we looked at had a failure rate of 52.6%. With supports of .0033 for successful executions, and .001 for crashes, we had a coverage of 81%, and a misclassification rate of 6.5%.

The coarsest tuning parameter is the minimum correlation needed between the predictors and the response to look for a split. When it is set too low (0.1 in the case of the main experiment), so many rules are found that even controlling for multiple testing with Bonferroni's correction, R will crash while trying to find rules, or take too long to evaluate the rules found. When it is set too high, too few rules will be found. In our experiment, both 0.2 and 0.3 gave good results. Since values which are too high lead to too few rules being found, and the code executing very quickly, we recommend that when trying to select a reasonable value, when in doubt start searching from the high end of the range under consideration.

The support for the majority class had a major affect on the coverage probability, whereas the support for the minority class had a much smaller impact. We think this is because with a lower support, we find more rules for predicting the dominant class, and can then make a decision about more of the cases we see in the future. The main effect of the support of the minority class (crashes) was to decrease the rate of false negatives, and increase the rate of false positives.

In a couple of cases there was a large increase in the coverage probability with changes in the support of the majority class. With 100 columns sampled, and 750 blocks of 12 rows in both the training and the test set, with a minimum correlation of 0.2, going from a support of .0066 to .0033 raised the coverage from 40% to 73%. Similarly, with a 50 columns sampled, 1500 blocks of 6, and the same minimum correlation, there was a jump when the support of the majority class went from .0033 to .0016 from 9% to 34% in the coverage probability. In all four cases, there was no significant change when the support of the minority class doubled from .0005 to .001. While these are extreme cases, patterns like this can be found throughout the data.

The number of predictors measured for each observation has a large impact in the quality of the classifier. This effect remained even when we tried sampling 50 columns by 18,000 rows for each of the training and test sets. This is not a surprise. For an extreme example, consider a situation where there are 1,000 measured predictors, and only the first one has any predictive power. If we measure 50 predictors at random for each observation, then we have a 1 in 20 chance of being able to make an accurate prediction with an optimal classifier. If we measure 100 predictors, then the chance increases to 1 in 10. Our findings are consistent with this, as sampling more observations and

Parameter	Estimate	t-ratio
intercept	0.37	29
MinCor = 0.2	0.058	2.87
MinCor = 0.3	-0.0015	-0.07
nblocks = 750	0.046	2.72
nblocks = 1500	-0.02	-1.33
npredictors = 100	.15	12.65
SupportSuccess = .0016	0.13	7.72
SupportSuccess = .0033	0.0065	0.38
SupportCrash = .0005	0.0008	0.07

Table 3: Main effects from model on one version with no interactions for coverage rate - $R^2 = 0.74$

Block-size	Number of Blocks	Min. Sup. Success	Min. Sup. Crash	Min. Cor.	Coverage	Misclassification
24	375	0.0016	0.001	0.2	80%	1%
24	375	0.0033	0.001	0.2	73%	0.7%
6	1500	0.0016	0.001	0.3	50%	0.3%
6	1500	0.0033	0.001	0.3	50%	0.3%

Table 4: Summary of good runs. All had 100 predictors sampled.

having 50 measured predictors per observation gives coverage similar to sampling a smaller number of observation, but slightly better classification rates.

Block size did not appear to be influential. We found results of every quality with every block size. However, this too should not be surprising, for if most of our rules are main effects or two-way interactions, then when we sample between 750 and 3000 blocks, and each block has between 1/12th and 1/24th of the total number of possible predictors in it, one could expect to have good coverage by chance.

5.2 Affects of Adaptive Sampling

Figure 1 depicts the results obtained for the Association Trees without adaptive sampling (hereafter called "fixed sampling Association Trees") side by side with those for the adaptive sampling Association Trees. To perform this comparison as fairly as possible, we selected what we judged as the best fixed sampling Association Trees results for each version. This is necessary because our fixed sampling Association Trees study included models built with non-optimal parameter settings. Note, however, that since we have four different performance measures, the definition of best is necessarily subjective. In this case, we have preferred models with higher coverage and lower false negatives because software developers are often more interested in identifying failing runs than passing runs. As shown in the figure, by sampling adaptively we generally achieved higher coverage and lower misclassification rates, with significantly less variability between versions. Note the outlier in coverage for the fixed sampling Association Trees.

For the experiments using adaptive sampling association trees, we varied the supports for both the crashes and the successful executions, and used a stochastic (Poisson(10)) block size. We sampled 100 predictors, used minimum confidences of 1, the same maximum p-value, and used a minimum correlation of 0.3.

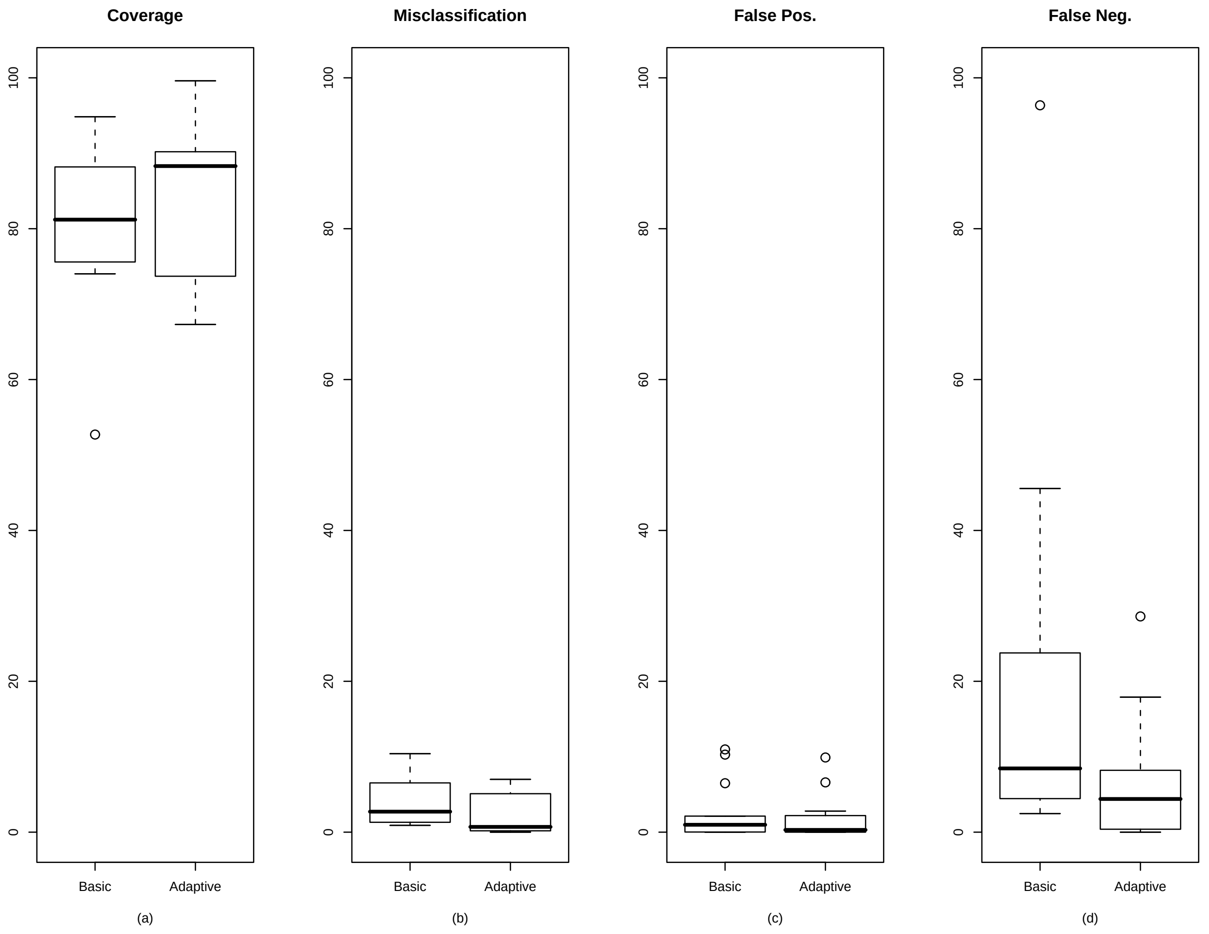

Figure 1: Basic Association Trees vs. Association Trees with adaptive sampling.

6. Discussion

We have presented here a new method for fitting decision trees, that performs better than CART when faced with large amounts of missing data. Our method requires the strong assumption that the whether a predictor is missing is independent of the observed outcome. We also have presented an adaptive sampling scheme which can greatly increase the performance of the Association Trees if the data is sampled sequentially.

When implemented in R, our method is expensive in both computational time and space. A major limitation for the amount of data that we considered in our experiments was with the memory of the machines we were using (1 GB).

We did not use a standard algorithm to fit decision trees because they do not work very well when the vast majority of the observations are missing. Since CART has an error rate of 8.5% on the *training* data, another method is needed. Our algorithm suffers from not being able to find high-way interactions. But the nature of the data we are dealing with precludes finding these rules, since we do not always have the same set of variables instrumented. For instance, if we observe 1/10 of our predictors with each observation, it will take us 1000 observations to see all 3-way interactions. Considering how many observations it takes to observe events we want to predict, it is impractical to expect to find rules that require more than 2- or 3-way interactions with the amount of data we are able to handle.

Acknowledgments

This research was supported by NSF grants CCR-0205422 to the Georgia Institute of Technology, CCR–0205118 to the University of Maryland and CCR–0218759 to the National Institute of Statistical Sciences. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Agrawal, R. and Srikant, R. 1994. Fast Algorithms for Mining Association Rules in Large Databases. 20th Conference on Very Large Databases, Santiago, Chile, 1994: 487-499

[2] Breiman, L., Friedman, J., Stone,C., and Olshen, R. 1984. *Classification and Regression Trees.* Wadsworth.

[3] Breiman, L. 2001. Random Forests. *Machine Learning* **45**, No. 1., 5-32

[4] Boros, E., Hammer, P., Ibaraki, T., Kogan, A., Mayoraz, E., and Muchnik, I. 2000. An Implementation of Logical Analysis of Data. *IEEE Transactions on Knowledge and Data Engineering*, **12**, No. 2. 292-306

[5] Brin, S., Rastogi, R., and Shim, Kyuseok. 2003. Mining Optimized Gain Rules for Numeric Attributes. Knowledge Discovery and Data Mining, *IEEE Transactions on Knowledge and Data Engineering*, **15**, No. 2, 324-338

[6] Haran, M., Karr, A., Orso, A., Porter, A., and Sanil, A. 2005. Applying Classification Techniques to Remotely-Collected Program Execution Data. *ESEC/FSE-13: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering.* New York, NY: ACM Press, 146-155

[7] Hastie, T., Tibshirani, R., and Friedman, J. 2001. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer-Verlag, New York, New York.

[8] Fukuda, T., Morimoto, Y., Morishita, S. and Tokuyama, T. 1996. Mining Optimized Association Rules for Numeric Attributes. *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 182-191. ACM Press.