USING CONVEX PSEUDO-DATA TO INCREASE PREDICTION ACCURACY

Leo Breiman
Statistics Department
University of California
Berkeley, CA 94720
leo@stat.berkeley.edu

ABSTRACT

A prediction algorithm is consistent if given a large enough sample of instances from the underlying distribution, it can achieve nearly optimal generalization accuracy. In practice, the training set is finite and does not give an adequate representation of the underlying distribution. Our work is based on a simple method for generating additional data from the existing data. Using this new data (convex pseudo-data) it is shown empirically that on a variety of data sets prediction accuracy of an algorithm can be significantly improved. This is shown first in classification using the CART algorithm. Similar results are shown in regression. Then pseudo-data is applied to bagging CART. Although CART is being used as a test bed, the idea of generating convex psuedo-data can be applied to any prediction method.

1. Convex Pseudo-Data

Given a training set $T=\{(y_n,\mathbf{x}_n), n=1, \ldots, N\}$ where the y are either class labels or numerical values and the **x** are M-dimensional input vectors, consider an algorithm that uses T to construct a predictor h(**x**) of future y-values given the input values **x**. If the algorithm is consistent i.e. CART, C4.5, Neural Nets, then the larger the size N of the training set, the smaller the generalization error.

One dream, assuming the training set consists of independent draws from the same underlying distribution P(dy,d**x**) is that we can keep drawing as long as we want--constructing a large training set. But generally, we are confined to a training set of given size. One way around this restriction is to try and manufacture new data from the given training set.

Another approach is to use T to estimate P(dy,d**x**) (Shang and Breiman[1996b]). Using the estimated probability distribution, an infinite number of instances can be generated. This method gave promising results but ran into serious technical difficulties. Kernel density estimates were used for the numerical estimates, and ad hoc methods for categorical values. Numerous parameters had to be estimated to plug into the probability estimate, and the result was an complex and unwieldy procedure.

Our method for generating new data from the old is relatively simple and depends only on a single parameter d, $0<d<1$. To create a new data instance, these steps are followed:

I) Select two instances (y,**x**) , (y',**x**') at random from the training set.
ii) Select a random number v from the interval [0,d], and let u=1-v.
iii) The new instance is (y'',**x**'') where y''=y, and **x**''=u**x**+v**x'.**

If **x** contains some categorical values, then iii) needs to be revised as follows: if x_m is the value of a categorical variable in **x**, and x'_m its value in **x'**, then x''_m equals x_m with probability u and x'_m with probability v.

In general, no attempt is made to optimize over all values of d. The generalization error is estimated only at 2-3 values of d, ands the best one of these is selected. In spite of the simplicity of this method, significant reductions in error are realized.

2. Application to CART Classification

CART trees are grown using convex pseudo-data (CPD) as follows: pseudo-data is generated and sent down the current tree until the particular unsplit node being studied has N instances in it--recall that there are N instances in the original training set. Then this node is optimally split using the Gini criterion. If either of the two children nodes contain no member of the original training set, then the parent node is declared terminal.

After a large tree is grown this way, then 1000*N instances of pseudo data are generated and passed down the tree. This data results in a cost estimate for every node, and using these cost estimates a sequence of pruned trees is computed. Tree selection is done as per CART methodology. The program was run on 9 data sets of which one (wave) used synthetic data. For wave, in each iteration, a new 300 instance training and 3000 instance test set were generated. Brief descriptions of the data sets are given in Table 1. They are all in the UCI repository.

Table 1 Data Set Summaries

Data Set	Size	No. Inputs	No. Classes
sonar	208	60	2
glass	214	9	6
breast(Wis)	699	9	2
ionosphere	351	34	2
soybean	683	35	19
vehicle	846	18	4
vowel	990	10	11
wave	300	21	3
dna	3186	60	3

With the exception of the dna data set, all inputs are real. Each of the 60 inputs for dna is a four-valued categorical.

Estimates of generalization error on various data sets were gotten as follows: in each one of 50 iterations, a random 10% of the data was set aside and two trees were grown on the remaining 90%. One was the usual CART tree. The other was a tree grown using pseudo-data as described above (CPD-CART). Then the 10% test set was passed down each tree and associated a test set error with each value of the cost parameter. For each set of 50 trees, the value of the cost parameter was found that minimized the average test set error. This latter average is reported in Table 2.

Both the CART trees and the pseudo-data trees were grown using exactly the same data and the same method of estimating generalization error. This minimized the variability of extraneous factors. Three values of d were tried, .10,.25 and .50. The results in Table 2 below reports the results using the most effective of the three values.

Table 2 Estimated Generalization Error (%)

Data Set	CART	CPD-CART	% Reduction	Best d
sonar	27.5	22.0	20.0	.25
glass*	29.8	27.0	9.7	.25
breast (Wis.)	5.9	4.7	20.0	.25
ionosphere	11.1	9.4	15.3	.25
soybean	8.3	7.2	13.3	.10
vehicle	19.4	17.9	7.7	.10
vowel	19.4	15.6	19.6	.25
wave	28.6	27.2	4.9	.50
dna	5.5	4.5	18.2	.50

* On the initial run on the glass data, there was a 1.3% reduction in error, but we noticed that there some terminal nodes in the maximal trees that contained relatively large numbers of the original training set. On the rerun, the size of terminal nodes was constraining to hold 3 or fewer of the original training set. The number in the table is for the rerun. Constraining the size of the terminal nodes did not make much difference on the other data sets.

It has proven difficult in the past to improve on CART's performance inside the single decision tree context. So an interesting question is what accounts for the significantly improved accuracies reflected in Table 1. In particular, did the improved accuracy result in much larger trees? We computed the number of terminal nodes in each of the 50 trees corresponding to the optimal value of the cost parameter and averaged. The result is given in table 2.

Table 2 Average Number of Terminal Nodes

Data Set	CART	CPD-CART
sonar	12.4	19.0
glass	9.7	37.8
breast (Wis.)	16.6	11.2
ionosphere	17.5	20.8
soybean	53.7	52.5
vehicle	90.4	101.6
vowel	137.8	149.3
wave	13.1	28.3
dna	13.6	34.2

Table 2 shows that using CPD-CART generally resulted in larger trees. But the story is not universal. In two of the data sets accuracy was improved using smaller trees. In the other data sets, except for the glass data, the trees are not much larger. This would indicate that most of the increased accuracy is due to the better splits given by use of the psuedo-data.

3. Application to CART Regression

In regression, convex pseudo-data is generated in a way similar to classification i.e. to generate a new instance

i) Select two instances (y,$\boldsymbol{x}$) , (y',$\boldsymbol{x}$') at random from the training set.
ii) Select a random number v from the interval [0,d], and let u=1-v.
iii) The new instance is (y'',$\boldsymbol{x}$'') where y''=uy+vy', and $\boldsymbol{x}$''=u$\boldsymbol{x}$+v$\boldsymbol{x}$'.

The only difference is in the 3rd step. Our experiments were conducted on the data sets summarized in Table 3.

Table 3 Data Set Summaries

Data Set	Size	No. Inputs
Ozone	330	8
Housing	506	12
Servo	167	4
Friedman #1	200	10
Friedman #2	200	4
Friedman #3	200	4

The Servo data set has two 5-valued categorical inputs. All other inputs are numerical. The last three data sets are synthetic, and are fully described in Breiman[1996a]. In each of 50 runson the synthetic data a freshly generated training and a 2000 instance test set was used. In the real data sets, 10% was set aside as a test set in each of the 50 runs. As in the classification runs, pruning and minimal cost estimation was done via CART methodology. The d values .25,.50,.75,1.00 were tried and the best reported in Table 4.

Table 4 Estimated Mean Squared Generalization Error

Data Set	CART	CPD-CART	% Reduction	Best d
Ozone	23.4	19.9	15.0	.50
Housing	18.2	15.7	13.7	.50
Servo*	98.4	41.4	69.8	.50
Friedman #1	12.0	8.5	29.0	.75
Friedman #2	30.6	21.5	29.7	.75
Friedman #3	41.9	36.7	12.4	1.00

* on the initial run on servo, there was about 40% reduction in error, but just as with the glass data, some terminal nodes in the maximal trees contained relatively large numbers of the original training set. On the rerun, the size of terminal nodes was constraining to hold 3 or fewer of the original training set. The number in the table is for the rerun.

Table 5 Average Number of Terminal Nodes

Data Set	CART	CPD-CART
Ozone	10.2	26.6
Housing	21.2	80.4
Servo	2.0	39.4
Friedman#1	14.1	142.5
Friedman#2	16.8	61.4
Friedman#3	23.7	117.0

The differences in tree sizes is large. But considerably smaller trees can be grown without too much of a loss in accuracy. For example, in pruning up the CPD trees grown for the Friedman #1 data, an increase in 10% over the minimum test set error was allowed. This cut the percent error decrease (compared to CART) from 29% to 22%. The number of terminal nodes decreased from 142 to 51.

4. Application to Bagging

In bagging, altered training sets of the same size as the original are constructed by random sampling with replacement from the original. Then a classifier is grown on each of the altered training sets and the ensemble of classifiers then votes to predict the class of any new input. Breiman [1996a],[1997] pointed out that the bootstrap training sets were imitations of independent replicas of the training sets--each replica consisting of N instances drawn from the same underlying distribution. That is, the underlying distribution is approximated by the bootstrap distribution which places probability 1/N at each of the N training instances.

If a better approximation to the underlying distribution was available, then sampling from it to form the new training set will give lower generalization error. This was shown in Breiman[1997] where kernel density estimates were used to approximate some simple multivariate Gaussian distributions. This type of approximation does not easily generalize to more complex data. But we can try using convex psuedo-data.

In the following experiments, the altered training sets consisted of N convex pseudo-data instances. In each run 10% of the data was held out, 100 altered data sets constructed using the other 90% and a CART tree grown on each. Then the accuracy was checked on the 10% test set. Fifty runs were done, and the test set results averaged. There were 6 potential values of d; .05,.1,.25,.5,.75,1.0. Usually, only 3-5 of these were tried. Because as soon as a value of d was found at which the test set error was less than that at the two adjacent d values, it was adopted. The results are compared to straight bagging in Table 5 which also gives the best d value.

Table 5. Test Set Error Rates(%)--Bagging Classifiers

Data Set	Bagging	CPD-bagging	Best d
sonar	22.2	17.8	.75
glass	23.1	22.1	.10
breast (Wis)	3.8	3.4	.25
ionosphere	8.5	6.7	.10
soybean	6.2	5.9	.10
vehicle	15.6	15.2	.10
vowel	8.0	7.2	.10
dna	5.1	4.0	.75
wave	19.3	17.7	.75

Table 5 shows that substantial gains in accuracy can be made by using convex pseudo-data in bagging classifiers. But, oddly enough, the same is not true in regression. These latter results are given in Table 6.

Table 6 Mean-squared Test Error--Bagging Regressions

Data Set	Bagging	CPD-bagging	Best d
Ozone	19.8	18.0	.50
Housing	10.9	11.0	.05
Servo	82.8	81.3	.10
Friedman #1	63.2-1	62.7-1	.05
Friedman #2	21.7-2	19.7-2	.50
Friedman #3	24.7+2	25.8+2	.05

In two of the data sets, using convex pseudo-data increased the error. In the remaining four, there were small decreases.

5. A Simple Experiment

To get more insight into how CPD works, we examine a single input variable with values $\{x_n\}$. Suppose the density of the distribution underlying the $\{x_n\}$ is estimated using a kernel density estimator. Then this density is sampled from by first sampling one of the $\{x_n\}$, then by sampling from the kernel, and adding the two together. If we sample repeatedly many times, then we can think of each x_n as a seed, surrounded by many other values whose differences from x_n are distributed according to the kernel.

Now sample repeatedly from ux+vx' where x,x' are independently drawn from the set $\{x_n\}$, v is drawn from the uniform distribution on [0,d] and u=1-v. Each time $x=x_n$, there is a scattering of samples around it with the differences from x_n drawn from the distribution of $v(x'-x_n)$. Denote the mean of the $\{x_n\}$ by $\bar{x}$ and their variance by $\hat{\sigma}^2$. Then the scatter about x_n has mean $(d/2)(\bar{x}-x_n)$ and variance $(d^2/3)(\hat{\sigma}^2+(\bar{x}-x_n)^2/4)$. The scatter has a density that is asymmetric about x_n with a sharp peak at zero. The probability that the scatter around x_n will go to the right of x_n equals the proportion of the x-values that are less than or equal to x_n.

There are two apparent problems with these CPD kernels:

1) The mean and variance of the kernel depend on x_n, whereas in a kernel density estimate they are constant. To compare CPD to the use of a standard kernel density estimate, we sample repeatedly from $x+d\hat{\sigma}z$, where z is a normal N(0,1) variable.. Then the scatter at any seed is a Gaussian with mean zero and variance $d^2\hat{\sigma}^2$. Call this kernel data.

2) The variance of the CPD kernel depends on the variance of all of the data regardless of class. It would be desirable to have the variance of the kernel for the data in a given class depend only on the variance of the data in that class. This can be done as follows: say our first pick of x is in class j, then sample at random from the other class j instances to get x' and let the x-value of the new instance be ux+vx'. Then for x_n in class j, the mean and variance for the scatter around x_n is $(d/2)(\bar{x}_j-x_n)$ and $(d^2/3)(\hat{\sigma}_j^2+(\bar{x}_j-x_n)^2/4)$ where $\bar{x}_j$ and $\hat{\sigma}_j^2$ are the mean and variance of the class j x-data. The data generated this way we call convex within-class data..

Here is a simple experiment that was conducted to compare these different sampling methods: All data consisted of two classes with one input variable and equal apriori probability for both class Two distributions were used--two normals with variances equal to 1,1: two normals with variances equal to 1,4. Their separation was determined so that the Bayes error was 2.28%, i.e. for variances equal to 1, the distance between means was 4. We used sample sizes 5,10,25,50,100 for the training sets, with the restriction that any acceptable training set had to have at least two instances of each class.

In each of 200 iterations, a training set was generated together with a test set of size 10000. Using the Gini criterion on the training set, the best single cut of the data was found, and the test set used to estimate the generalization error given by this cut. Then 5000 instances of pseudo-data were generated from the training set, Gini again used to find the optimum cut on the pseudo data, and the same test set run down this cut. The test set errors for the 200 iterations were averaged and the percent reduction resulting from the use of pseudo-data computed.

For each distribution and each method of generating pseudo-data (but not sample size) we choose the value of d from the set {.05,.10,.25,.50,75,1.00} that gave the best performance on sample size 100. The test set results are summarized in Table 7 where the numbers in parentheses are the best value of d.

Table 7 Estimated Reduction in Generalization Error (%)

Method	Convex	Kernel	Convex Within-Class
Distribution #1	(.50)	(.25)	(.75)
ss.100	15.2	13.1	6.1
ss.50	19.4	10.0	4.4
ss 25	20.7	9.7	2.4
ss 10	13.8	2.8	1.2
ss 5	19.3	0.1	2.0
distribution #2	(.25)	(.10)	(.50)
ss 100	3.6	6.4	3.7
ss 50	11.9	4.5	3.7
ss 25	12.7	4.4	-.5
ss 10	7.7	1.0	0.4
ss 5	8.2	0.8	0.8

It is surprising how well use of convex pseudo-data holds up even at small sample sizes, where the kernel and within-class convex loss advantage quickly. This experiment is small, but since CART makes use of univariate splits only, the increased accuracy of CPD shown in the table may be reflected in the increased accuracy of trees built using CPD. This leaves unsolved the problem of what gives the CPD kernel its increased accuracy as compared, say, to a Gaussian kernel. I have not come up with a satisfactory explanation yet.

6 . Discussion

After arriving at the method described above and the encouraging results tabled, I tried some other ideas that did not work out--that is, did not result in any significant increases in accuracy:

i) Both in regression and classification, take y''=y with probability u and equal to y' with probability v.

ii) Generate a random number z. If z<u, set all categorical x_m'' equal to x_m, otherwise to x_m'.

iii) In regression set y''=y instead of a convex combination of y and y'

iv) Letting the number of instances used to split each node be larger--say a multiple of N.

Using CPD does not guarantee improvement. In particular, it does not seem to give improved accuracy on large data bases. I tried it on the letters, satellite and zip code data without success. These contain 15,000, 4,435, and 7,921 instances respectively. Failure may be due to the fact that the data is already packed so tight in these data sets that augmenting it with pseudo-data is not helpful.

Still, given that efforts over the past 15 years have succeeded in producing trees only slightly more accurate than the basic CART trees, use of pseudo-data seems a promising new possibility.

References

Breiman, L. [1997] Bias, Variance, and Arcing Classifiers, Technical Report 460, Statistics Department, University of California (available at www.stat.berkeley.edu)

Breiman, L. [1996a] Bagging predictors , Machine Learning 26, No. 2, pp. 123-140

Shang, N and Breiman, L.[1996b]Distribution Based Trees are More Accurate (with), Proceedings ICONIP,September, 1996, Hong Kong.

APPENDIX Programming Notes

Building a CPD-tree takes considerably more computing time than ordinary tree building. The additional time is in filling the unsplit nodes with N instances of pseudo-data. In a deep tree, as pseudo-data is generated, only a very small fraction may drop into the unsplit node under scrutiny. The direct approach of computing pseudo-data and continuing until the node is filled is cycle intensive.

This computation is cut down by an order of magnitude using the following device: As the data is being generated to fill the current unsplit node, those instances that miss the current node have two destinations. They can either wind up in nodes already designated as terminal or in other, as yet unsplit, nodes. Our program stores the data falling into the other unsplit nodes up to a maximum of N instances in each unsplit node. Then when these nodes come under scrutiny, they are often filled or almost filled with data.

The maximum number of unsplit nodes that data generation will fill is specified in advance. The usual default is 50. Another piece of code in our program functions as a safety valve. If the current unsplit node cannot be filled after 1000 N additional instances are generated, the node is declared terminal, and we move on to the next unsplit node.

To handle larger data sets, two other modifications were made. For the node under scrutiny, let K be the number of instances of the original training set falling into the node. A multiplicative factor M is preset and the number of pseudo-data instances required for the node is min(N,MK). Our usual default for M has been either 50 or 100. This serves to limit the amount of data required in very small nodes. The other modification is that memory may not be available to store 50 data sets of size N that fall into the unsplit nodes. Therefore, since the problem is filling the smaller nodes, only data up to N_O<N instances have space reserved for them in the unsplit nodes. When MK gets to be smaller, on average, than N_O a large portion of the nodes are already filled or nearly filled.