Computational Statistics & Data Analysis 40 (2002) 27–37

COMPUTATIONAL STATISTICS & DATA ANALYSIS

www.elsevier.com/locate/csda

Using k-nearest-neighbor classification in the leaves of a tree

Samuel E. Buttrey*, Ciril Karo

Department of Operational Research OR/Sb, Naval Postgraduate School, Monterey CA 93943, USA

Received 1 May 2001; accepted 1 October 2001

Abstract

We construct a hybrid (composite) classifier by combining two classifiers in common use—classification trees and k-nearest-neighbor (k-NN). In our scheme we divide the feature space up by a classification tree, and then classify test set items using the k-NN rule just among those training items in the same leaf as the test item. This reduces somewhat the computational load associated with k-NN, and it produces a classification rule that performs better than either trees or the usual k-NN in a number of well-known data sets. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Nearest neighbors; Classification; Hybrid classifiers

1. Introduction

In the general classification problem, we are given a training set of, say, n items, on which we have p measurements each, with measurement j for observation i being denoted by x_{ij}, $i = 1, \ldots, n$; $j = 1, \ldots, p$. We also have the class to which each observation belongs, denoted y_i where y_i takes on one of a (typically) small set of labels. For example, the (row) vector $\mathbf{x}_i$ might be a set of pixel brightness measurements for a particular hand-written digit, say, after the image is digitized, and y_i the actual digit that was written. The object is to predict the class memberships of a new item, or generally a whole set of items (the "test set"), for which the $\mathbf{x}$'s are given. For this paper, we will assume that each of the measurements is continuous rather

* Corresponding author.
E-mail address: buttrey@nps.navy.mil (S.E. Buttrey).

PII: S0167-9473(01)00098-6

than categorical. We will denote the column vector of measurements for variable j as $\mathbf{x}_{[j]} = (x_{1j}, x_{2j}, \ldots, x_{nj})$.

In this paper, we describe a new classification technique that is a hybrid of two well-known approaches: the classification tree and k-nearest-neighbors (k-NN). Specifically we fit a classification tree, and then apply k-NN within the leaves of the tree. That is, a test set item that falls into a particular leaf of the tree is classified by k-NN; only training set items that fall into that leaf are considered as possible neighbors.

The remainder of this paper is organized in this way: the rest of this section describes the two classification techniques we are combining, and the way in which we have combined them. In Section 2, we describe the data we use and discuss the estimation of misclassification rates. Section 3 gives the results of our technique, and Section 4 offers some conclusions and ideas for further refinement.

1.1. *k-nearest-neighbor classification*

The well-known k-NN approach to classification has proven successful in many applications. In this method, we measure the distance from a test set item to each of the training set items, noting the k training set items that are nearest. We then classify the test set item by whichever class is most common among those k "nearest neighbors," letting each neighbor "vote." (In case of ties, we have chosen to include all training set items no farther away than the kth nearest neighbor, so in this case there will be more than k voters.) A number of investigators have considered the question of how best to measure distance: approaches have included global metrics (Fukunaga and Flick, 1984), local metrics (Short and Fukunaga, 1980), metrics that are specific to the problem (Simard et al., 1993) and so on. By far the most common metric, though, has been Euclidean distance, under which the distance between two points $\mathbf{x}_r$ and $\mathbf{x}_s$, say, is given by the square root of the (possibly weighted) sum of the squared distances over each co-ordinate. Although generalizations are possible, we use the simple form:

$$d(\mathbf{x}_r, \mathbf{x}_s) = \left[\sum_{i=1}^{p} c_i (x_{ri} - x_{si})^2 \right]^{1/2}. \tag{1}$$

In ordinary Euclidean distance, the weights $c_i, i = 1, \ldots, p$ are all equal to 1. However, experience suggests that two related steps can improve classification accuracy. First, we might expect some of the measurements to be irrelevant to the problem. Naturally we hope to be able to give weights of zero to these irrelevant columns. This echoes, of course, the variable selection problem that appears in almost every statistical model.

Second, relevant variables may be measuring similar quantities on quite different scales. Under these circumstances, it seems obvious that reducing each of the variables to a common scale may help k-NN classification by preventing one of the measurements from dominating all the others. A third problem that needs to be tackled is that of selecting the best value of k, the number of neighbors to be considered.

In our approach, we attack the choices of k and of which variables to include by using a stepwise approach. Our implementation permits either forward or backward selection; for reasons of speed and parsimony we usually use the former. In this scheme, we start with every variable out of the model, plus a vector of possible values of k. Currently, this set is not chosen by reference to the data; we merely use the set $1, 3, 5, \ldots, 31$ since this has a "reasonably large" range. We use leave-one-out cross-validation to estimate the misclassification rate of the classifier for each choice of k. That is, each element of the training set is classified by all the others, using the current set of variables in the model and the entire vector of k's. Of course, at the very beginning of this process when every variable is "out" of the model, every training set item is equidistant from every test set item, and regardless of k, every training item gets to vote. This gives us the so-called "naïve classifier," in which every test set item is simply given the most frequent training set classification.

Then one of the variables is added to the model and a new set of misclassification rates, one for each k, is computed. This is done for each variable in turn. At the end of this process we choose the combination of k and added variable that produces the lowest misclassification rate. If no addition produces an improvement then the process is finished; the current set of variables and the best k are used. If the addition of a variable produces a misclassification rate strictly better than that of the current set, then that variable is added to the current set and the process continues. Our approach, as with other "greedy" algorithms, is reasonable but not guaranteed to produce an optimal set of variables. Since we require strict improvement at each stage we expect our routine to be resistant to the presence of "noise" variables, and we have seen this in an example (see "noise resistance" below).

Finally, we perform this entire stepwise routine twice: once with the data in its original form and once with each column scaled to have standard deviation equal to 1. (We have also looked at scaling by median absolute deviation but in our examples we have found it makes little difference.) Of course we choose whichever of the scaled and unscaled results has the lower misclassification rate. Thus when we talk about k-NN classification, we are referring to a process that does all three of (i) variable selection; (ii) choice of k; and (iii) choice of using scaled or unscaled data.

1.2. Tree classification

The classification tree technique (Breiman et al., 1984) divides the training set into a number of mutually exclusive subsets. At any point the tree program chooses a "split" which divides the data into two pieces, based on the value of one of the columns of $\mathbf{x}$. This produces two subsets, which may or may not then be split further. (There are other implementations, but in ours, every split is of the form $\mathbf{x}_{[j]} < t$.) Continuing the tree metaphor, the final subsets produced by this process are called "leaves." We observe that some of leaves are relatively "pure" (that is, they consist of observations primarily of one class); others are more mixed. A tree classifies a new observation by the most common class among observations in the leaf into which that item falls, so most classification errors will occur in these mixed leaves.

Each split is made so as to maximize the change in some splitting criterion. There are several criteria in common use; since we are using the S-Plus statistical package, we use its default, which is the multinomial deviance, that is, -2 times the log-likelihood under a multinomial model. If there are n items in a node, of which n_j are in class j, the deviance is then $-2\sum_j n_j \log(n_j/n)$, taking $0 \log 0 = 0$.

At each node, the tree program finds the split which reduces the total deviance the most and implements it. Splitting continues until some stopping rule is met (for example, by default the program will not split a node with fewer than ten entries). The stopping rules are constructed so that the resulting trees are typically over-fit. Then cross-validation is used to find the "optimal" size. (The size of a tree is simply the number of leaves it has.) So suppose that the original tree has m leaves. Then trees of every size from 1 to m are constructed with 90% of the data, and the numbers of errors they make when classifying the remaining 10% are stored. This process is repeated nine more times until each 10% of the data has been used as the test set. Finally, we choose the "optimal" size with the "one SE rule" (Breiman et al., 1984). We identify the minimum misclassification rate among all the tree sizes, say r^*. Then looking at classifications as Bernoulli events, we choose as "optimal" the smallest size for which the misclassification rates is no more than one estimated standard error above this minimum. That is, we choose the smallest size whose misclassification rate is smaller than $r^* + (r^*(1-r^*)/n)^{1/2}$.

1.3. Combining k-NN and tree models

Our approach is to apply k-NN classification separately within every leaf of a tree. That is, first we build a classification tree (whose size will be determined below). Then, given a test item we (i) determine the leaf into which the item falls; and (ii) perform k-NN classification using only the training set items that fall into that leaf. By restricting the population of training set items to those within the same leaf as the test set item, we reduce the computational burden.

Furthermore, it is often the case that the structure of the tree is too "coarse," that is, that the optimally sized tree produces leaves inside of which there is still some information that can be exploited. As a simple example consider a population with two variables distributed uniformly on the vertices of the unit square, so that $\mathbf{x}_{[1]} \in \{0,1\}$ and $\mathbf{x}_{[2]} \in \{0,1\}$. Suppose that all items at (0,0) and (1,1) belong to class A and that all items at (0, 1) and (1, 0) belong to class B, and suppose that each vertex has exactly the same number of observations. A leaf with this population will have an error rate of 0.50. The tree method will consider the two splits $\mathbf{x}_{[1]} < 0.5$ and $\mathbf{x}_{[2]} < 0.5$, and since neither of those produce a decrease in deviance, no split will be made. However, assuming at least one training set item at each of the four vertices, the 1-NN classifier with have an error rate of 0. This is the sort of structure we hope will be visible to the k-NN classifier. The performance of a global k-NN classifier is harmed by heterogeneity in the data: the tree, by dividing the training set into more-homogeneous subsets, helps protect against this.

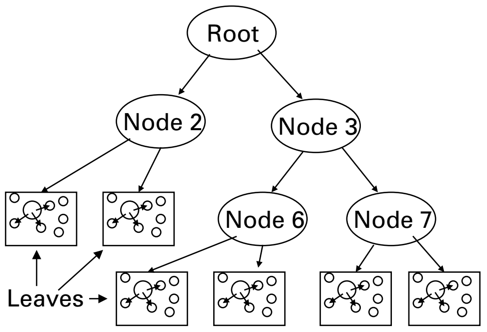

Fig. 1. Schematic of k-NN classification inside leaves. The tree is constructed in the usual way. Then a separate k-NN classifier is produced in each leaf. Each k-NN classifier may have a different k, a different set of included variables, and a different choice of scaling.

Fig. 1 shows a schematic of our process. The usual tree procedure divides the “root” node into two child nodes. The left is divided further into two leaves; then k-NN classification is performed separately inside each of those leaves. (That is, only items falling into that leaf are considered as possible “neighbors.”) Similarly, the right side is itself divided into two nodes; one of these is a leaf and supports k-NN; the other is divided into two more leaves and in those, too, k-NN is performed.

In the ordinary tree model, the best size of the tree (call it s_{OPT}) is chosen by cross-validation. It is not clear that this size should be best for this work, but it seems intuitive that the best tree for our purposes will be no larger. We therefore choose the “best” size for the tree by our own cross-validation, choosing from among trees of size $1, 2, \ldots, s_{\text{OPT}}$ using the “one SE” rule. Our tree size will be denoted by $s_{k\text{-NN}}$. Our experience has been that $s_{k\text{-NN}}$ is often, but not always, strictly smaller than the s_{OPT}. For large problems the computational burden grows quickly and we might use a smaller maximum than s_{OPT} or base the choice of $s_{k\text{-NN}}$ on a subset of the training set.

The use of composite classification rules goes back at least 20 years (Dasarathy and Sheela, 1979). The approach in that paper was to optimally partition the training set feature space so as to allow the deployment of the best classifier in each region. Different regions might be assigned different classifiers under this scheme. The advantage of this method was that the computationally expensive techniques like k-NN could be foregone in regions where a simple linear discrimination function, say, could be used. This technique might be called “nested” since the classifiers operate within non-overlapping subsets of the original data space. Our method uses a carefully chosen k-NN classifier inside every leaf. This approach includes the usual k-NN classifier as the special case corresponding to $s_{k\text{-NN}} = 1$. It also includes the ordinary tree classifier as the special case in which no leaf has had any variable added, leaving the “naïve” classifier in every node.

Another technique that combines trees and k-NN classification is the “machete” (Friedman, 1984). In that approach a tree-like procedure is used to select a “relevant” predictor, and the training set is divided into a set of items that are "close" to the test item on this predictor and a set that is not. This process is repeated until only k items remain, and then those items are used in a k-NN classification. Our procedure differs in that only one tree is ever constructed. Therefore some of the interpretability of the tree remains, even as the k-NN technique tries to decrease misclassification rates inside the leaves.

1.4. Implementation

Our work was done with the tree routines from the S-Plus statistical package (MathSoft Inc., 1999) and our own C-language code for k-NN linked in. (This code is available from the authors.) It may be worth noting that, other than linking C code, we made no particular effort to produce code that would run quickly. The time required by the algorithm grows very quickly in the size of the training set. For example, getting one estimate of the cross-validated misclassification rate for a relatively small data set like the image segmentation data (see below) took between 5 and 10 min on a Pentium 500 MHz machine. Of course the problem is readily parallelizable and indeed we use a primitive parallelization scheme to grow different trees on different machines simultaneously. We assume that careful coding would produce further performance improvements. In any case the time required to do prediction is very much less than the time needed to construct the classifier; in the image segmentation case (where the number of leaves is small), the entire test set can be categorized in a few seconds.

2. Data

2.1. Source

Data sets for this article come from the UC Irvine Machine Learning Repository (Merz and Murphy, 1996). For simplicity we required that all the predictor variables be continuous, that there be no or hardly any missing data, and that the data set be "mid-size"—that is, that there be on the order of 200–1000 items. Data sets in the repository meeting these criteria include "breast", "diabetes", "glass", "image segmentation", "sonar", "vehicle" and "vowel". We also used the well-known simulated "wave" data set (Breiman et al., 1984).

A benchmark study of classification techniques called the Statlog project (Michie et al., 1994) has studied these same data sets and more (with one exception), so error rates for our classifier can be compared to those for a number of competitors. In the nearest neighbor case, however, Michie et al. (1994) appear to have restricted themselves to 1-NN, without scaling or variable selection. A more recent comparison study (Lim et al., 2000) is also valuable, though they do not make clear how the k in k-NN was chosen. Their k-NN classifier used Mahalanobis distance with a pooled covariance matrix, and did not incorporate variable selection. As the latter paper did, we used the "sonar" data set which does not appear in Michie et al. (1994). We also

Table 1
Data sets

Data set	Trg. size	Test size	No. of vars	No. of classes
Breast	683	(10 cv)	9	2
Diabetes	768	(10 cv)	7	2
Glass	214	(5 cv)	9	6
Image	210	2100	19	7
Sonar	208	(5 cv)	60	2
Vehicle	846	(10 cv)	18	4
Vowel	528	462	11	11
Wave	300	3000	21	3

note somewhat different treatment of missing values in the breast and diabetes data sets in the two sources which precludes direct comparison of results from the two sources on those datasets.

Table 1 shows the data sets used, the sizes of the training and test sets, the number of predictor variables, and the number of classes. Where there is no test set, we used cross-validation to estimate the error rate, choosing the number of cross-validations to agree with Michie et al. (1994).

2.2. *Estimating misclassification rates*

In general our interest lies in the misclassification rate of the classifiers. A test set item is classified by finding the leaf in which it falls, then finding its k-nearest neighbors just among training set items that also fell into that leaf. Of course k will be specific to that leaf, as will the set of variables on which distance is being measured. These variables will be scaled for training and test items, if necessary, by using the SDs for those variables (computed only from the training set).

The act of constructing the classifier requires two levels of cross-validation. In the "outer" loop, we use n-fold cross-validation. In this technique the data is broken into n "chunks" of approximately equal size. Each chunk is held out in turn, and the remaining $n-1$ chunks used to build the classifier. Then in the "inner" loop, we construct trees of every size, using leave-one-out cross-validation to choose the best variable subset, scaling, and k within each leaf. After these are chosen, the number of errors made by each of the trees on the chunk left out is computed. Finally, after all chunks have been set aside in this way, the best size is determined with the "one SE" rule and the final model of that size (using all the data) is constructed.

When a test set is provided, as for the image, vowel, and waveform data sets, the misclassification rate on the test set can then be determined directly. When no test set is provided, we need a third, outermost, layer of cross-validation to compute the overall error rate. (Here we chose the number of chunks to agree with Michie et al. (1994).) In this case, the algorithm for computing the misclassification rate of our classifier is as shown in Algorithm 1:

Algorithm 1. Three layers of cross-validation are used to estimate error rates when no test set is supplied. The innermost chooses the best k-NN classifier within a leaf; the middle selects the best size of tree; the outermost uses the best tree, constructed on 90% (say) of the data, to estimate the error rate from the remaining 10%.

```
1: Break data into $n_1$ subsets
2: Loop, omitting subset $i$, $i = 1, \ldots, n_1$, in turn
    i: Break data into $n_2$ subsets
   ii: Loop, omitting subset $j$, $j = 1, \ldots, n_2$, in turn
      a: Build tree to "full size"
      b: For tree size = 1 to "full size"
         $\rightarrow$ Find best tree of that size
         $\rightarrow$ For each leaf
            Find best $k$, variable subset, and scaling in each leaf by
            leave-one-out cross-validation
         $\rightarrow$ Classify subset $j$, count misclassifications
      c: Choose tree size with fewest misclassifications
  iii: Construct "best-sized" tree and $k$-NN classifiers in leaves
       using all $n_2$ subsets
   iv: Use this to classify subset $i$; accumulate misclassifications
3: Report misclassification rate across all $n_1$ subsets
```

2.3. Comparing misclassification rates

As we have noted, certain data sets come equipped with a pre-specified test set. For these data sets we report the misclassification rate on that test set. In data sets with no pre-specified training set, we have, like Michie et al. (1994), used cross-validation to get an estimate of misclassification rate. Of course, this estimate is random; a different division of the data into chunks would produce a different estimate. Michie et al. (1994) appear to have run only one cross-validation for each classifier. We ran each cross-validation ten times by choosing, at random, ten random number seeds from the set of available seeds. We could therefore ensure the same division of the data for the different classifiers on each replication.

3. Results

Table 2 shows the results achieved by our method ("k-NN-i-l") and by "regular" k-NN and classification trees on these data sets. For "regular" k-NN, the number of nearest neighbors was chosen with leave-one-out cross-validation from the set $1, 3, \ldots, 31$; for classification trees the best tree size was chosen with cross-validation and the "one SE" rule. When error rates were estimated by cross-validation, the numbers of leaves for the tree and the k-NN-in-leaf are the averages of the numbers of leaves in the trees built during cross-validation.

Table 2
Misclassification rates

Data set	k-NN rate (%)	Tree rate (%)	Leaves	k-NN-i-l rate (%)	Leaves
Breast	3.94	5.39	5.55	3.95	1.01
Diabetes	27.6	25.7	4.41	27.5	1.39
Glass	34.8	37.8	7.46	35.0	1.28
Image	39.4	12.5	9	9.47	3
Sonar	22.7	27.2	4.92	22.5	1.04
Vehicle	36.6	28.7	21.4	30.8	4.35
Vowel	45.9	60.4	31	45.9	1
Wave	20.8	28.7	17	20.8	1

We note that in four cases the composite classifier is lowest or tied for lowest in misclassification rate. In two examples the "optimal" k-NN-in-leaf tree has only one leaf, in which case, of course, our classifier is identical to the k-NN one. Our model for the "image" data shows a large increase in quality over both the competitors. For the other data sets, our classifier falls strictly between the other two in misclassification rate.

3.1. Noise resistance

One problem with the usual k-NN classifier is its susceptibility to "noise" variables. This is a consequence of the lack of variable selection in that implementation. Of course when variables with no predictive power are included in the model they contribute only noise to the Euclidean distance; when there are lots of noise variables, accurate classification becomes impossible. To ensure that our technique was holding up under noise, we used the simulated "wave" data with ten added variables that were iid Normals generated from the random number generator in S-Plus. Each of the ten added variables had the same SD; the experiment was performed with those SDs set at 0.1, 1.0, and 10.0. In each case the algorithm included precisely those variables that had been included with it was run on the original data sets. We conclude that the forward selection algorithm is effective, at least in these cases, in excluding "noise" variables from the model.

Our (somewhat limited) experience suggests that our technique will show greatest improvement over the classification tree or the k-nearest-neighbor classifier when the tree does "reasonably well." When the tree does poorly, the division of the measurement space does not improve classification much, so using k-nearest-neighbors inside the leaves cannot be expected to show improvement. Naturally, when the tree does well, there is little improvement to be made: the leaves are relatively "pure," and no technique can be expected to help classification accuracy much.

4. Conclusions

The first thing we notice is that the performance of the usual k-NN is greatly improved when scaling, variable selection and choice of k are included. (In an earlier draft of this paper we used a more naïve k-NN classifier that did not perform nearly as well.) Using separate k-NN classifiers inside the leaves of a classification tree seems often to reduce misclassification error over using the tree alone, using the usual k-NN classifier, or both. Indeed in the data sets we looked at our approach was never worse than both the classification tree and the ordinary k-NN classifier.

We believe that our approach is successful when some leaves in the tree have information left in them that the tree, because it makes linear splits, cannot exploit. k-NN, conversely, relies on a global approach to “distance” that may not be reasonable in some heterogeneous data sets. Our technique uses the local nature of the tree fit but adds a more sophisticated model within leaves than the simple “plurality vote” rule. Cross-validation suggests that the “best” tree for this work will often be smaller than the optimal classification tree. We expect our approach to be weakest where there are noise variables, so a variable selection rule would be useful here.

4.1. Future work

This work raises a number of interesting questions. The notion of dividing the measurement space into pieces and applying a separate classifier inside each piece is not a new one. We have shown that combining k-NN and trees in this way can reduce both the computational load associated with k-NN and the misclassification rate.

One modification is to re-consider the use of the Euclidean distance in the nearest-neighbor metric. For example we might choose a Minkowski-type distance like $d(\mathbf{x}_r, \mathbf{x}_s) = [\sum_{i=1}^{p} c_i |x_{ri} - x_{si}|^q]^{1/q}$, in which not only the set of c_i but also q are chosen by cross-validation. Further, there is no reason to restrict ourselves to using a k-NN classifier in each of the leaves. In pure leaves, there is no reason to use any additional classifier, and certainly there may in general be some leaves where, say, a linear discriminant might be best, others where a neural network might be best and so on. Future work to determine which classifier should be used in each piece is necessary. Of course, this echoes the general question often asked in classification problems: given a data set, which classifier ought to work best? There is no general answer to this of which we are aware.

While hybrid classifiers of this sort have promise, it must be noted that there is a cost in interpretability. Separately both trees and k-NN are fairly easy to understand; in this hybrid form the resulting classifier is more difficult to describe. Might a more general hybrid be even more difficult to understand? In a memorable recent address, Leo Breiman postulated that no classifier can be both simple and accurate. Where accuracy is required, it may be that hybrid classifiers are one way to achieve that.

References

Breiman, L., Friedman, J., Olshen, R., Stone, C., 1984. Classification and Regression Trees. Wadsworth and Brooks, Monterey, CA.

Dasarathy, B., Sheela, B., 1979. A composite classifier design: concepts and methodology. In: Proceedings of the IEEE (Special Issue on Pattern Recognition and Image Processing), Vol. 67, pp. 708–715.

Friedman, J., 1984. Flexible metric nearest-neighbor classification. Technical Report, Stanford University.

Fukunaga, K., Flick, T., 1984. An optimal global nearest neighbor metric. IEEE Trans. Pattern Anal. PAMI-6 (3), 314–318.

Lim, T., Loh, W., Shih, Y., 2000. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classifcation algorithms. Mach. Learning 40, 203–228.

MathSoft Inc., 1999. *S*-Plus 2000 Guide to Statistics, Data analysis Products Division, MathSoft, Seattle, WA.

Merz, C., Murphy, P., 1996. U.C. Irvine Repository of Machine Learning Databases, (www.ics.uci.edu.∼mlearn/∼MLRepository.html) Department of Information and Computer Science, Irvine, CA: University of California.

Michie, D., Spiegelhalter, D., Taylor, C., 1994. Machine Learning, Neural and Statistical Classification. Ellis Horwood, New York.

Short, R., Fukunaga, K., 1980. A New Nearest Neighbor Distance Measure. In: Proceedings of the Fifth IEEE Computer Society Conference on Pattern Recognition and Image Processing. IEEE Computer Society, Silver Spring, MD.

Simard, P., LeCun, Y., Denker, J., 1993. Efficient pattern recognition using a new transformation distance. In: Advances in Neural Information Processing Systems. Morgan Kaufman, Los Altos, CA.