Notes On Setting Up, Using, And Understanding
Random Forests V3.0

The V3.0 version of random forests contains some modifications
and major additions to Version 2.2.   It deletes some portions, like
the density estimation procedure, which I found unreliable. and
fixes a flaw in version 2.

I apologize in advance for all bugs and would like to hear about
them.   To find out how this program works, read my paper "Random
Forests"   published in the Machine Learning. Journal and available
on my web site

The program is written in extended Fortran 77 making use of a
number of VAX extensions.   It runs on SUN workstations f77 and on
Absoft Fortran 77 (available for Windows) and on the free g77
compiler. but may have hang ups on other f77 compilers.   If you
find such problems and fixes for them,   please let me know.

Random forests does
    classification
    computes principal   coordinates to use as variables.
    variable importance (in a number of ways)
    computes proximity measures between cases
    computes scaling displays for the data
    gives a measure of outlyingness for each case

The last three can be done for the unsupervised case i.e. no class
labels.   I have used proximities   to cluster data and they seem to do
a reasonable job.   The new addition uses the proximities to do
metric scaling of the data.   The resulting pictures of the data are
often useful.

The first part of these notes contains instructions on how to set up
a run of random forests.   The second part contains the notes on the
features of random forests and how they work.

I. *Setting   Parameters*

The first seven lines following the parameter statement need to be
filled in by the user.

Line 1 Describing The Data

mdim0=number of variables
nsample0=number of cases (examples or instances) in the data
nclass=number of classes
maxcat=the largest number of values assumed by a categorical
        variable in the data
ntest=the number of cases in the test set.   NOTE: Put ntest=1 if
        there is no test set.   Putting ntest=0 may cause compiler
        complaints.
l**abelts**=0 if the test set has no class labels, 1 if the test set has class
        labels.

If their are no categorical variables in the data set maxcat=1.   If
there are categorical variables, the number of categories assumed
by each categorical variable has to be specified in an integer vector
called cat, i.e. setting cat(5)=7 implies that the 5th variable is a
categorical with 7 values.   If maxcat=1, the values of cat are
automatically set equal to one.   If not, the user must fill in the
values of cat in the early lines of code.

For a J-class problem, random forests expects the classes to be
numbered 1,2, ...,J.   For an L valued categorical, it expects the
values to be numbered 1,2, ... ,L. At prezent, L must be less than or
equal to 32.

A test set can have two purposes--first: to check the accuracy of RF
on a test set.   The error rate given by the internal estimate will be
very close to the test set error unless the test set is drawn from a
different distribution.   Second: to get predicted classes for a set of
data with unknown class labels.   In both cases the test set must
have the same format as the training set.   If there is no class label
for the test set, assign each case in the test set label classs #1, i.e.
put cl(n)=1, and set labelts=0.   Else set labelts=1.

Line 2   Setting Up The Run

mtry=number of variables randomly selected at each node
jbt=number of trees to grow
look=how often you want to check the prediction error
ipi=set priors
indsize=minimum node size

mtry:
this is the only parameter that requires some judgment to set, but
forests isn't too sensitive to its value as long as it's in the right ball
park. I have found that setting mtry equal to the square root of
mdim gives generally near optimum results. My advice is to begin
with this value and try a value twice as high and half as low
monitoring the results by setting look=1 and checking the internal
test set error for a small number of trees. With many noise
variables present, mtry has to be set higher.

jbt:
this is the number of trees to be grown in the run.   Don't be
stingy--random forests produces trees very rapidly, and it does not
hurt to put in a large number of trees. If you want auxiliary
information like variable importance or proximities grow
a lot of trees--say a 1000 or more.   Sometimes, I run out to 5000
trees if there are many variables and I want the variables
importances to be stable.

look:
random forests carries along an internal estimate of the test set
error as the trees are being grown.   This estimate is outputted to
the screen every look trees. Setting look=10, for example, gives the
internal error output every tenth tree added. If there is a labeled
test set, it also gives the test set error.   Setting look=jbt+1
eliminates the output. Do not be dismayed to see the error rates
fluttering around slightly as more trees are added.   Their behavior
is analagous to the sequence of averages of the number of heads in
tossing a coin.

ipi: pi is an real-valued vector of length nclass which sets prior
probabilities for classes. ipi=0 sets these priors equal to the class
proportions. If the class proportions are very unbalanced, you may
want to put larger priors on the smaller classes. If different
weightings are desired, set ipi=0 and specify the values of the {pi(j)}
early in the code. These values are later normalized, so setting

pi(1)=1, pi(2)=2 implies that the probability of seeing a class 2 instance is twice as large as that of seeing a class 1 instance.

ndsize: setting this to the value k means that node node fewer than k cases will be split. The default that always gives good performances is ndsize=1. Pn larege data sets, memeroy will be preserved and speed enchanced of ndsize is set larger. Usually, this iresults in only a negligble loss of accuracy.

Line 3 Variables to Include

This option is included as a matter of convenience. I coded it when searching to find which variables were "important". To use this option the data must be read in as x0(mdim0,nsample) instead of x(mdim, nsample). The values of the msel variable have to be set.

ivarin: only those variaables for which msel =1 will be used in prediction

inarout: only those variaables for which msel <>1 will be used in prediction.

Line 4 Options

i**mp**=1 turns on the variable importances method described below.

iprox=1 turns on the computation of the intrinsic proximity measures between any two cases .

iaddcl=1 If the data is without labels (i.e. unsupervised data) then iaddcl =1 labels this data class #1 and generates a synthetic data set of the same size which is labeled class #2. The synthetic data is sampled independently from the marginals of the original data.

noutlier=1 computes an outlyingness measure for all cases in the data. If this is on, then iprox must also be switched to one, If iaddcl=1 then the outlyingness measure is computed only for the original data.

Line 5 Scaling

iscale=1 turns on the scaling and extracts the scaling coordinates from the proximities. iprox=1 is necessary. If iaddcl=1, then the scaling is outputted only for the original data.

msdim is the number of scaling coordinates to output. Generally, 4-5 are more than sufficient.

Line 6 Transform to Principal Coordinates

ipc=1 takes the x-values and computes principal coordinates from the covariance matrix of the x's. These will be the new variables for RF to operate on. This will not work right if some of the variables are categorical.

mdimpc: This is the number of principal components to extract. It has to be <=mdim.

norm=1 normaizes all of the variables to mean zero and sd one before computing the principal componenets.

Line 7 Output Controls

Note: user must supply file names for all output listed below or send it to the screen.

nsumout=1 writes out summary data to the screen. This includes errors rates and the confusion matrix

infout=1 prints the following columns to a file
i) case number
ii) 1 if predicted class differs from true class, 0 else
iii) true class label
iv) predicted class label
v) margin=true class prob. minus the max of the other class prob.
vi)-vi+nclass) class probabilities

ntestout=1 prints the follwing coumns to a file
i) case number in test set
ii) true class (true class=1 if data is unlabeled)
iii) predicted class
iv-iv+nclass) class probabilities

imp=1   prints the following columns to a file
i)   variable number

variables importances computed as:
ii) The % rise in error over the baseline error.
iii) 100* the change in the margins averaged over all cases
iv) The proportion of cases for which the margin is decreased minus
the proportion of increases.
v) The gini increase by variable for the run

impsetout=1 prints out for each case the following columns:
i) case number
ii) margin for the case
iii - iii+mdim) altered margin due to noising up mth variable.

iproxout=1 prints to file
i) case #1 number
ii) case #2 number
iii) proximity between case #1 and case #2

iscale=1 prints out the following columns:
i) case number
ii) true class
iii) predicted class.
iv)   0 if ii)=iii), 1 otherwise
v-v+msdim) scaling coordinates

noutlier=1 prints the following columns to a file
i) sequence number
ii) class
iii) case nunber
iv) outlyingness measure

LUSER WORK:

The user has to construct the read-in the data code of which I have
left an example.   This needs to be done after the dimensioning of
arrays.   If maxcat >0 then the categorical values need to be filled in.
If ipi=0, the user needs to specify the relative probabilities of the
classes.

REMARKS:

The proximities can be used in the clustering program of your choice.   Their advantage is that they are intrinsic rather than an ad hoc measure.   I have used them in some standard and home-brew clustering programs and gotten reasonable results.   The proximities between class 1 cases in the unsupervised situation can be used to cluster.   Extracting the scaling coordinates from the proximities and plotting scaling coordinate i versus scaling coordinate j
gives illuminating pictures of the data.   Usually, i=1 and j=2 give the most information   (see the notes below).

There are four measures of variable importance: They complement each other. Except for the 4th they are based on the test sets left out on each tree construction.   On a microarray data with 5000
variables and less than 100 cases, the different measures single out much the same variables (see notes below). But I have found one synthetic data set where the 3rd measure was more sensitive than the first three.

Sometimes, finding the effective variables requires some hunting.   If the effective vzriables are clear-cut, then the first measure will find them.   But if the number of variables is large compared to the number of cases, and if the predictive power of the individual variables is small, the other measures can be useful.

Random forests <u>does not overfit.</u> You can run as many trees as you want.   Also, It is fast.   Running on a 250mhz machine, the current version using a training set with 800 cases, 8 variables, and mtry=1, constructs each tree in .1 seconds.   On a training set with 2200
cases, 11 variables, and mtry=3, each tree is constructed in .2
seconds.   It takes 6 seconds per tree on a training set with 15000 cases and 16 variables with mtry=4, while also making computations for a 5000 member test set.

The present version of random forests does not handle missing values.   A future version will.   It is up to the user to decided how to deal with these.   My current preferred method is to replace each missing value by the median of its column.   My impression is that because of the randomness and the many trees grown, filling in missing values with a sensible values does not effect accuracy much.

For large data sets, if proximities are not required, the major memory requirement is the storage of the data itself, and the three

integer arrays a,at,b.   If there are less than 64,000 cases, these latter three may be declared integer*2 (non-negative).   Then the total storage requirement is about three times the size of the data set.   If proximities are calculated, storage requirements go up by the square of the number of cases times eight bytes (double precision).

Outline Of How Random Forests Works

Usual Tree Construction--Cart

Node=subset of data.   The root node contains all data.

At each node, search through all variables to find
        best split into two children nodes.

Split all the way down and then prune tree up to
        get minimal test set error.

Random Forests Construction

Root node contains a bootstrap sample of data of same size as original data.   A different bootstrap sample for each tree to be grown.

An integer K is fixed, K<<number of variables.   K is the **only** parameter that needs to be specified.   Default is the square root of number of variables.

At each node, K of the variables are selected at random.   Only these variables are searched through for the best split. The largest tree possible is grown and is not pruned.

The forest consists of N trees. To classify a new object having coordinates **x**, put **x** down each of the N trees. Each tree gives a classification for **x.**

The forest chooses that classification having the most out of N votes.

Transformation to Principal Coordinates

One of the users lent us a data set in which the use of a few principal components as   variables reduced the error rate by

2/3rds.   On experimenting, a few other data sets were found where the error rate was significantly reduced by pre-transforming to principal coordinates   As a convenience to users, a pre-transformation   subroutine was incorporated into this version.

Random   Forests   Tools

The design of random forests is to give the user a good deal of information about the data besides an accurate prediction. Much of this information comes from using the "out-of-bag" cases in the training set that have been left out of the bootstrapped training   set.

The information includes:

a) <u>Test set error rate.</u>

b) <u>Variable   importance   measures</u>

c) <u>Intrinsic   proximities   between   cases</u>

d) <u>Scaling   coordinates   based   on   the   proximities</u>

e) <u>Outlier   detection</u>

I will explain how these function and give applications, both for labeled   and   unlabeled   data.

Test   Set   Error   Rate

In random forests, there is no need for cross-validation or a separate test set to get an unbiased estimate of the test set error.   It is gotten internally, during the run, as follows:

  Each tree is constructed using a different bootstrap sample from the original data.   About one-third of the cases are   left out of the bootstrap sample and not used in the construction of the kth tree.

<u>Test Set Error Rate</u>

Put each case left out in the construction of the kth tree down the kth tree to get a classification.

In this way, a test set classification is gotten for each case in about one-third of the trees. Let the final test set classification of the forest be the class having the most votes.

Comparing this classification with the class label present in the data gives an estimate of the test set error.

Variable   Importance.

Because of the need to know which variables are important in the classification, random forests has four different ways of looking at variable importance. Sometimes influential variables are hard to spot--using these four measures provides more information.

Measure 1

To estimated the importance of the mth variable. In the left out cases for the kth tree, randomly permute all values of the mth variable Put these new covariate values down the tree and get classifications.

Proceed as though computing a new internal error rate. The amount by which this new error exceeds the original test set error is defined as the importance of the mth variable.

Measures 2 and 3

For the nth case in the data, its margin at the end of a run is the proportion of votes for its true class minus the maximum of the proportion of votes for each of the other classes. The 2nd measure of importance of the mth variable is the average lowering of the margin across all cases when the mth variable is randomly permuted as in method 1.

The third measure is the count of how many margins are lowered minus the number of margins raised.

Measure 4

The splitting criterion used in RF is the gini criterion--also used in CART. At every split on of the mtry variables is used to form the split and there is a resulting decrease in the gini. The sum of all decreaes in the forest due to a given variable, normalized by the number of trees, froms measure 4.

Additional Case-Wise Information.

For the mth variable, the values of all of the margins in the training set with the mth variable noised up is computed. When the graph of these values is compared to the graph of the original margins, interesting information about individual cases often emerges.

To illustrate the use of this information by some examples. Some of these were done on versioon 1 so may differ somewhat from the version 3 output.

An   Example--Hepatitis   Data

Data: survival or non survival of 155 hepatitis patients with 19 covariates. Analyzed by Diaconis and Efron in 1983 Scientific American. The original Stanford Medical School analysis concluded that the important variables were numbers 6, 12, 14, 19.

Efron and Diaconis drew 500 bootstrap samples from the original data set and used a similar procedure, including logistic regression, to isolate the important variables in each bootstrapped data set.

 Their conclusion , "Of the four variables originally selected not one was selected in more than 60 percent of the samples. Hence the variables identified in the original analysis cannot be taken too seriously."

Logistic   Regression   Analysis

 Error rate for logistic regression is 17.4%.

Variables importance is based on absolute values of the coefficients of the variables divided by their standard deviations.

FIGURE 1 STANDARDIZED COEFFICIENTS-LOGISTIC REGRESSION
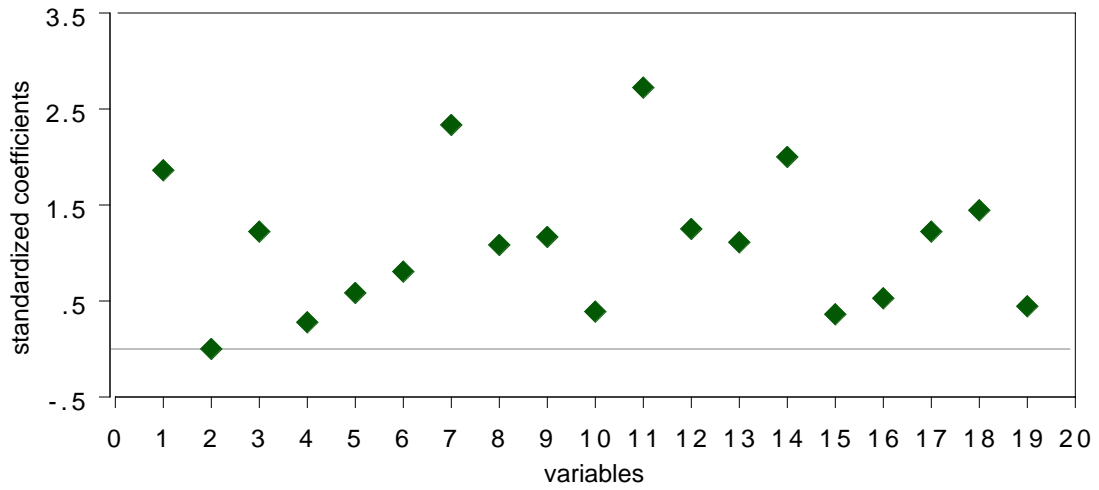
The conclusion is that variables 7 and 11 are the most important covariates.   When logistic regression is run using only these two variables, the cross-validated error rate rises to   22.9% .

Analysis Using Random Forests

The error rate is 12.3%--30% reduction from the logistic regression error.   Variable importances (measure 1) are graphed below:

FIRURE 2 VARIABLE IMPORTANCE-RANDOM FOREST
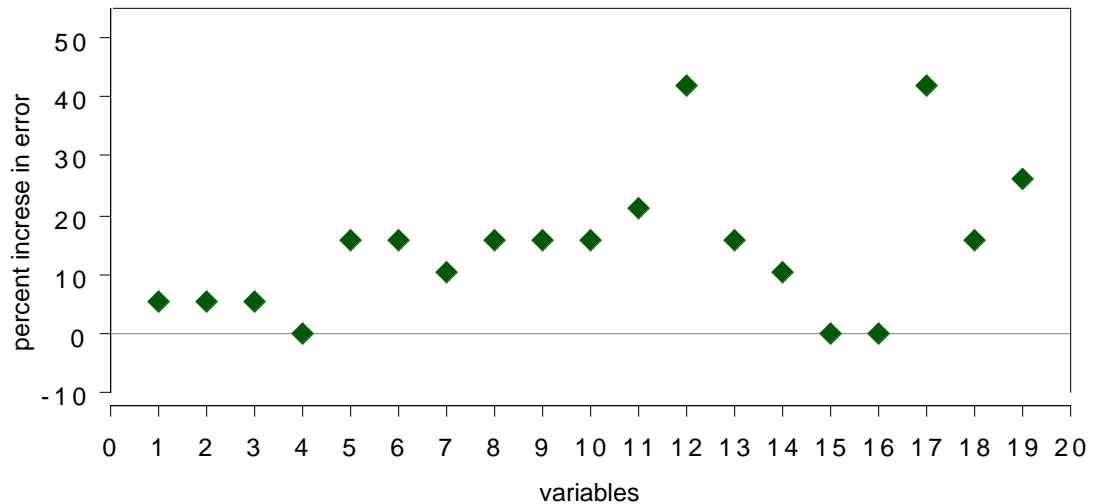
Two variables are singled out--the 12th and the 17th   The test set
error rates running 12 and 17 alone were   14.3% each.   Running
both together did no better.   Virtually all of the predictive capability
is provided by a single variable, either 12 or 17.   (they are highly
correlated)

The standard procedure when fitting data models such as logistic
regression   is to delete variables;   Diaconis and Efron (1983) state ,
"...statistical experience suggests that it is unwise to fit a model that
depends on 19 variables with only 155 data points available."

Newer methods in Machine Learning thrive on variables--the more
the better.   There is no need for variable selection ,On a sonar data
set with 208 cases and 60 variables, Random Forests error rate is
14%.   Logistic Regression has a 50% error rate.

Microarray Analysis

Random forests was run on a microarray lymphoma data set with
three classes, sample size of 81 and 4682 variables (genes) without
any variable selection.   The error rate was   low (1.2%) using
mtry=150.

What was also interesting from a scientific viewpoint   was an
estimate of the importance of each of the 4682 genes.

The graphs below were produced by a run of random forests.

VARIABLE IMPORTANCE-MEASURE 1

Variable Importance Measure 2

Variable Importance Measure 3
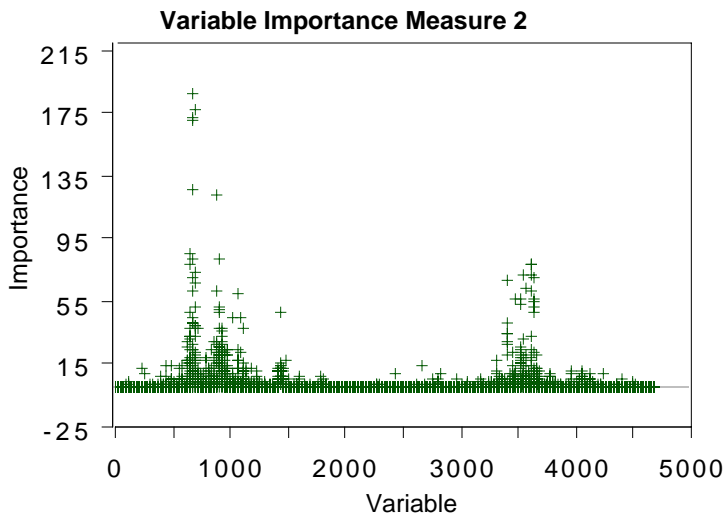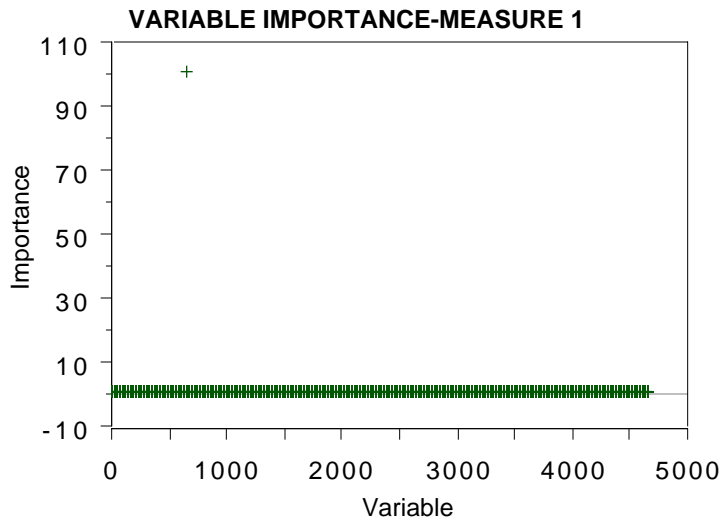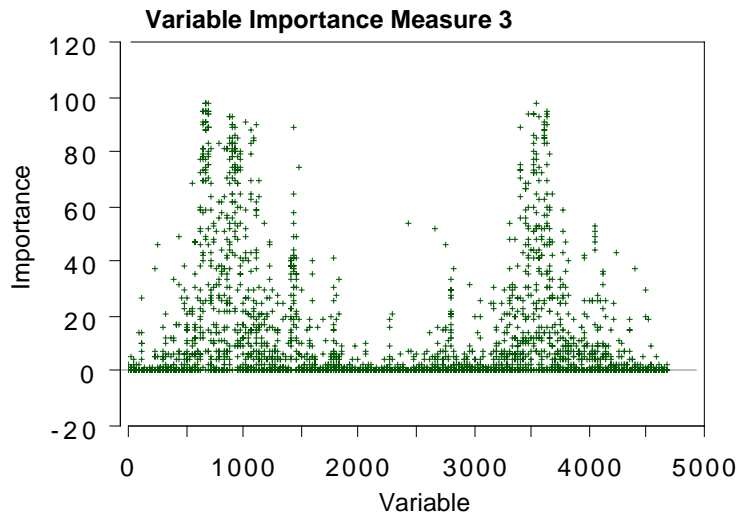
The graphs show that measure 1 has the last sensitivity, showing only one significant variable. Measure 2 has more, showing not only the activity around the gene singled out by measure 1 but also a secondary burst of activity higher up. Measure 3 has too much sensitivity, fingering too many variables.

Class probability estimates

At run's end, for each case there is an out-of-bag estimate of the probability that it is in each one of the J classes. This estimate is given by the proportion of votes for each class. For each member of a test set (with or without class labels), these probabilities are also estimated.

An Astronomical Example:

Bob Becker allowed the use of his quasar data set of 2000 astronomical objects of which about half have been verified as quasars.

Verification is expensive, but there are some variables that are cheap to measure.

Using these cheap variables the data set was run through random forests and for each case a probability PQ(n) outputted that was a probability that the nth case was a quasar.

There is also an unverified test set which we ran through
that assigned a probability PQ(n) to the nth case in the test set.

Telescope time is valuable--the question is: Given an estimate of PQ
for a stellar object, should verification be undertaken.
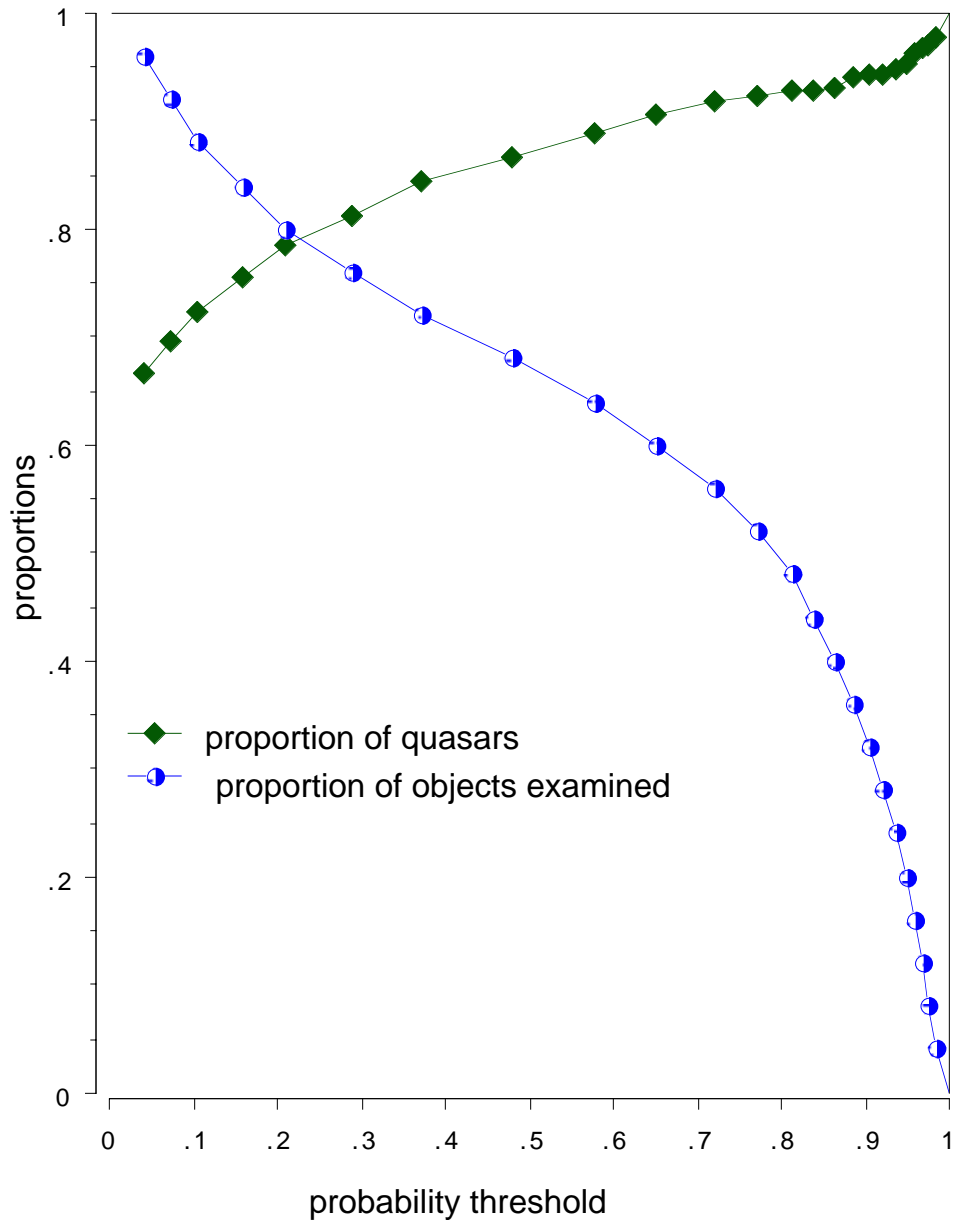
An answer is provided by the training set. For instance,
if all objects with PQ> .9 are verified, then about 95% of
them will be quasars.

An intrinsic proximity measure

Since an individual tree is unpruned, the terminal nodes will contain
only a small number of instances.   Run all cases in the training set
down the tree. If case i   and case j both land in the same terminal
node. increase the proximity between i and j by one.   At the end of
the run, the proximities are divided by twice the number of trees   in
the run and proximity between a case and itself set equal to one.

To cluster-use the above proximity measures.

Example-Bupa   Liver   Disorders

This is a two-class biomedical data set consisting of the covariates

1. mcv   mean corpuscular volume
2. alkphos   alkaline   phosphotase
3. sgpt   alamine   aminotransferase
4. sgot   aspartate   aminotransferase
5. gammagt   gamma-glutamyl   transpeptidase
6. drinks   number of half-pint equivalents of
                                      alcoholic beverage drunk per day

The first   five attributes are the results of blood tests thought to be
related to liver functioning.   The 345 patients are classified into two
classes by the severity of their liver disorders.

The misclassification error rate is 28% in a Random Forests run.
What can we learn about this data?

\

A) *Variable   Importance   (method   1)*

FIGURE 2 VARIABLE IMPORTANCE-BUPA LIVER
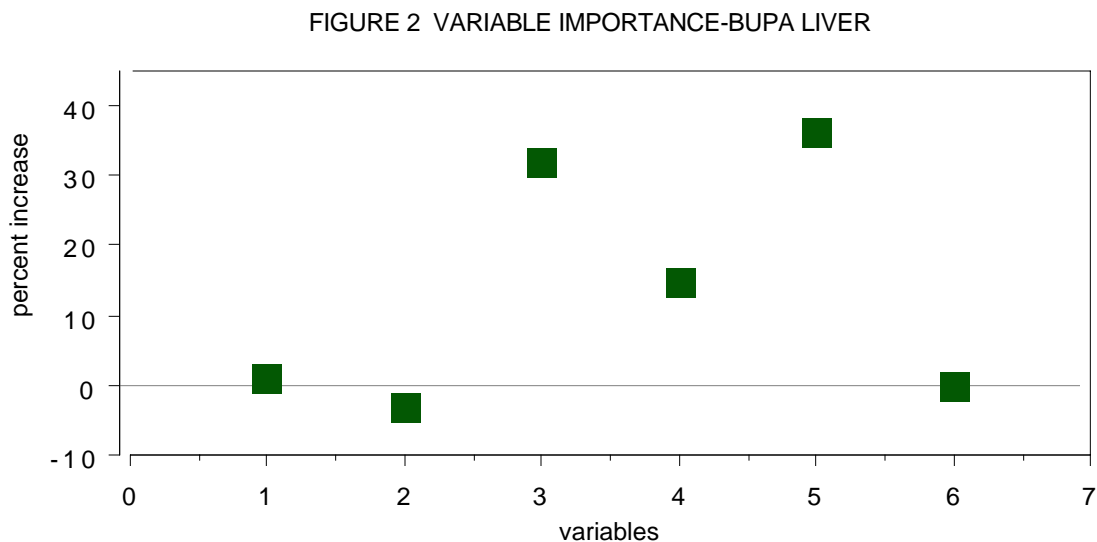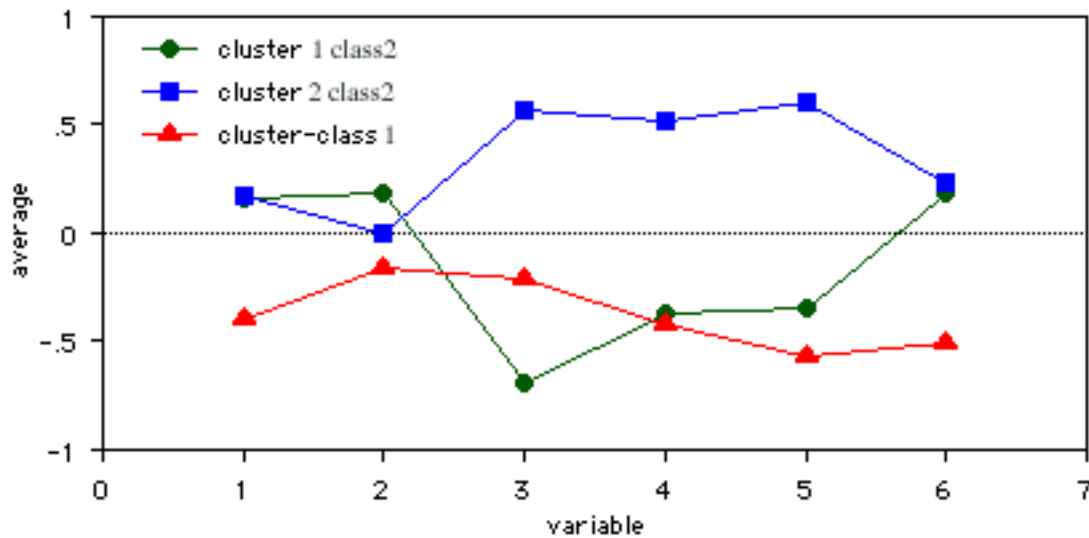

Blood tests 3 and 5 are the most important, followed by test 4.

B) *Clustering*

Using the proximity measure outputted by Random Forests to cluster,   there are two class #2 clusters.

In each of these clusters, the average of each variable is computed and   plotted:

FIGURE 3 CLUSTER VARIABLE AVERAGES

S omething interesting emerges.   The class two subjects consist of two distinct groups: Those that have high scores on blood tests 3, 4, and 5   Those that have low scores on those tests.   We will revist this example below.

Scaling   Coordinates

The proximities between cases n and k form a matrix {prox(n,k)}. From their definition, it is easy to show that this matrix is symmetric, positive definite and bounded above by 1, with the diagonal elements equal to 1.   It follows that the values 1-prox(n,k) are squared distances in a Euclidean space of dimension not greater than the number of cases.   For more background on scaling see "Multidimensional Scaling" by T.F. Cox and M.A. Cox

Let prox(n,-) be the average of prox(n,k) over the 2nd coordinate. and prox(-,-) the average over both coordinates. Then the matrix:

cv((n,k)=.5*(prox(n,k)-prox(n,-)-prox(k,-)+prox(-,-))

is the matrix of inner products of the distances and is also positive definte symmetric.   Let the eigenvalues of cv be $\lambda(l)$ and   the eigenvectors $v_l(n)$ Then the vectors

$$x(n) = (\sqrt{\lambda(1)}v_1(n), \sqrt{\lambda(2)}v_2(n), ...)$$

have squared distances between them equal to 1-prox(n,k).   We
refer to the values of $\sqrt{\lambda(j)}v_j(n)$_ as the jth scaling coordinate.

In metric scaling, the idea is to approximate the vectors $\mathbf{x}(n)$ by the
first few scaling coordinates.   This is done in random forests by
extracting the number msdim of the largest eigenvalues and
corresponding eigenvectors of the cv matrix.   The two dimensional
plots of the ith scaling coordinate vs. the jth often gives useful
information about the data.   The most useful is usually the graph of
the 2nd vs. the 1st.

We illustrate with three examples.   The first is the graph of 2nd vs.
1st scaling coordinates for the liver data

**Metric Scaling**
Liver Data
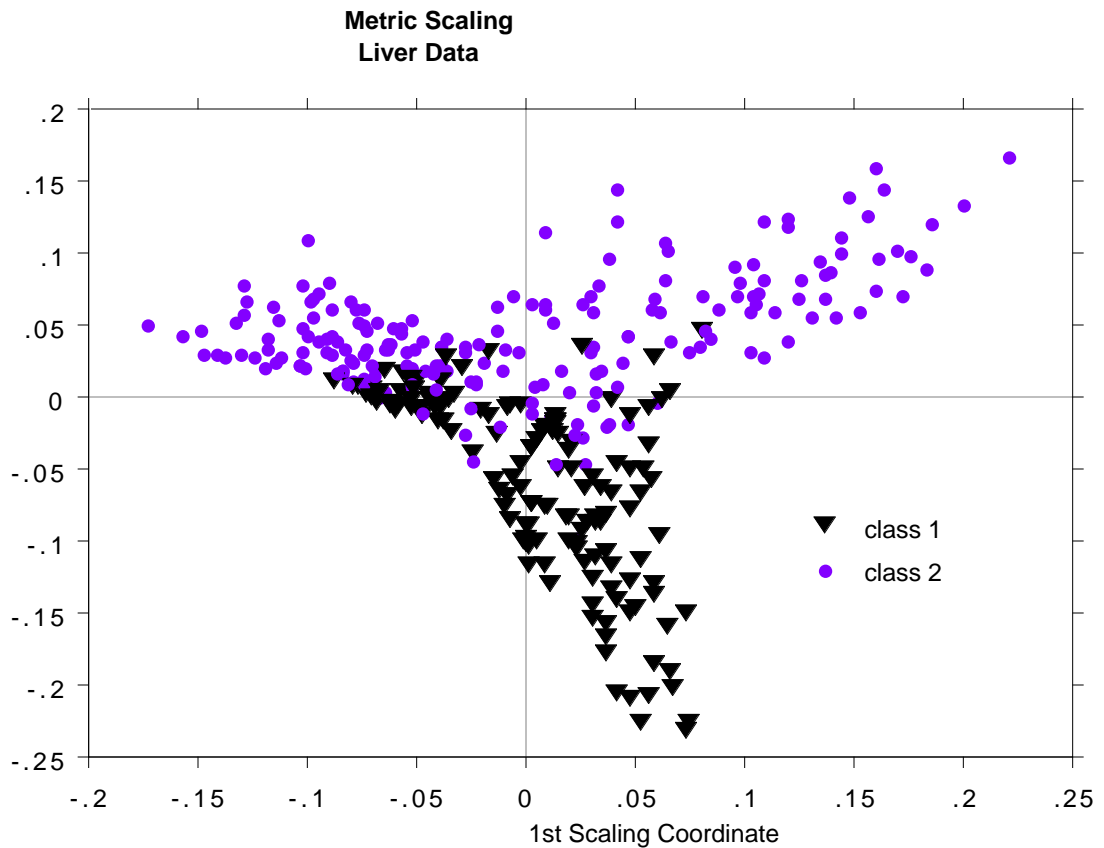
The two arms of the class #2 data in this picture correspond to the
two clusters found and discussed above.

The next example uses the microarray data. With 4682 variables, it is difficult to see how to cluster this data. Using proximities and the first two scaling coordinates gives this picture:

Metric Scaling
Microarray Data
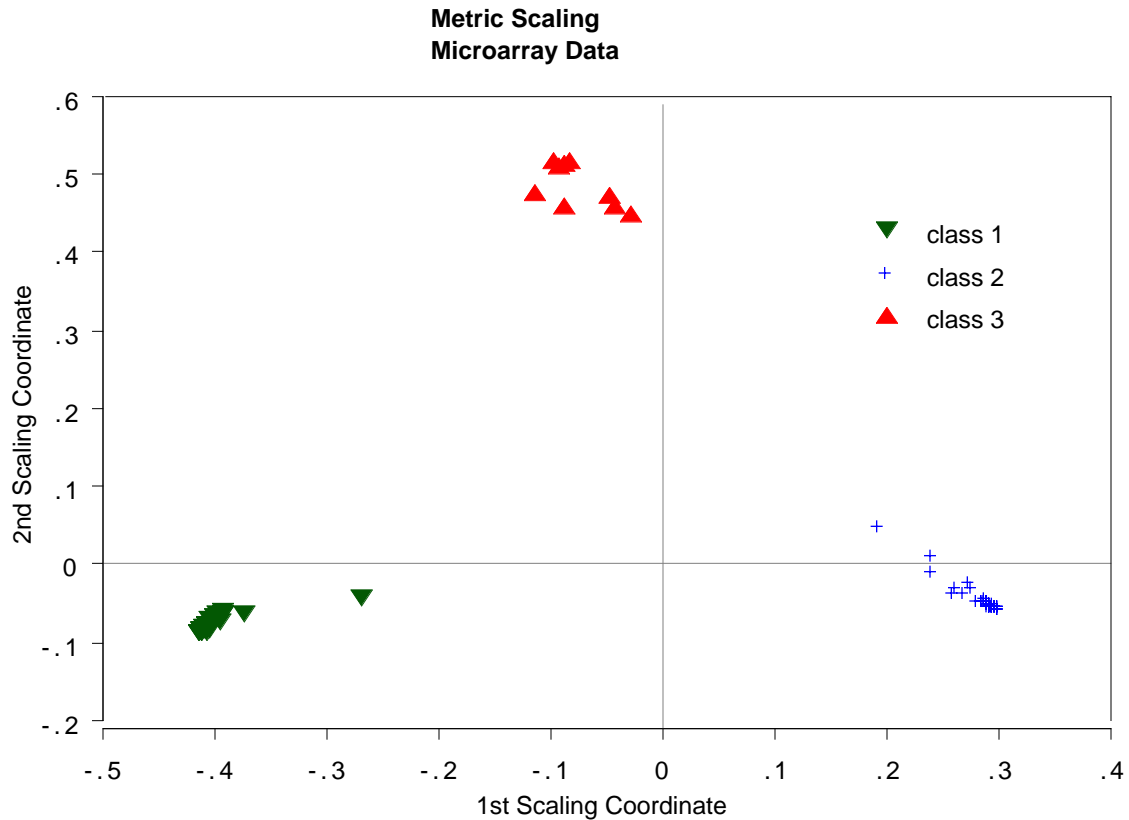
Random forests misclassifies one case. This case is represented by the isolated point in the lower left hand corner of the plot.

The third example is glass data with 214 cases, 9 variables and 6 classes. This data set has been extensively analyzed (see Pattern recognition and Neural Networkks-by B.D Ripley). Here is a plot of the 2nd vs. the 1st scaling coordinates.:

Metric Scaling
Glass data
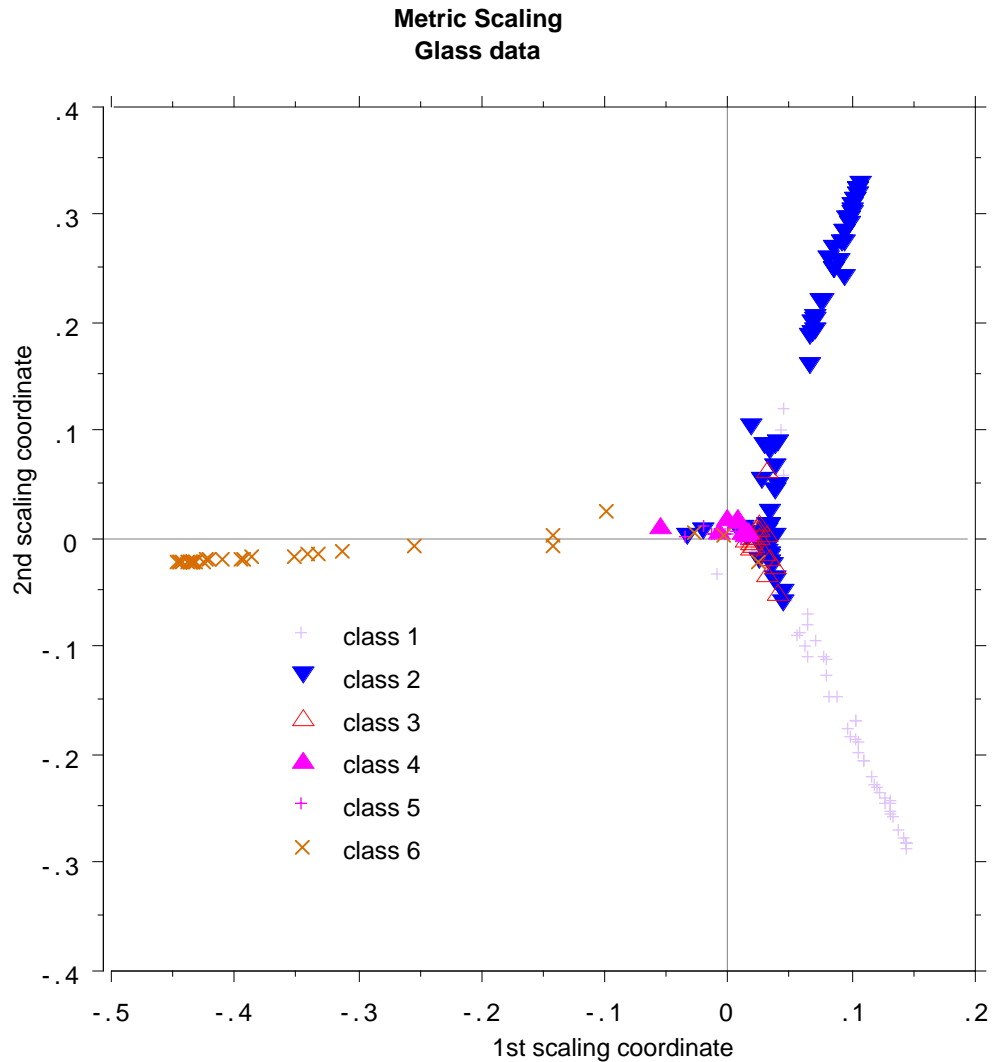
None of the analyses to data have picked up this interesting and revealing structure of the data--compare the plots in Ripley's book.

Outlier   Location

Outliers are defined as cases having small proximities to all other cases.   Since the data in some classes is more spread out than others, outlyingness is defined only with respect to other data in the same class as the given case.   To define a measure of outlyingness, we first compute, for a case n, the sum of the squares of prox(n,k) for all k in the same class as case n.   Take the inverse of this sum--it will be large if the proximities prox(n,k) from n to the other cases k in the same class are generally small. Denote this quantity by out(n).

For all n in the same class, compute the median of the out(n), and then the mean absolute deviation from the median.   Subtract the median from each out(n) and divide by the deviation to give a normalized measure of outlyingness.   Yhe values less than zero are set to zero.   Generally, a value above 10 is reason to suspect the case of being outlying.   Here is a graph of outlyingness for the microarray   data

Outlyingness Measure
Microarray Data
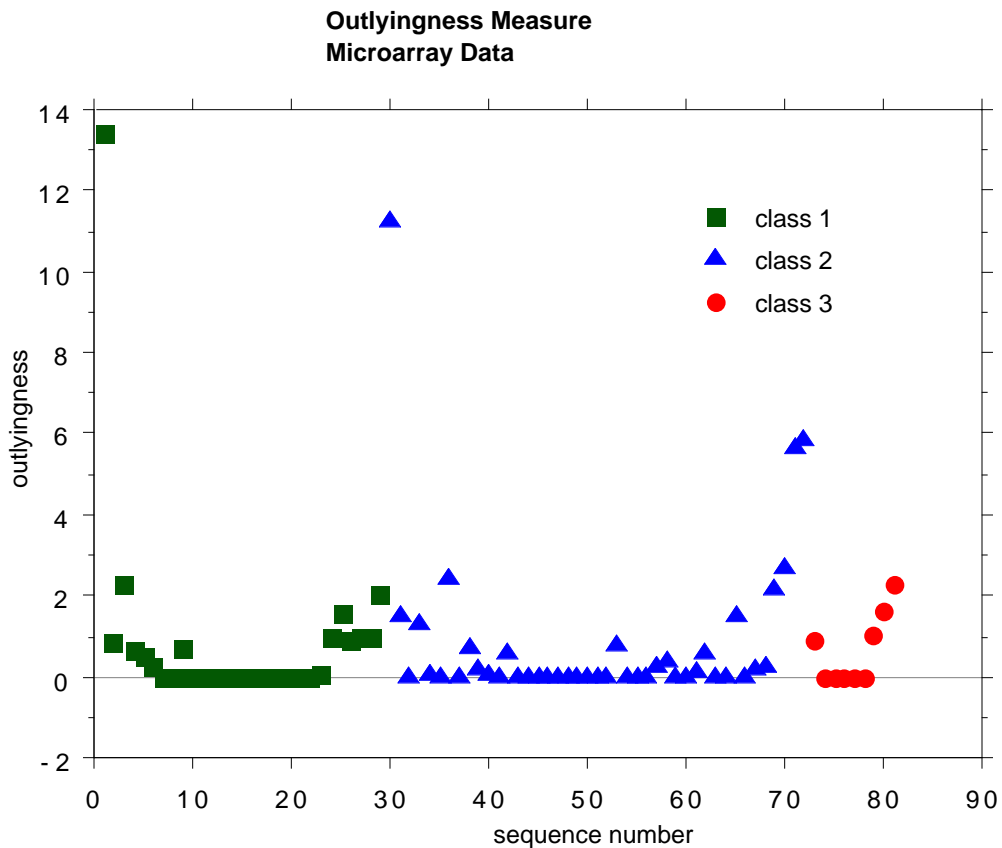
There are two possible outliers--one is the first case in class 1,   the second is the first case in class 2.

As a second example, we plot the outlyingness for the Pima Indians hepatitis data.   This data set has 768 cases, 8 variables and 2 classes. It has been used often as an example in Machine Learning papers but has been suspected of containing a number of outliers.

Outlyingness Measure
Pima Data
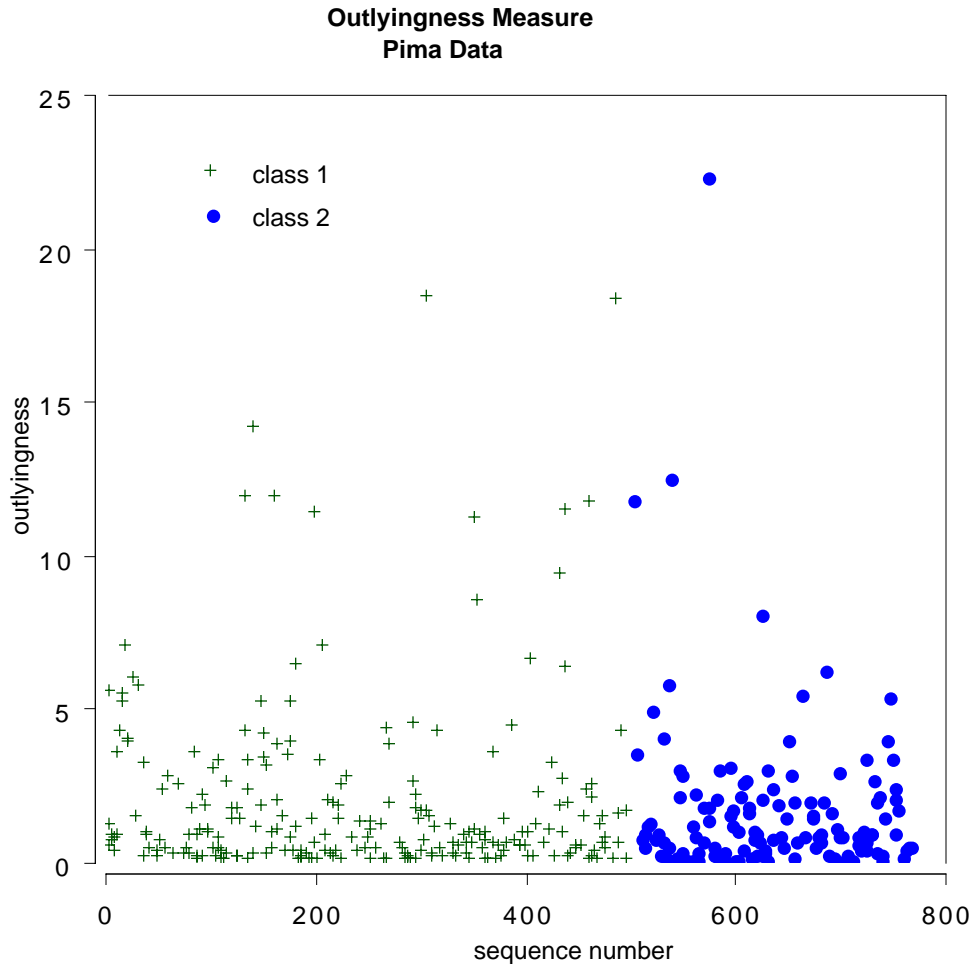
If 10 is used as a cutoff point, there are 12 cases suspected of being outliers.

Analyzing Unlabeled Data

Unlabeled date consists of N vectors $\{\mathbf{x}(n)\}$ in M dimensions. Using the iaddcl option in random forests, these vectors are assigned class label 1. Another set of N vectors is created and assigned class label 2. The second synthetic set is created by independent sampling from the one-dimensional margin distributions of the original data.

For example, if the value of the mth coordinate of the original data for the nth case is x(m,n), then a case in the synthetic data is constructed as follows: its first coordinate is sampled at random from the N values x(1,n), its second coordinate is sampled at

random from the N values x(2,n), and so on.   Thus the synthetic
data set can be considered to have the distribution of M
independent variables where the distribution of the mth variable is
the same as the univariate distribution of the mth variable in the
original data.

When this two class data is run through random forests a high
misclassification rate--say over 40%, implies that there is not much
dependence structure in the original data.   That is, that its structure
is largely that of M independent variables--not a very interesting
distribution.   But if there is a strong dependence structure between
the variables in the original data, the error rate will be low.   In this
situation, the output of random forests can be used to learn
something about the structure of the data.   The following is an
example.

An Application to Chemical Spectra

Data graciously supplied by Merck consists of the first 468 spectral
intensities in the spectrums of 764 compounds.   The challenge
presented by Merck was to find small cohesive groups of outlying
cases in this data.   Using the iaddcl option, there was excellent
separation between the two classes, with an error rate of 0.5%,
indicating strong dependencies in the original data.

We looked at outliers and generated this plot.

Outlyingness Measure
Spectru Data
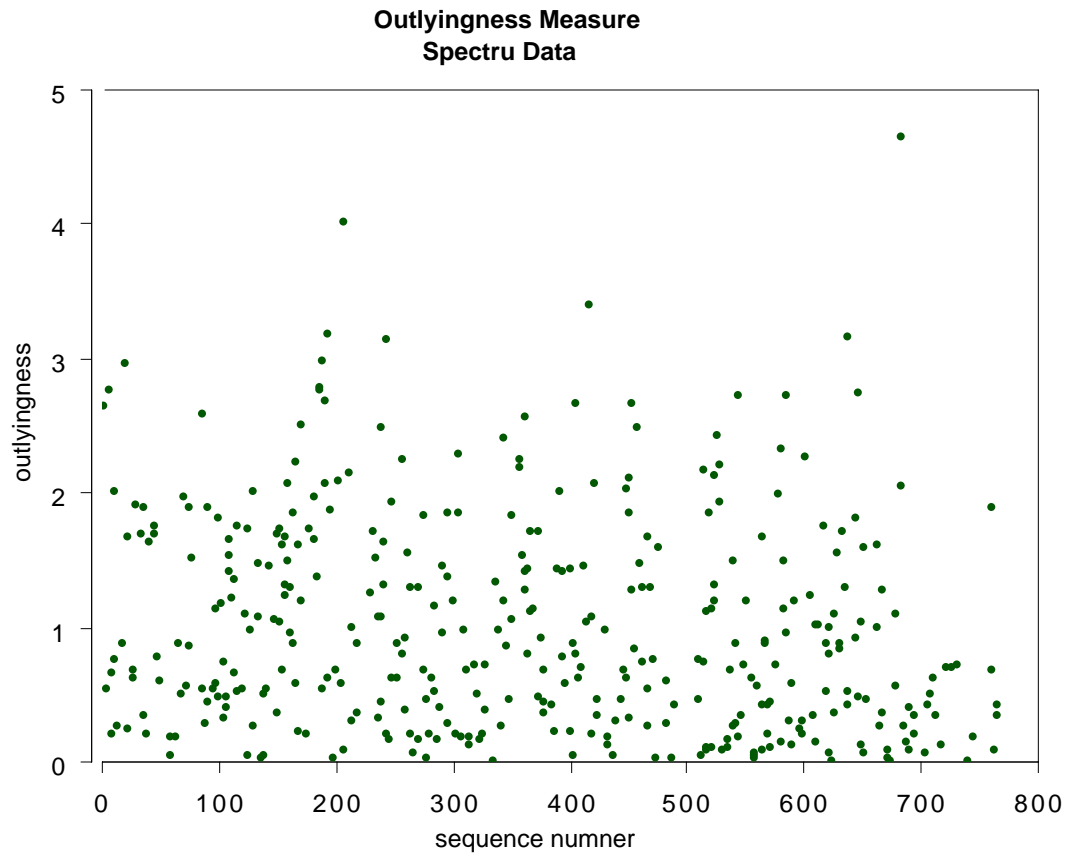
This plot gives no indication of outliers. But outliers must be fairly isolated to show up in the outlier display. To search for outlying groups scaling coordiantes were computed. The plot of the 2nd vs. the 1st is below:

Metric Scaling
Specta Data
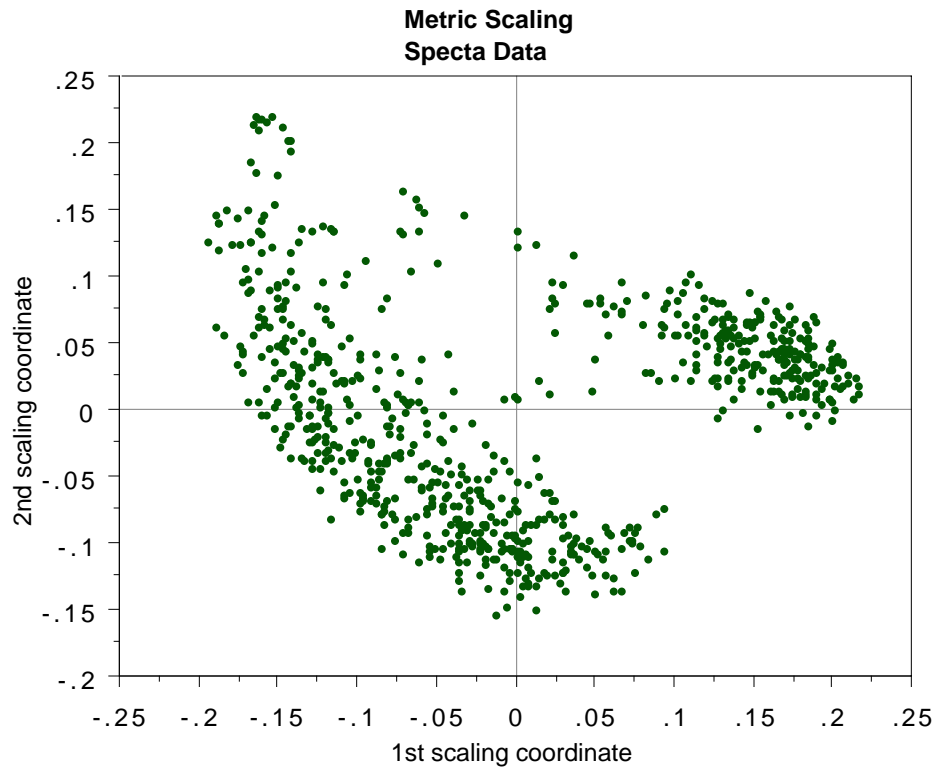
This shows, first, that the spectra fall into two main clusters. There is a possiblity of a small outlying group in the upper left hand corner. To get another picture, the 3rd scaling coordiante is plotted vs. the 1st.

Metrc Scaling
Specta Data
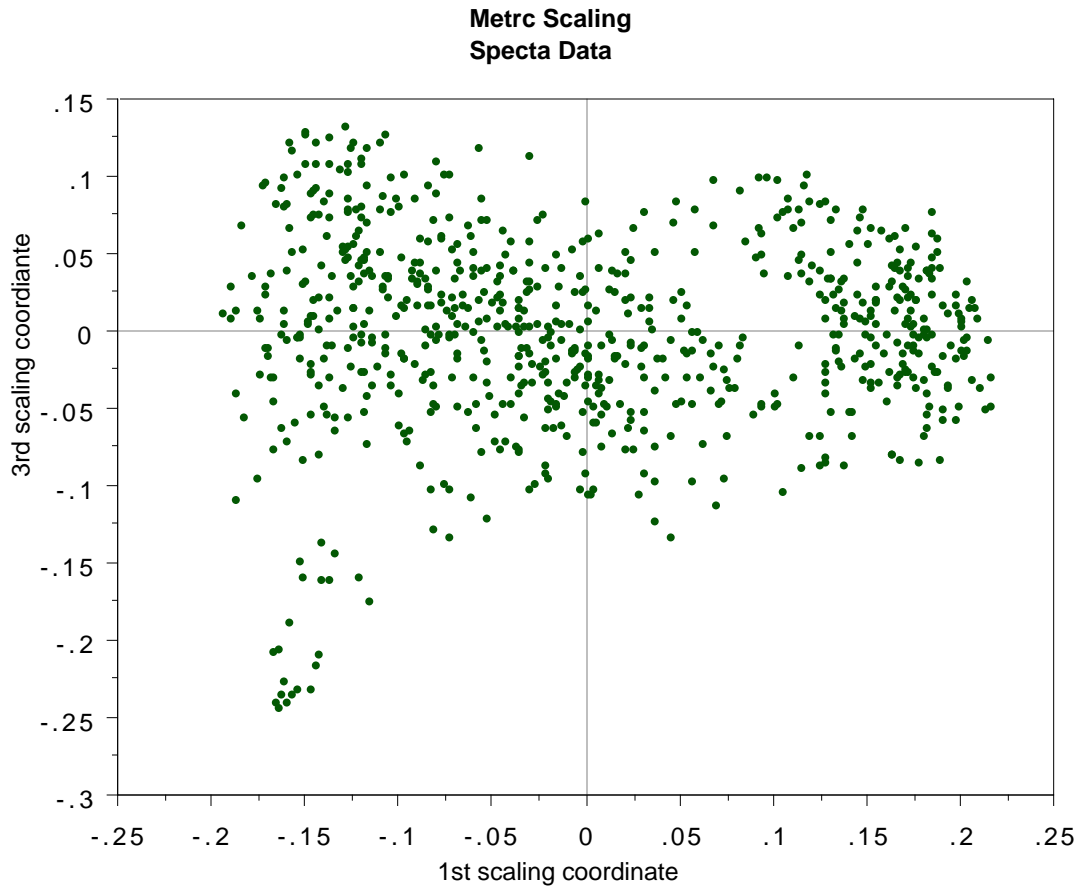
The group in question is now in the lower left hand corner and its separation from the body of the spectra has become more apparent.