# ***Notes On Setting Up, Using, And Understanding Random Forests***

Version 2 of random forests has been restructured to be faster on large data sets then v.1 plus it contasins more options. 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-Random Features" Its available as a technical report if you go back to my department home page (www.stat.berkeley.edu) and click on technical reports. It will be published soon in Machine Learning. More abbreviated information is in the notes folloowing the set up instructions.

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) but may have hang ups on other f77 compilers.

Random forests does

classification
variable importance (in a number of ways)
computes proximity measures between cases
computes densities
gives a measure of outlyingness for each case

The last two are done for the unsupervised case i.e. no class labels. The density estimation is experimental and has not been thoroughly tested, so treat it with caution. I have used proximities to cluster data and they seem to do a reasonable job.

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

## I. ***Setting Parameters***

The first five 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.
**mult**=1 if you are not doing density estimation or outliers

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 puposes--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 labeled 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
iaddq=add quadratic terms

**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 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 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=1 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.

**iaddq:** iaddq=0 does nothing. iaddq=1 adds quadratic interaction variables to the set of predictor variables . It first centers x(m) at its mean, and then enters additional variables of the form x(k)*x(m) for all k,m with k not equal to m. For some data sets accuracy is improved with the use of these additional variables.

## *Line 3 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 .

**iden**=1,2 computes the density of labelless data with respect to a set of synthetic data having a known distribution.  If iden=1, the synthetic data is sampled independently from the marginals.  For iden=2, the synthetic data is sampled from a product of uniforms the mth of which stretches from the minimum of the mth variable in the data to its maximum.

The sample size of the synthetic data is (mult-1)*nsample0, so mult needs to be set to any integer larger than 1.  If mult>2, then the sample sizes can be equalized by setting pi(1)=pi(2).  Whether taking mult>2 increases accuracy is unknown.

**noutlier**=1 computes an outlingness measure for labelless data.  If this is on, then iprox must also be switched to one, iden set equal to 1 or 2, and mult>1.

***Line 4 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) variuable 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 number of increases.

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

**idenout**=1 prints the following columns to a file
i) case number
ii) class#1 probability
iii) class#2 probability
iv) density estimate

**noutlier**=1 prints the follwing columns to a file
i) case number
ii) measure of outlyingness

LINE 5 Automatic

USER 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.

I have not played with the density estimate p(1| x)/p(2| x) very much. So be carefull in interpreting these. If RF has a high error rate in discriminating between class#1 and the synthetic class #2 the density estimation is not reliable. If iden=1 is used and the error rate is close to 50% or higher, there is low dependence between variables. Similarly the measure of outlyingness has not been well tested. If users try the density estimation and outlyingness measure, I would appreciate comments.

There three measures of variable importance: They complement each other. 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 outline below). But I have found one synthetic data set where the third measure was more sensitive than the first two

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, contructs 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. The next version will. So 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.

## ***<u>Outline Of How Random Forests Works</u>***

### *<u>Usual Tree Construction--Cart</u>*

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.

## ***Random Forests Tools***

A lot of information can be obtained in a single run of Random Forests. 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) Test set error rate.

b) Variable importance

c) Intrinsic proximities between cases

d) Density estimation

e) Outlier detection

I will explain how these function and give applications.

## ***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 three different ways of looking at variable importance.

<u>Measure 1</u>

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.

<u>Measures 2 and 3</u>

For the nth case in the data, its margin at the end of a run is the probability of its true class minus the maximum of the probabilities 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.

## 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.

## *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.

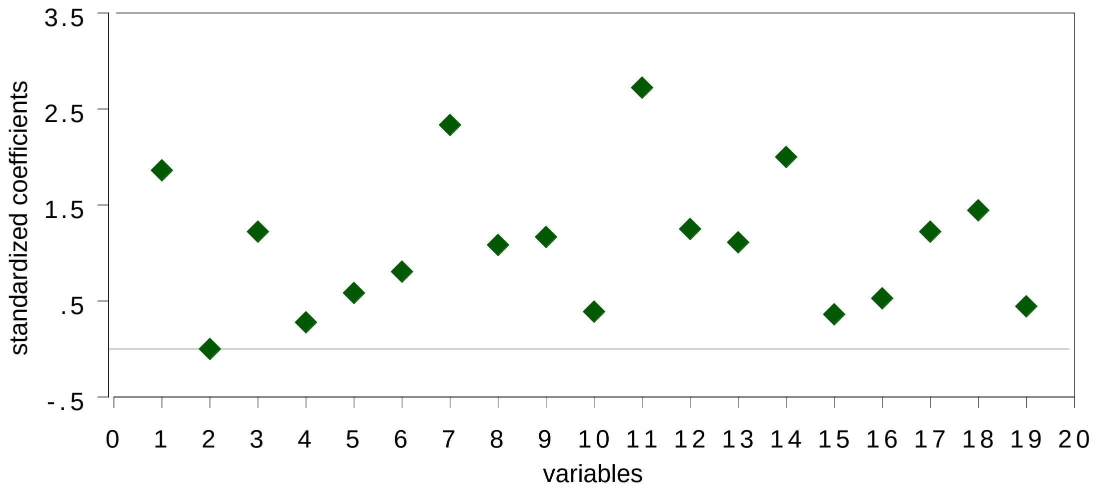
Their conclusion , "Of the four variables originally selected not one was selected in more than 60 percent of the samples. Hence the variables identified in the original analysis cannot be taken too seriously."

## *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

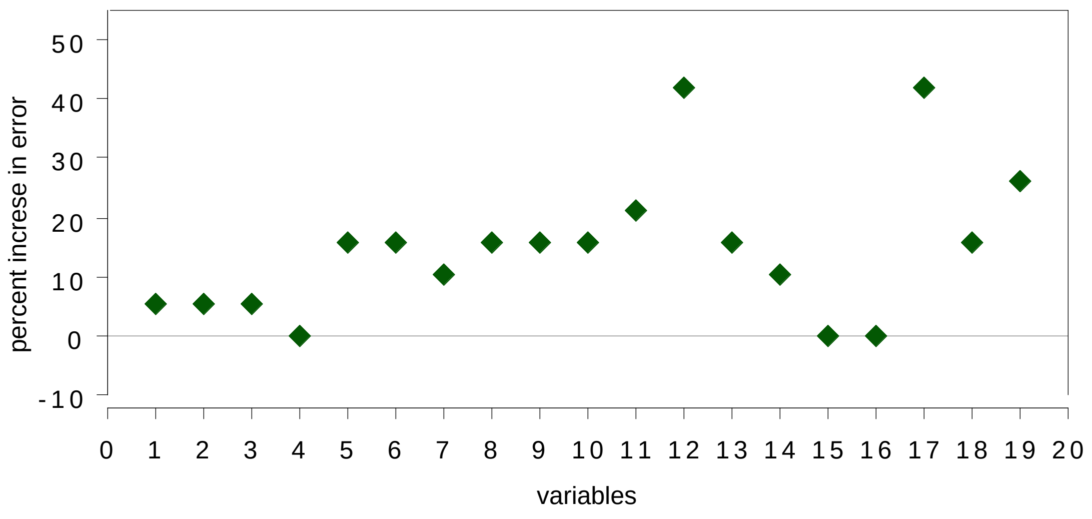


The conclusion is that variables 7 and 11 are the most important covariates. When logistic regression is run using only these two variables, the cross-validated error rate rises to 22.9% .

## *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."

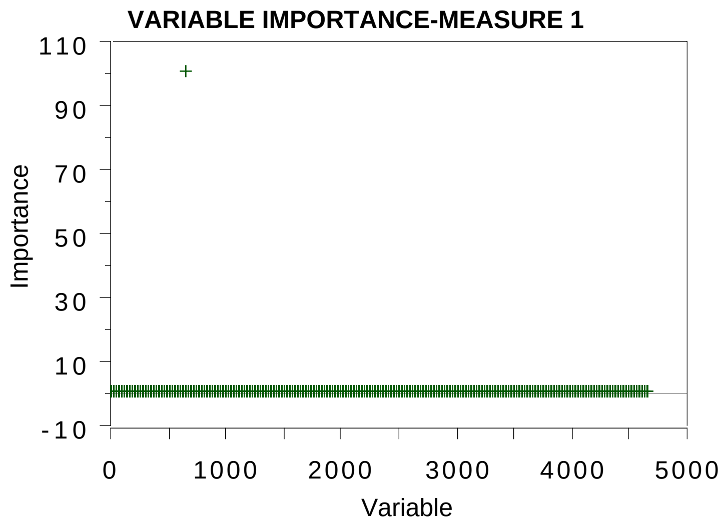
Newer methods in Machine Learning thrive on variables--the more the better.   There is no need for variable selection ,On a sonar data set with 208 cases and 60 variables, Random Forests error rate is 14%.  Logistic Regression has a 50% error rate.

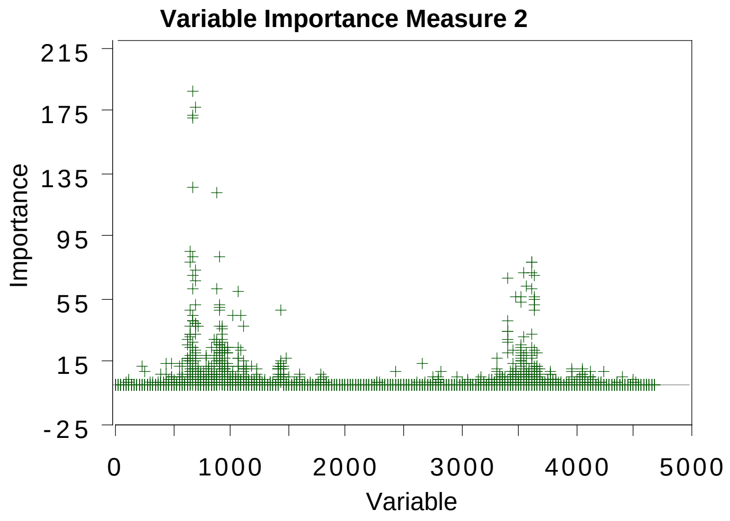
## *Microarray  Analysis*

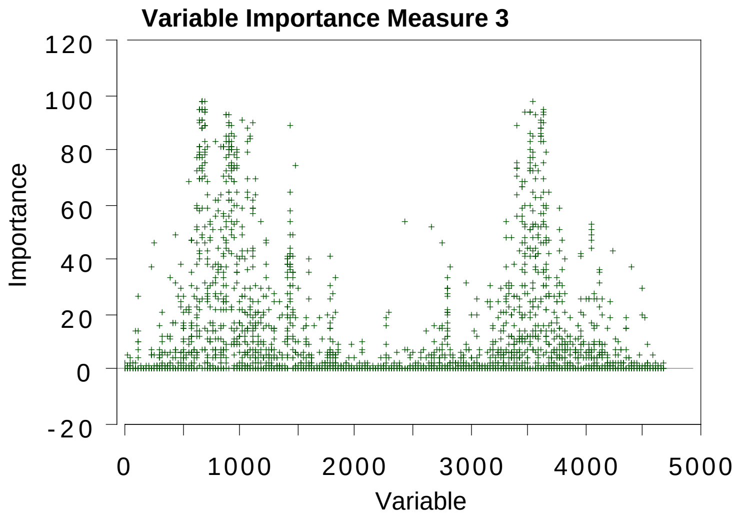
Random forests was run on a microarray lymphoma data set with three classes, sample size of 81 and 4682 variables (genes) without any variable selection.  The error rate was  low (1.2%) using mtry=150.

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.







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 socondary burst of activity higher up. Measure 3 has too mcu sensitivity, fingering too many variables.

## <u>Class probabilities</u>

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. For each member of a test set (with or without class labels), these probabilities are also estimated.

### <u>An Astronomical Example:</u>

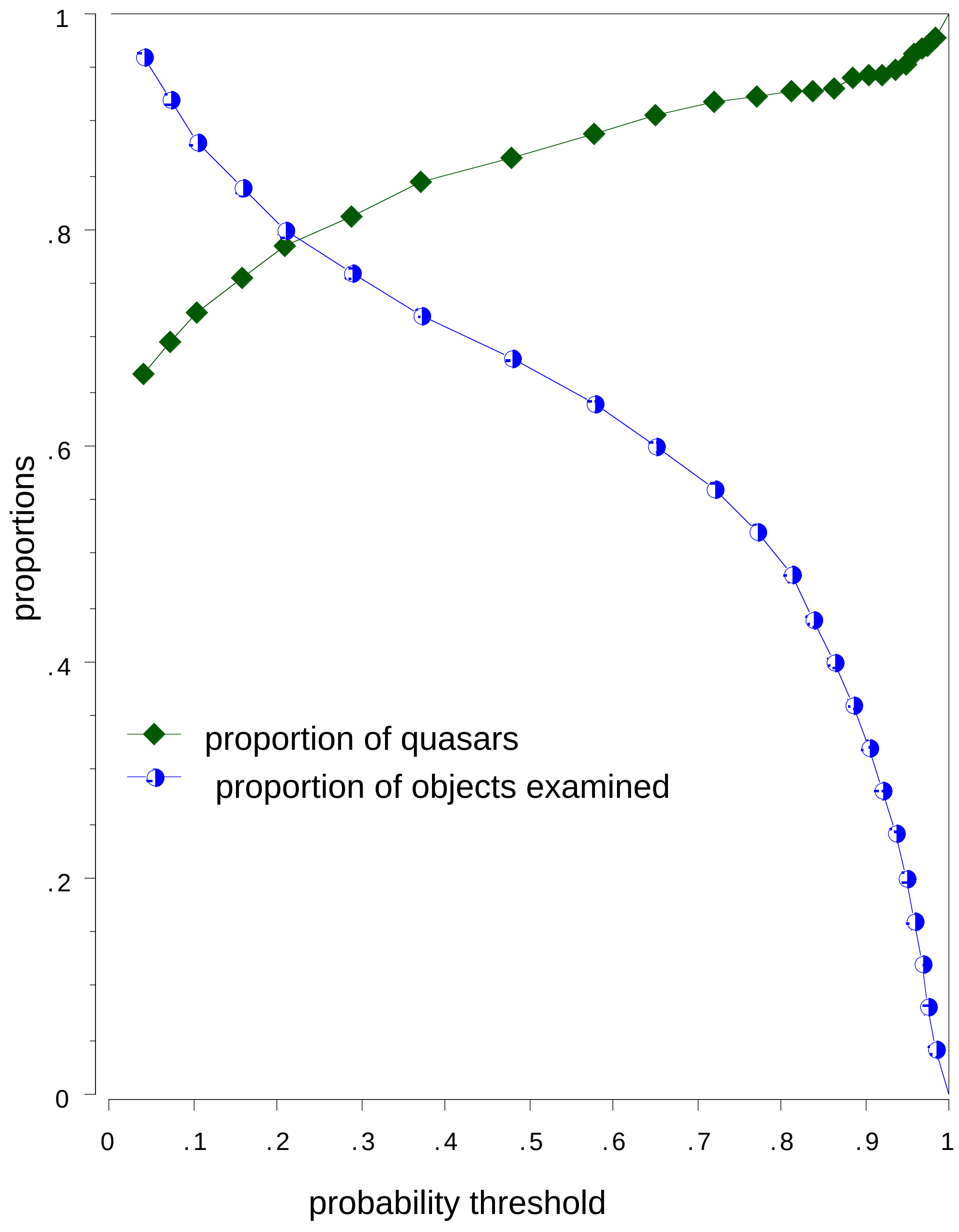
Bob Becker allowed the use of his quasar data set of 2000 astronomical objects of which about half have been verified as quasars.

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 the out-of-bag cases down the tree.

If out-of-bag case i lands in a terminal node occupied the training set case j, then increase the proximity between i and j by one.

In a run that consists of growing J trees, each case will have added about J/3 marks to its proximities.

At the end of the run, the proximities are symmetrized and divided by the number of trees in the run.

## 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 |

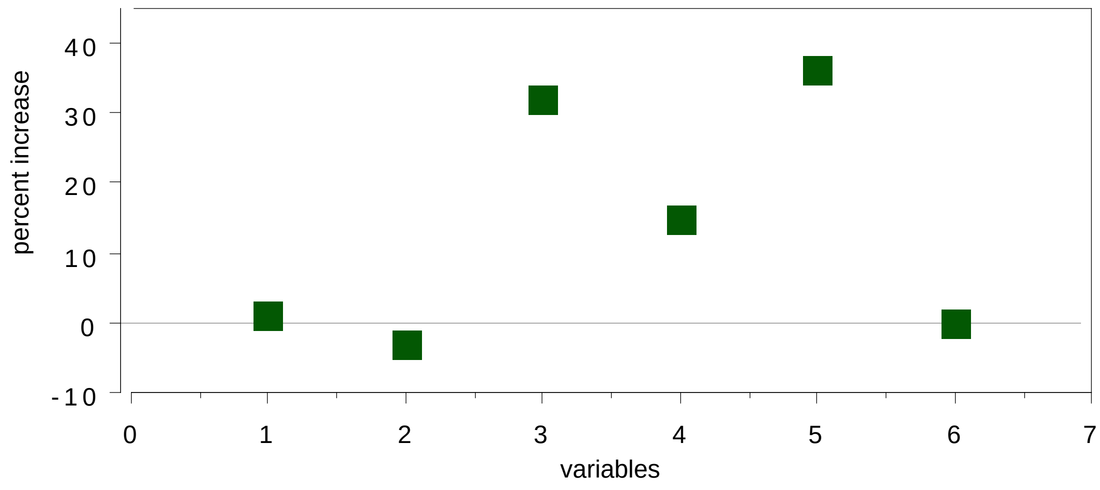
The first five attributes are the results of blood tests thought to be related to liver functioning. The 345 patients are classified into two classes by the severity of their liver disorders.

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



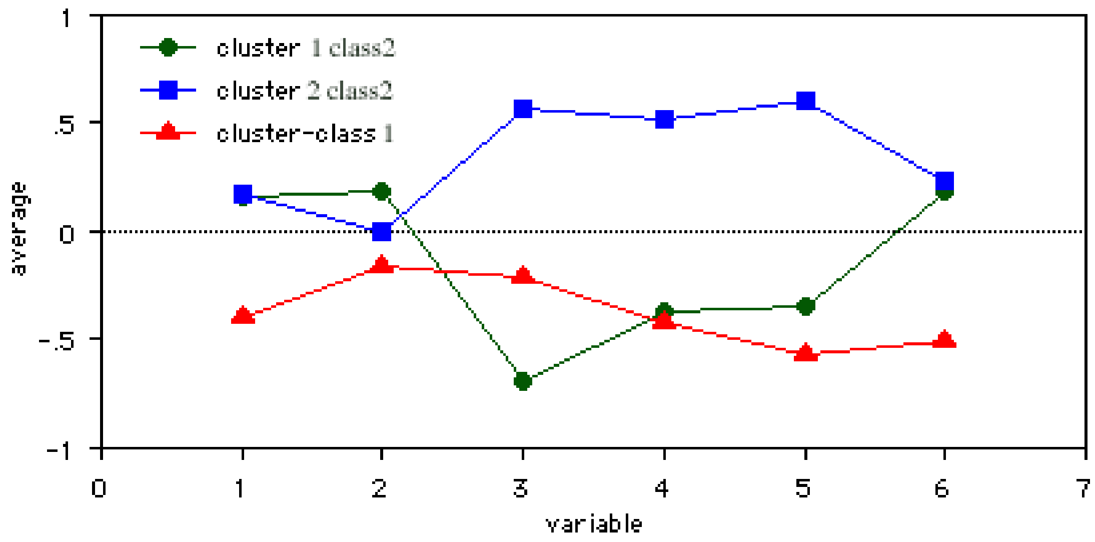
Blood tests 3 and 5 are the most important, followed by test 4.

## 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



Something 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.

## ***Density  Estimation***

Assume data of the form $\{\mathbf{x}_n,\ n=1,...,N\}$. i.e. no response variable and M-dimensional data vectors.

Procedure:

i. Compute all sample univariate distribution functions, $F_1(x)$, $F_2(x)$, ... ,$F_M(x)$.

ii Construct a new data set of size N by sampling $x_1$ at random from $F_1(x)$, $x_2$ from $F_2(x)$, etc. and repeating N times.

Label the original data set class#1. Label the synthetic data set class #2. Now that we have two-class data . run random forests.

The ratio $R(\mathbf{x}_n)=P(1|\mathbf{x}_n)/P(2|\mathbf{x}_n)$ computed from output of RF is an estimate of the density at $\mathbf{x}_n$ of the data in class #1 with respect to the distribution of class #2 .

Note: density estimation has not been tested. We hope to see how accurate it is in higher dimensions using synthetic data where the density is known.

## ***<u>Outlier Location</u>***

Assume data of the form $\{\mathbf{x}_n, n=1,...,N\}$. with no class labels and M-dimensional data vectors. Set up class #2 data as above and run random forests. Between every pair of points $\mathbf{x}_n,\mathbf{x}_k$ in the <u>original</u> data random forests outputs a proximity measure

$$PX(\mathbf{x}_n,\mathbf{x}_k).$$

For each $\mathbf{x}_n$ in the original data compute

$$O(\mathbf{x}_n)=1/\sum_{k\neq n} PX^2(\mathbf{x}_n,\mathbf{x}_k)$$

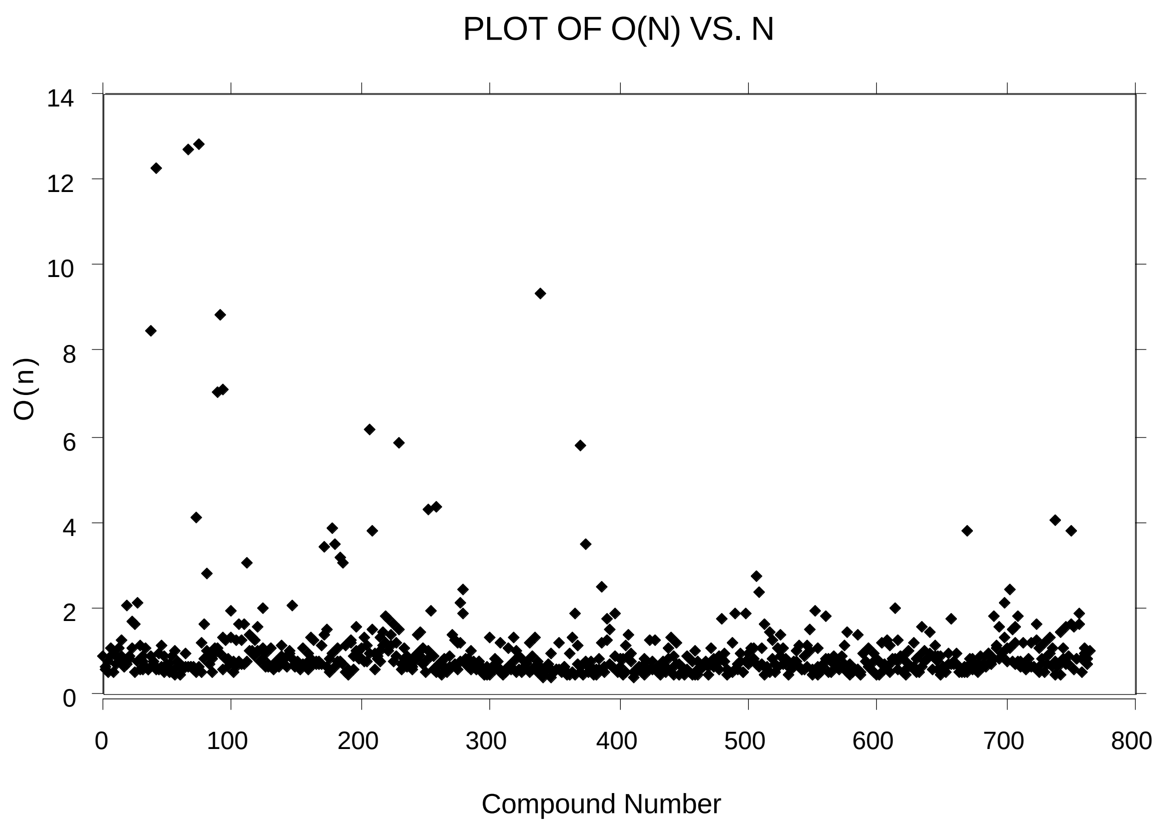
If $\mathbf{x}_n$ has small proximities to the rest of the cases in the original data, then $O(\mathbf{x}_n)$ will be large.

## *<u>An Aplication to Chemical Spectra</u>*

This is data supplied by Merck. It consists of the first 468 spectral intensities in the spectrums of 764 compounds. The challenge presented by Merck was to find the outliers in this data.

The above procedure was applied to the data. There is a 6% misclassification rate between the two classes, indicating a strong dependence structure in the original data.

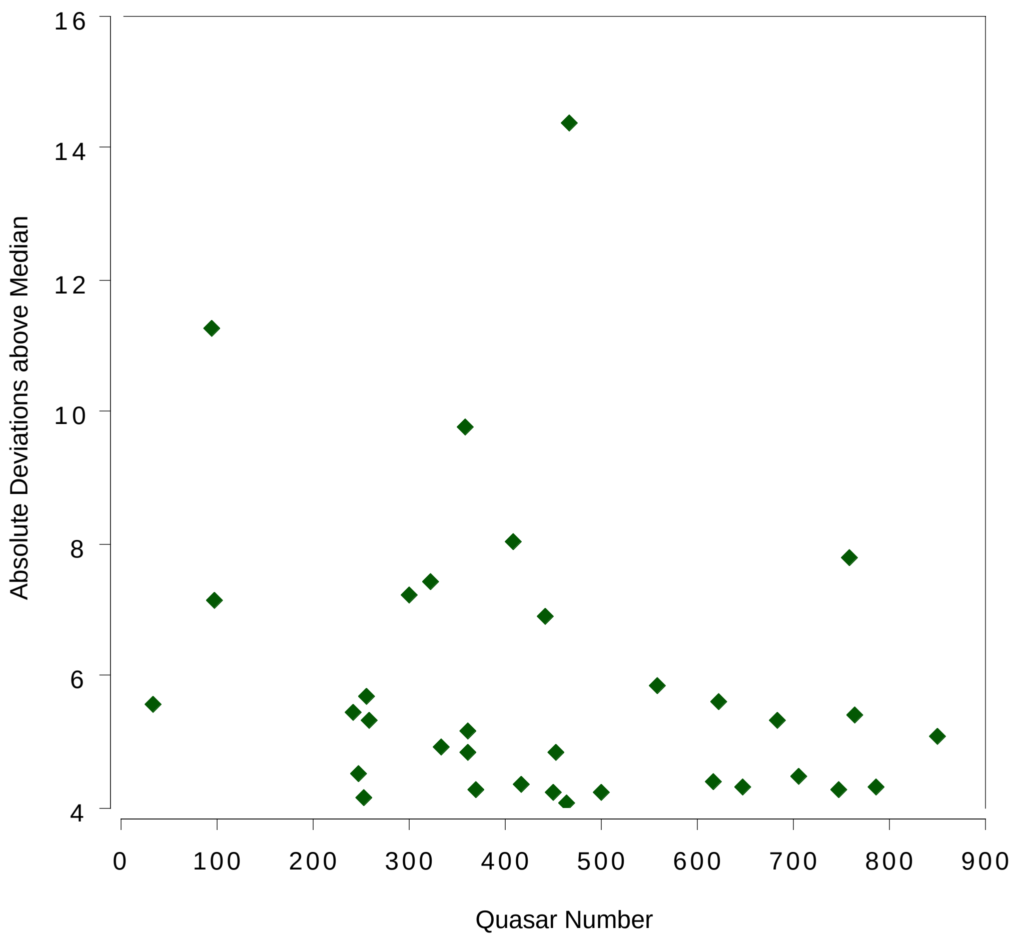
The graph below is the graph of $O(\mathbf{x}_n)$ vs. n for the 764 compounds.



*<u>Outliers in the Quasar Data</u>*

Procedure--the values of O($\mathbf{x}_n$) were normalized by subtracting the median and dividing by the average absolute deviation from the median

OUTLIERS IN QUASAR DATA



Note: two quasars are more than 10 absolute deviations above the median.