Random Forests for Scientific Discovery

Leo Breiman, UC Berkeley
Adele Cutler, Utah State University

Contact Information

- adele@math.usu.edu

Outline

8:30 - 10:15 Part 1

10:15 - 10:30 Refreshment Break

10:30 - 12:00 Part 2

12:00 - 1:30 Lunch (on our own)

1:30 - 3:15 Part 3

3:15 - 3:30 Refreshment Break

3:30 - 5:00 Part 4

Part 1 Introduction and Definitions

1. Introduction

2. Fundamental paradigm
   - error, bias, variance
   - randomizing “weak” predictors: some two-dimensional illustrations
   - unbiasedness in higher dimensions.

3. Definition of Random Forests
   - the randomization used
   - properties as a classification machine.

Part 2 Classification in Depth

4. Using the out-of-bag data to
   - estimate error
   - choose **m**
   - find important variables.

5. Using proximities to
   - visualize the forest
   - replace missing values
   - identify mislabeled data, outliers, novel cases
   - compute prototypes.

6. Learning from unbalanced data.

Part 3 Visualization

7. Local variable importance.

8. Visualization.

Part 4 Other Applications

9. Random forests for unsupervised learning.

10. Random forests for regression.

11. Random forests for survival analysis.

12. Some Case Studies
    - Autism
    - Dementia
    - Metabolomics
    - Microarrays

Part 1

Introduction and Definitions

The Data Avalanche

We can gather and store larger amounts of data than ever before:

- satellite data
- web data
- EPOS
- microarrays etc
- text mining and image recognition.

Who is trying to extract meaningful information from these data?

- academic statisticians
- machine learning specialists
- *people in the application areas!*

Data Analysis Tools

- SAS
- S-Plus
- SPSS
- R
- Other scattered packages.

Much of what we teach non-majors, as well as almost all of what is available to them in mainline statistical software, is at least two generations old.

We should be able to do better!

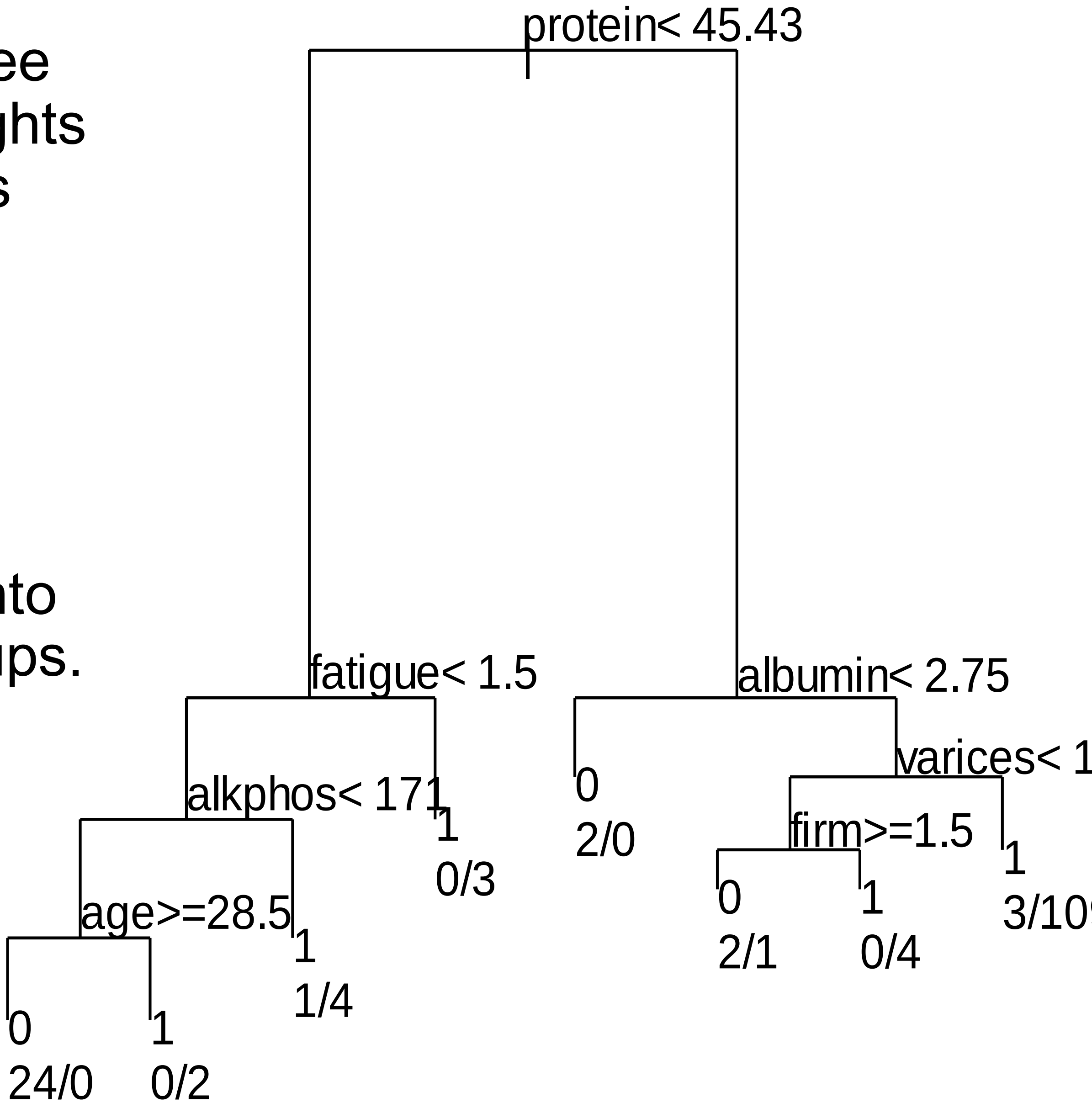
CART (Breiman, Friedman, Olshen, Stone 1984)

Arguably one of the most successful tools of the last 20 years. Why?

1. Universally applicable to both classification and regression problems with no assumptions on the data structure.
2. Can be applied to large datasets. Computational requirements are of order MNlogN, where N is the number of cases and M is the number of variables.
3. Handles missing data effectively.
4. Deals with categorical variables efficiently.

5. The picture of the tree gives valuable insights into which variables are important and where.

6. The terminal nodes suggest a natural clustering of data into homogeneous groups.



Drawbacks of CART

- *Accuracy* - current methods, such as support vector machines and ensemble classifiers often have 30% lower error rates than CART.

- *Instability* – if we change the data a little, the tree picture can change a lot. So the interpretation is not as straightforward as it appears.

Today, we can do better!

What do we want in a tool for the sciences?

- Universally applicable for classification and regression
- Unexcelled accuracy
- Capable of handling large datasets
- Effective handling of missing values

$\left.\right\}$ minimum

- Variable importance
- Interactions
- What is the shape of the data?
- Are there clusters?
- Are there novel cases or outliers?
- How does the multivariate action of the variables separate the classes?

Random Forests

- General-purpose tool for classification and regression
- Unexcelled accuracy - about as accurate as support vector machines *(see later)*
- Capable of handling large datasets
- Effectively handles missing values
- Gives a wealth of scientifically important insights

Random Forests Software

- Free, open-source code (fortran, java)
  www.stat.berkeley.edu/users/breiman/RandomForests

- Commercial version (academic discounts)
  www.salford-systems.com

- R interface, independent development (Andy Liaw and Matthew Wiener)

The Fundamental Paradigm

Given a set of training data

$$\mathbf{T} = \{ (\mathbf{y}_n, \mathbf{x}_n) \text{ iid from } (\mathbf{Y}, \mathbf{X}),\ n=1,\ldots,N \}$$

define the prediction error as

$$PE(f, \mathbf{T}) = E_{Y,X} L(\mathbf{Y}, f(\mathbf{X},\mathbf{T}))$$

where $L(\mathbf{Y}, f(\mathbf{X},\mathbf{T}))$ measures the loss between $\mathbf{Y}$ and $f(\mathbf{X},\mathbf{T})$.

We want to find a function f so that the prediction error is small.

The Fundamental Paradigm (ctnd)

Usually, Y is one-dimensional.

If Y represents unordered labels, the problem is classification and the loss is 0/1 loss.

If Y is numerical, the problem is regression and loss is squared error.

Bias and Variance in Regression

A random variable $\mathbf{Y}$ related to a random vector $\mathbf{X}$ can be expressed as

$$\mathbf{Y} = f^{\star}(\mathbf{X}) + \varepsilon$$

where $f^{\star}(\mathbf{X}) = E(\mathbf{Y} | \mathbf{X})$, $E(\varepsilon | \mathbf{X}) = 0$.

This decomposes Y into

- $f^{\star}(\mathbf{X})$ structural part, can be predicted in terms of $\mathbf{X}$
- unpredictable noise ε.

Bias and Variance in Regression (ctnd)

The mean squared generalization error of a predictor $f(\mathbf{x},\mathbf{T})$ is

$$PE(f(\bullet,\mathbf{T})) = E_{Y,X}(\mathbf{Y} - f(\mathbf{X},\mathbf{T}))^2$$

where the subscripts indicate expectation with respect to **Y** and **X**, holding **T** fixed. So the mean squared generalization error of f is

$$PE^{\star}(f) = E_{T}(PE(f(\bullet,\mathbf{T}))).$$

Bias and Variance in Regression (ctnd)

Now let $\overline{f}(\mathbf{x})$ be the average over training sets of the predicted value at **x**, that is

$$\overline{f}(\mathbf{x}) = E_T(f(\mathbf{x},\mathbf{T})).$$

The bias-variance decomposition is

$PE^\star(f) = E(\boldsymbol{\varepsilon}^2)$ *noise variance*

$+ E_X(f^\star(\mathbf{X}) - \overline{f}(\mathbf{X}))^2$ *bias squared*

$+ E_{X,T}(f(\mathbf{X},\mathbf{T}) - \overline{f}(\mathbf{X}))^2$ *variance.*

Weak Learners

Definition: a *weak learner* is a prediction function that has low bias.

Generally, low bias comes at the cost of high variance, so weak learners are usually not accurate predictors.

Examples:

- smoothers with low bias, high noise
- join the dots
- large trees.

A two-dimensional example

$N = 200$

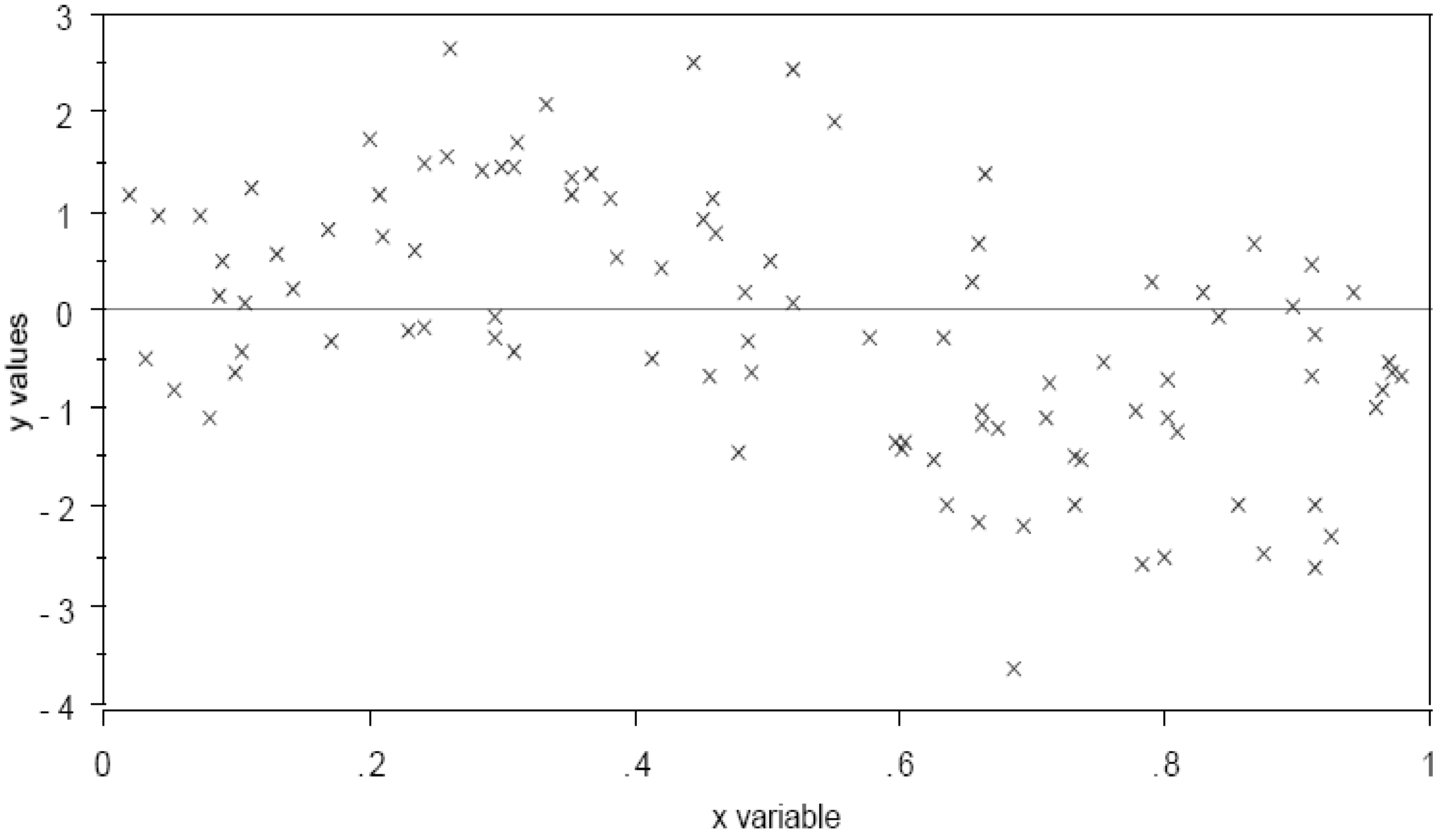
$x_1,\ldots,x_N$ independent, $U[0,1]$

$z_1,\ldots,z_N$ independent, $N(0,1)$

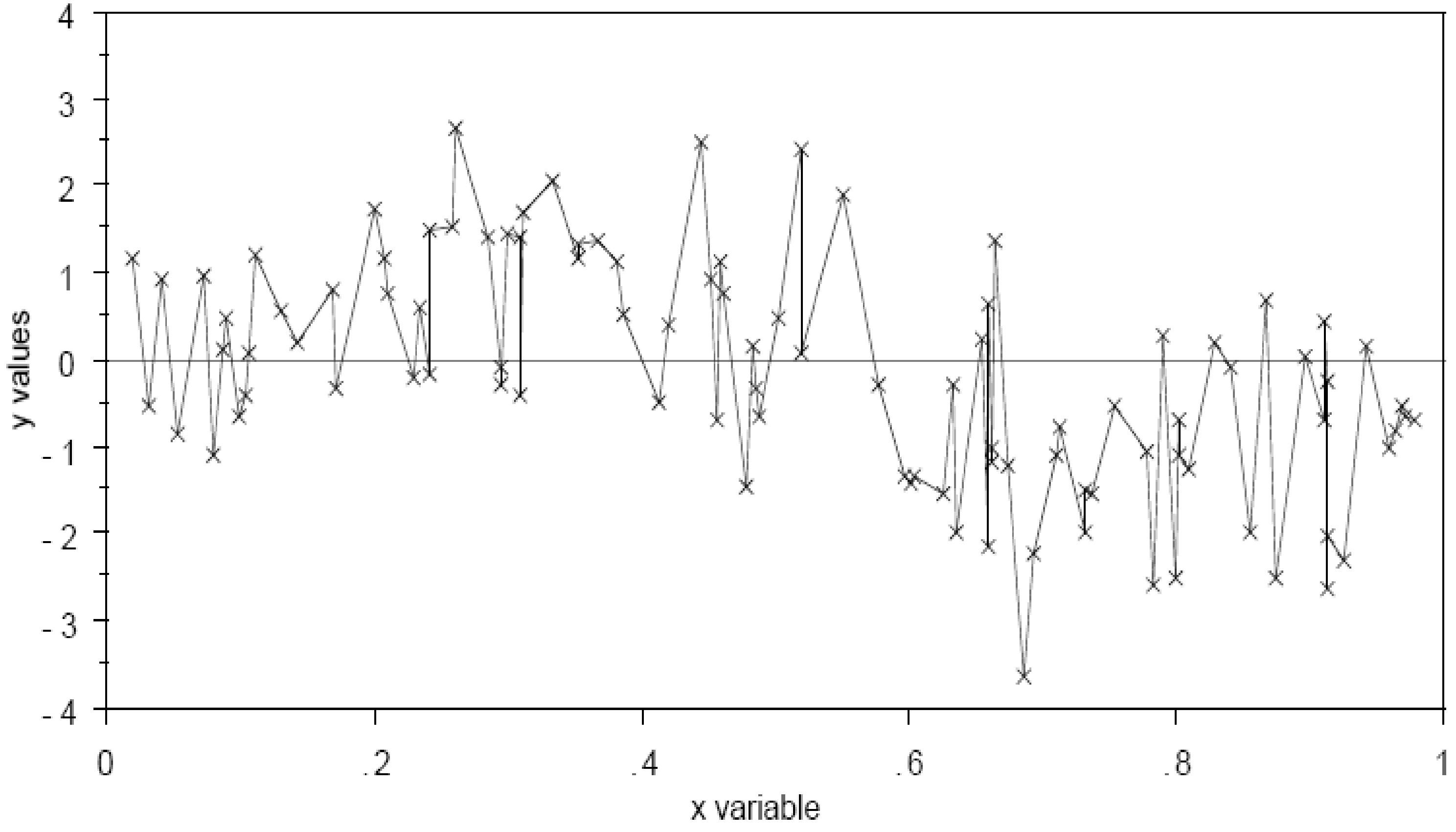
$y_n = \sin(2\pi\, x_n) + z_n$

Weak learner: join the dots between ordered x_n's.

First training set

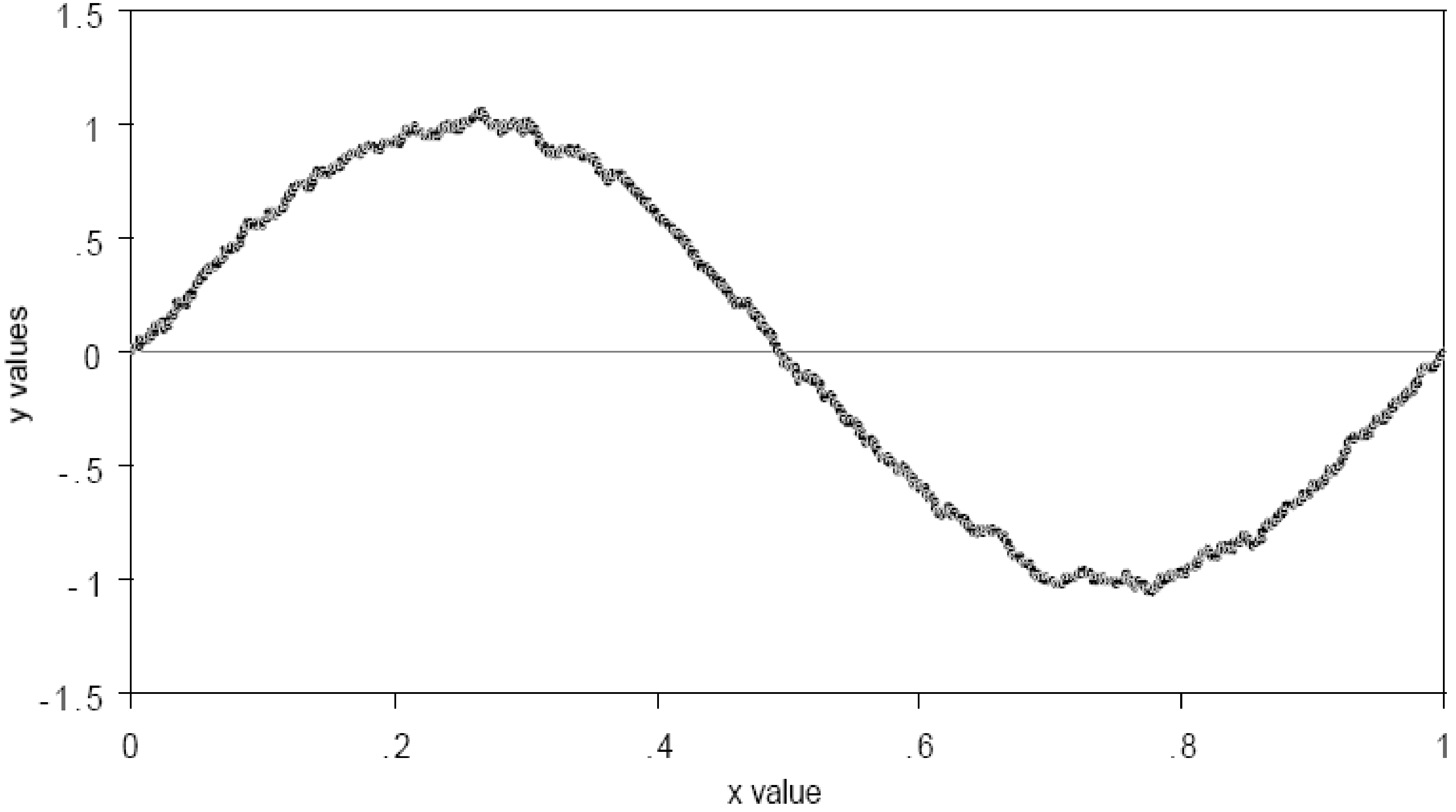


Weak learner for first training set



The weak learner is almost unbiased, but with large variance.

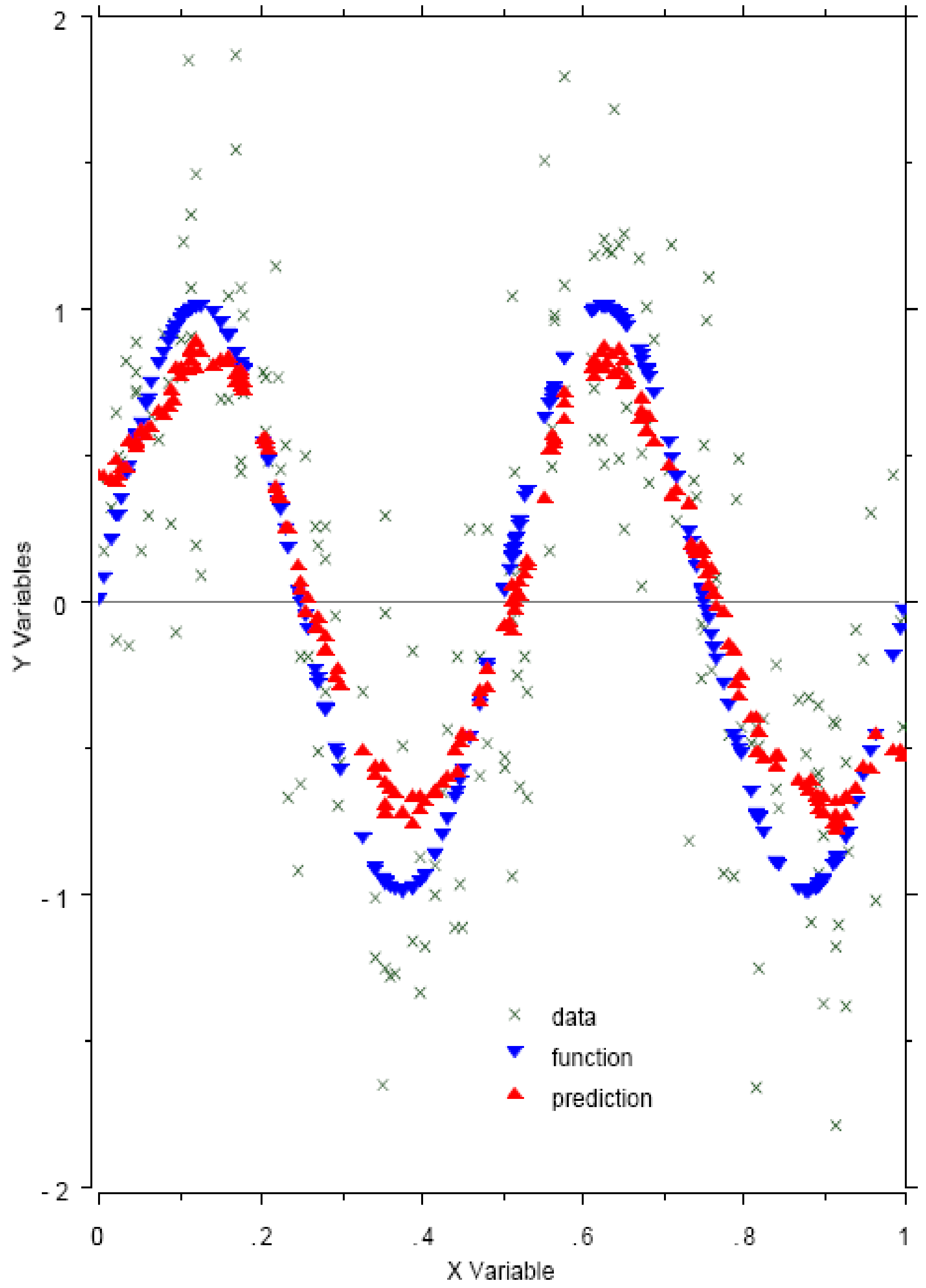
Average of weak learners over 1000 training sets

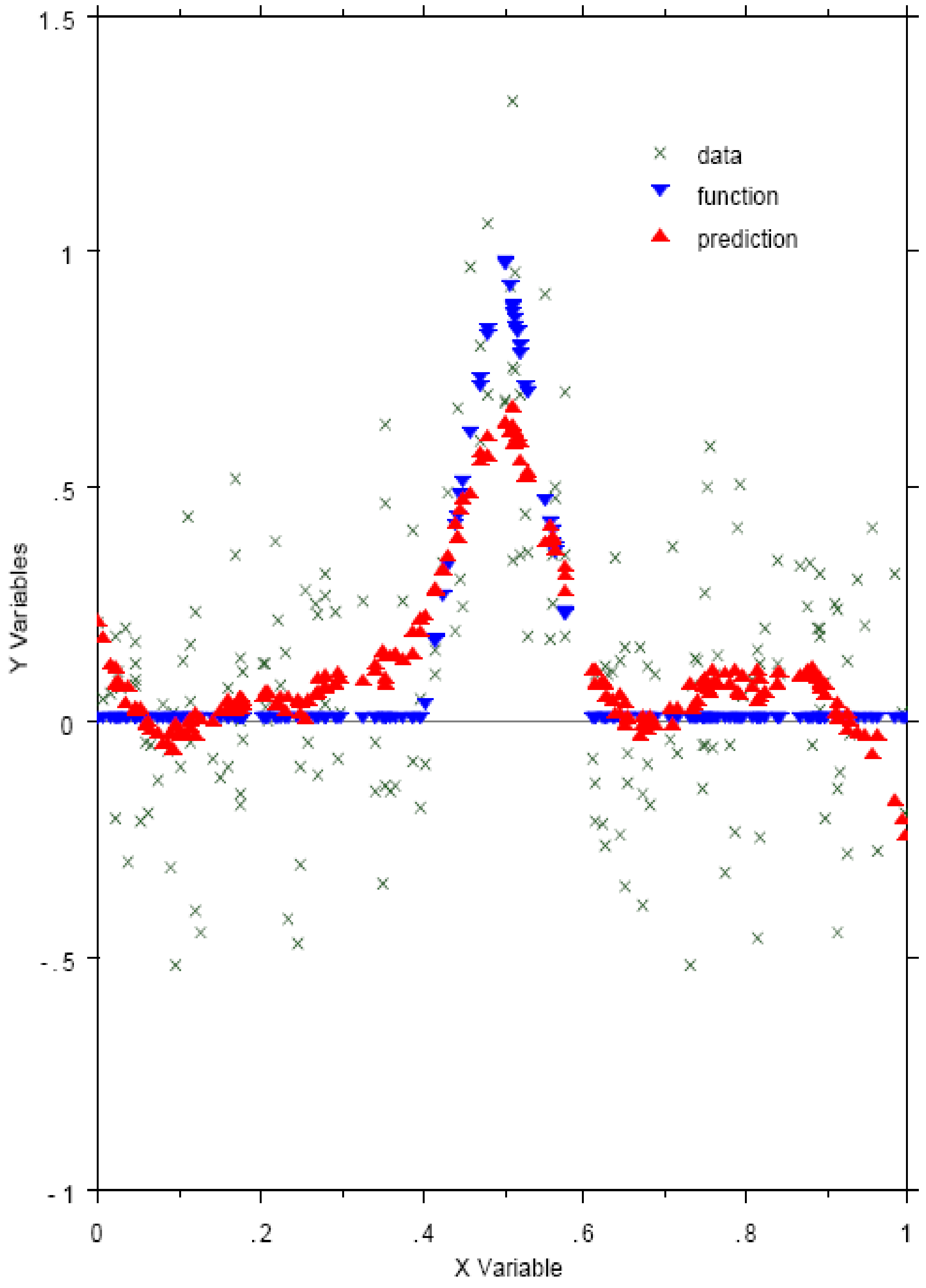


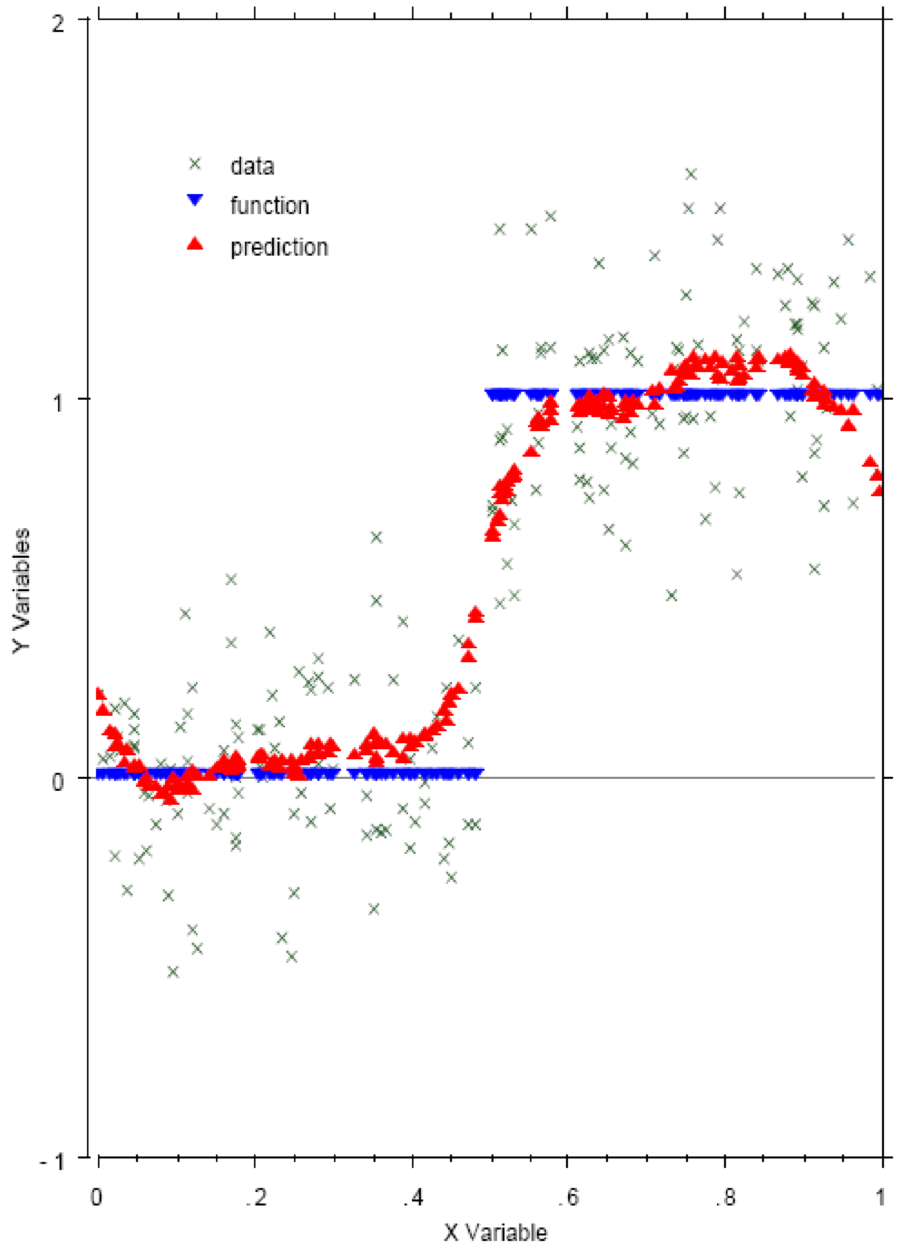
The average approximates the underlying function.

Averaging Weak Learners

- Weak learners have low bias but high variance.
- Averaging weak learners, over many training sets, gives estimates with low bias *and* low variance.
- Replicate training sets are rarely available, but consider the next set of examples…


2
1
0
- 1
- 2
Y Variables
0
.2
.4
.6
.8
1
X Variable
data
function
prediction


1.5
1
.5
0
-.5
-1
0
.2
.4
.6
.8
1
Y Variables
X Variable
data
function
prediction

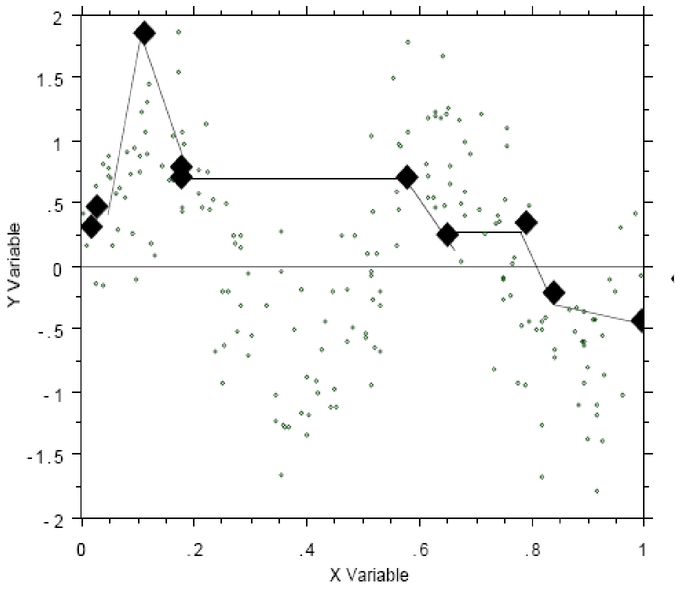

2
1
0
- 1
Y Variables
0
.2
.4
.6
.8
1
X Variable
data
function
prediction

How did we do it?

We used 1000 weak learners.

Each weak learner was formed by selecting two-thirds of the cases at random and using the join-the-dots strategy.

Then we averaged these 1000 weak learners.



Averaging randomized predictors

The kth weak learner is of the form $f_k(\mathbf{x},\mathbf{T}) = f(\mathbf{x},\mathbf{T},\boldsymbol{\Theta}_k)$ where $\boldsymbol{\Theta}_k$ is the random vector that selects the points to be in the weak learner, $k=1,\ldots,K$.

The $\boldsymbol{\Theta}_k$ are independent and identically distributed.

The ensemble predictor is $F(\mathbf{x},\mathbf{T}) = \sum_k f(\mathbf{x},\mathbf{T},\boldsymbol{\Theta}_k) /K$.

Breiman (Machine Learning 2001) shows that

$$\mathrm{Var}(F) = E_{\mathbf{x},\boldsymbol{\Theta},\boldsymbol{\Theta}'} [\rho_T(f(\mathbf{x},\mathbf{T},\boldsymbol{\Theta})f(\mathbf{x},\mathbf{T},\boldsymbol{\Theta}'))] \mathrm{Var}_T(f(\mathbf{x},\mathbf{T},\boldsymbol{\Theta}))$$

$$\mathrm{Bias}^2 (F) \leq E_{\boldsymbol{\Theta}} \mathrm{bias}^2 f(\mathbf{x},\mathbf{T},\boldsymbol{\Theta}) + E(\boldsymbol{\varepsilon}^2) .$$

Bias

$$\begin{aligned}\mathrm{Bias}^2(\,F\,) &= E_{yx}(\,\mathbf{y} - E_{T,\Theta}\, f(\mathbf{x},\mathbf{T},\boldsymbol{\Theta})\,)^2 \\ &\leq E_{yx\Theta}(\,\mathbf{y} - E_{T}\, f(\mathbf{x},\mathbf{T},\boldsymbol{\Theta})\,)^2 \\ &\leq E_{\Theta}\, \mathrm{bias}^2\, f(\mathbf{x},\mathbf{T},\boldsymbol{\Theta}) + E(\varepsilon^2)\end{aligned}$$

The first term is just the squared bias of a single predictor, so if the predictors are almost unbiased, it will be small.

The second part is unavoidable.

Correlation

So we are left with

Var(F) =

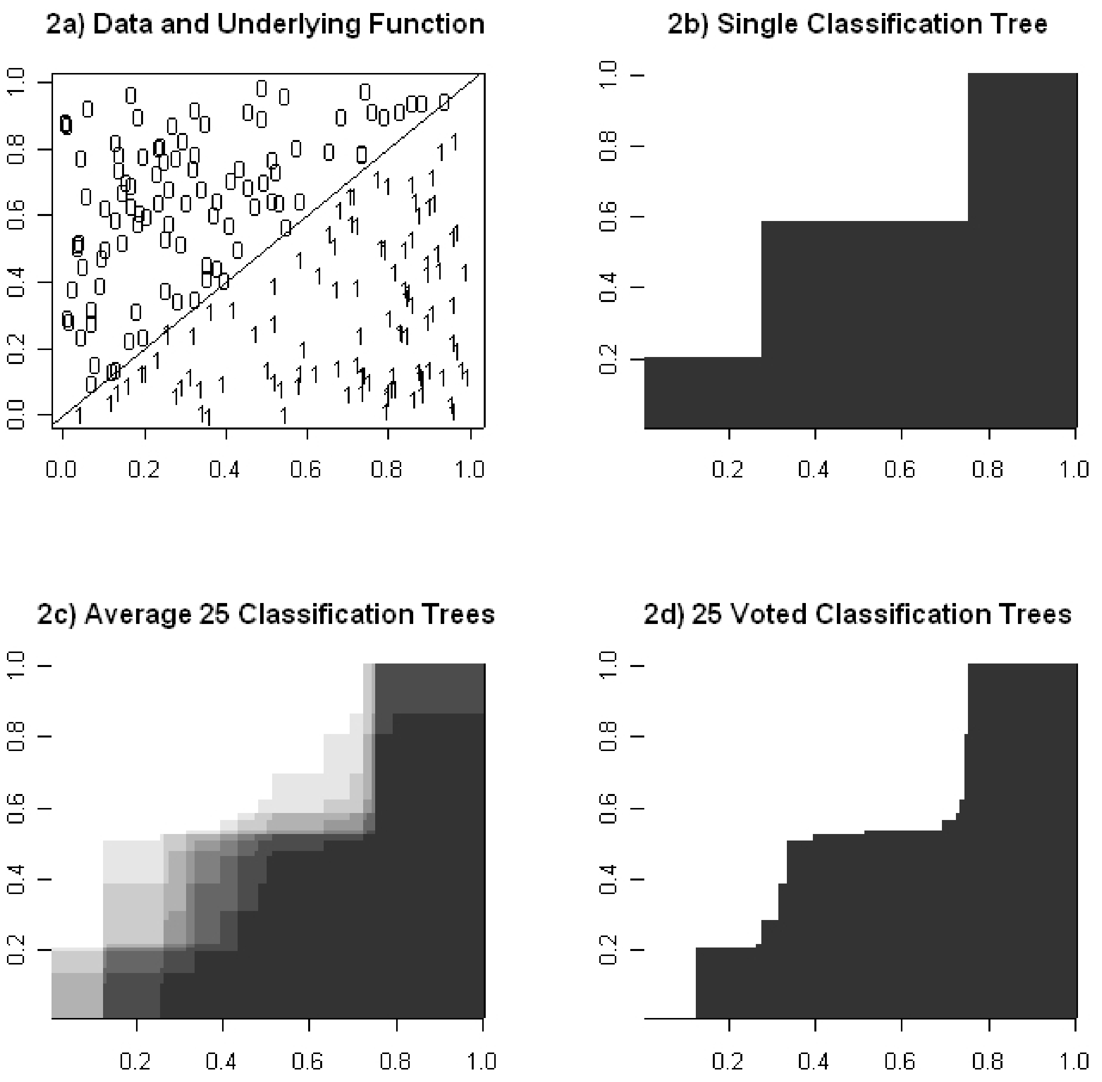
$E_{\mathbf{x},\boldsymbol{\Theta},\boldsymbol{\Theta}'}[\rho_T(f(\mathbf{x},\mathbf{T},\boldsymbol{\Theta})f(\mathbf{x},\mathbf{T},\boldsymbol{\Theta}'))] \, \mathrm{Var}_T(f(\mathbf{x},\mathbf{T},\boldsymbol{\Theta}))$

The first part is the mean correlation.

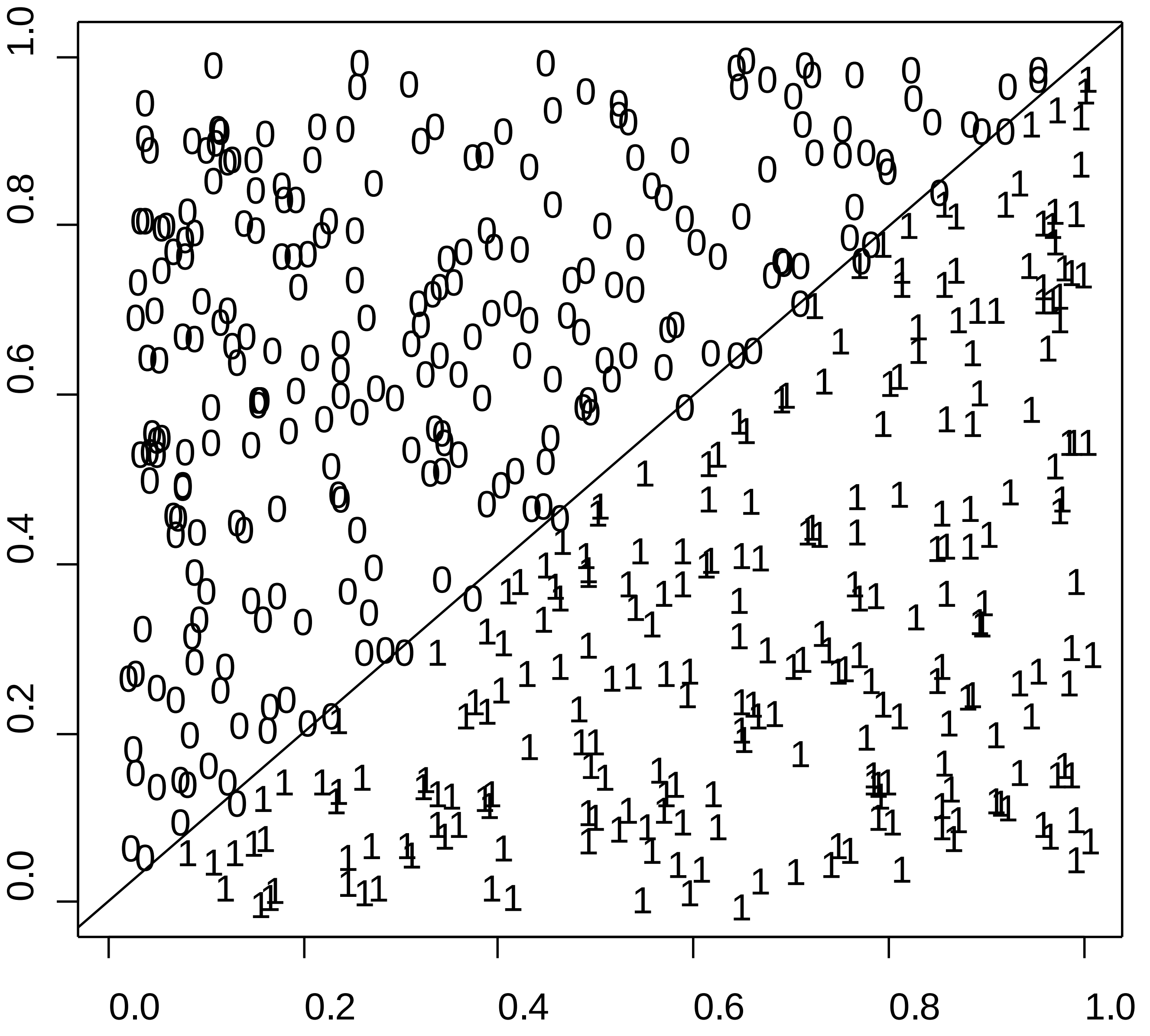
We want it to be small. What does this mean?

Fix **x** and T. Suppose someone tells you that $f(\mathbf{x},\mathbf{T},\boldsymbol{\Theta})$ is on the high side, then you don't learn much about $f(\mathbf{x},\mathbf{T},\boldsymbol{\Theta}')$.

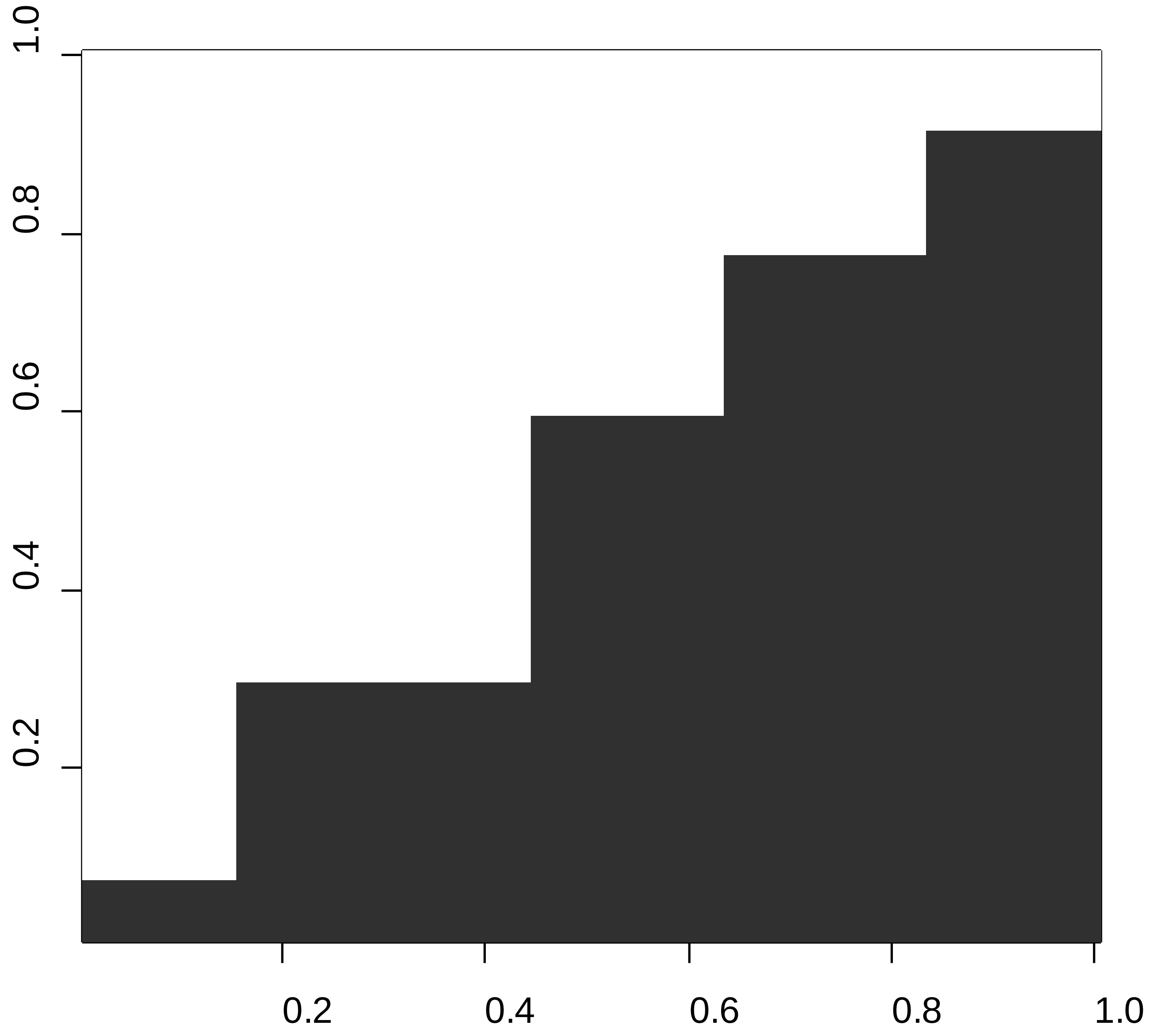
Randomized Classification Trees



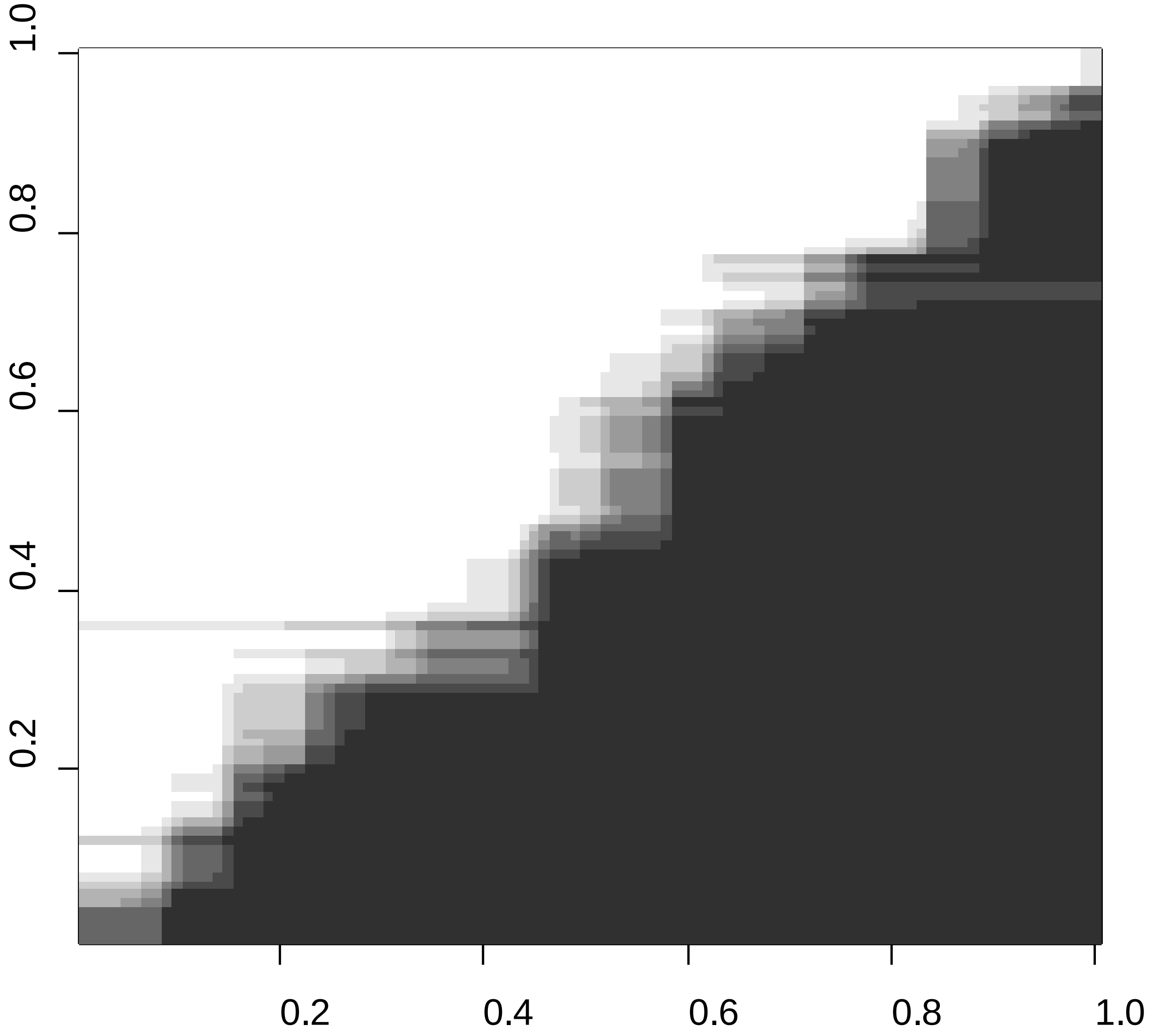
Data and Underlying Function



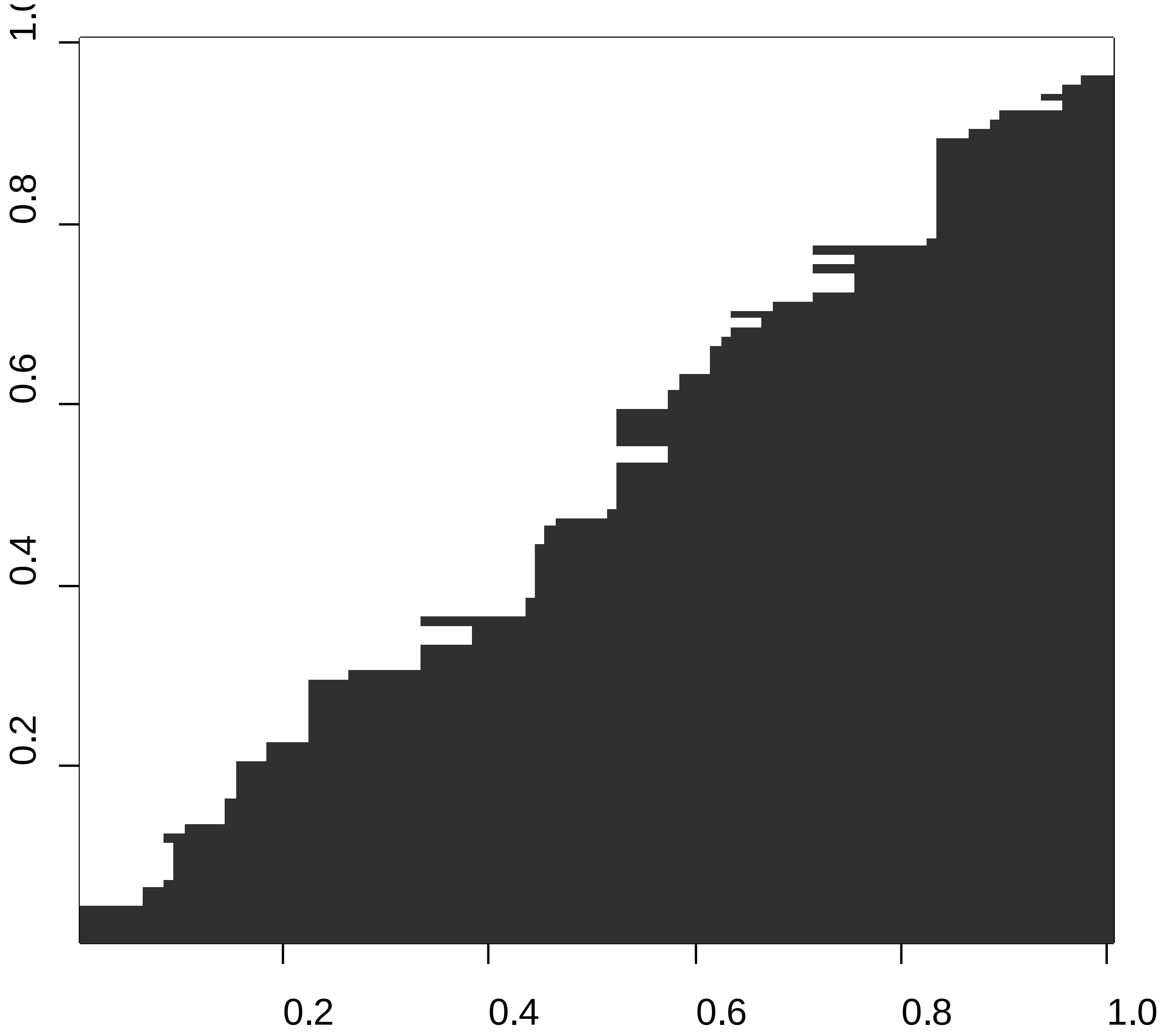
Single Classification Tree (all data)



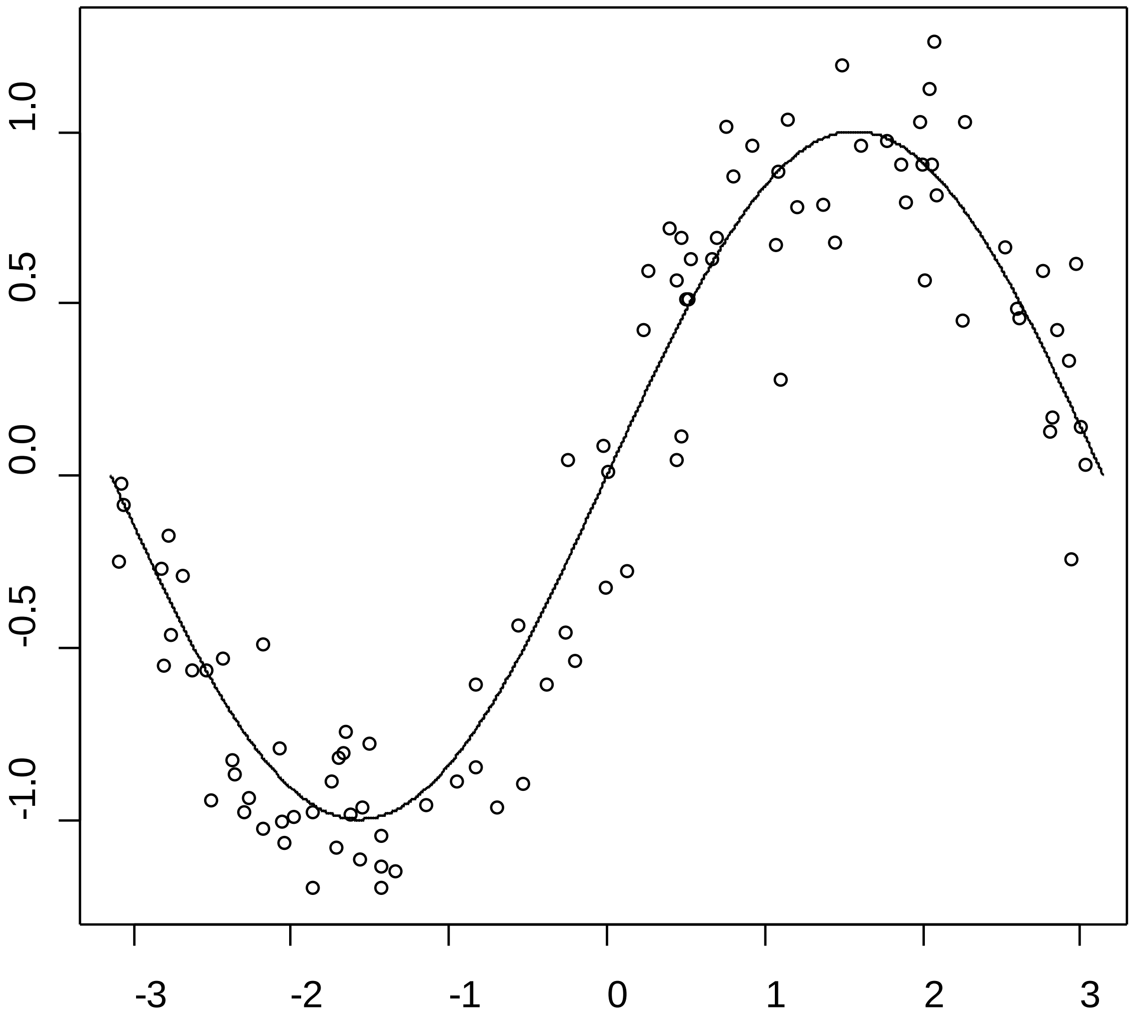
Average of 25 Classification Trees (fit to boostrap samples)



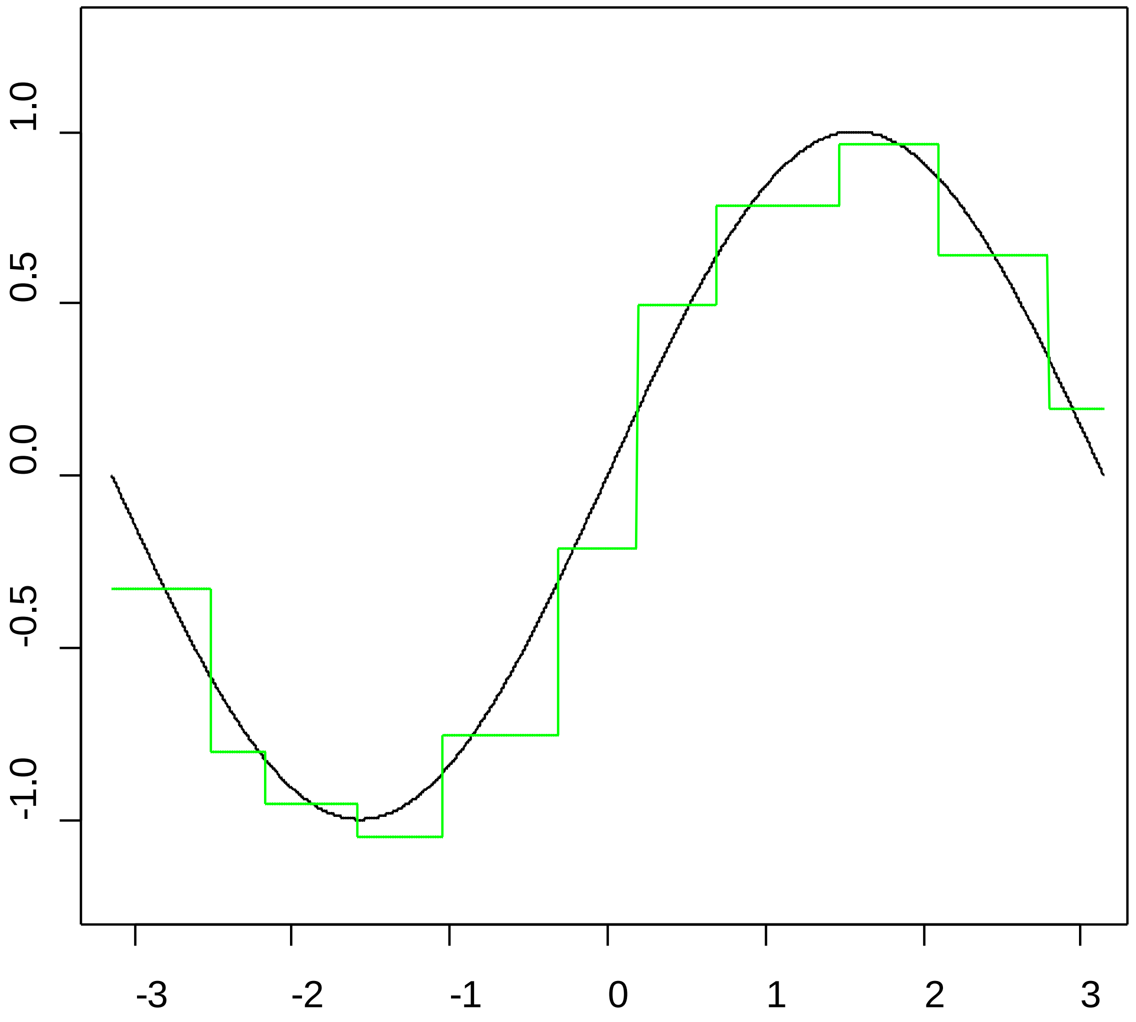
25 Voted Classification Trees (fit to boostrap samples)



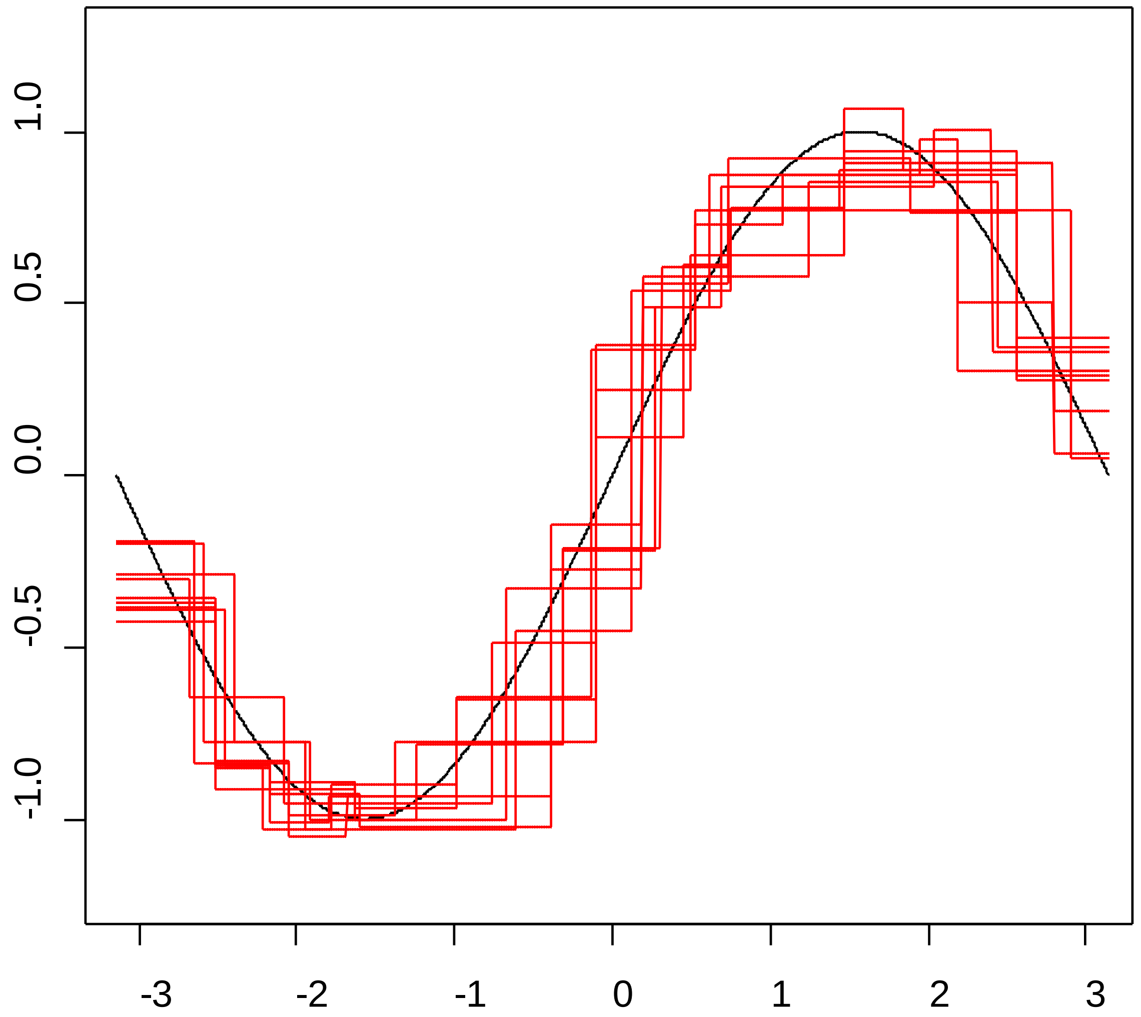
Data and Underlying Function



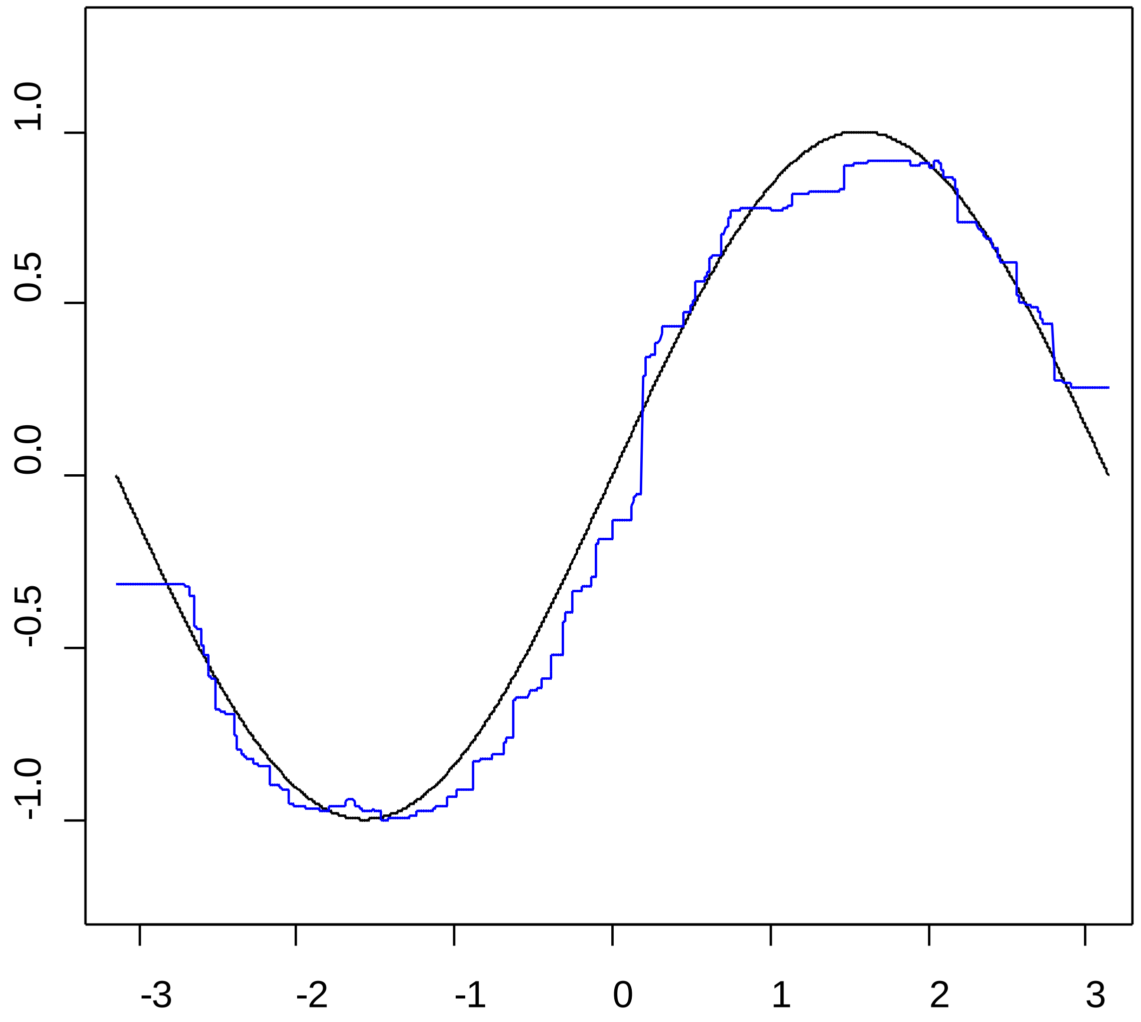
Single Regression Tree (all data)



10 Regression Trees (fit to boostrap samples)



Average of 100 Regression Trees (fit to bootstrap samples)

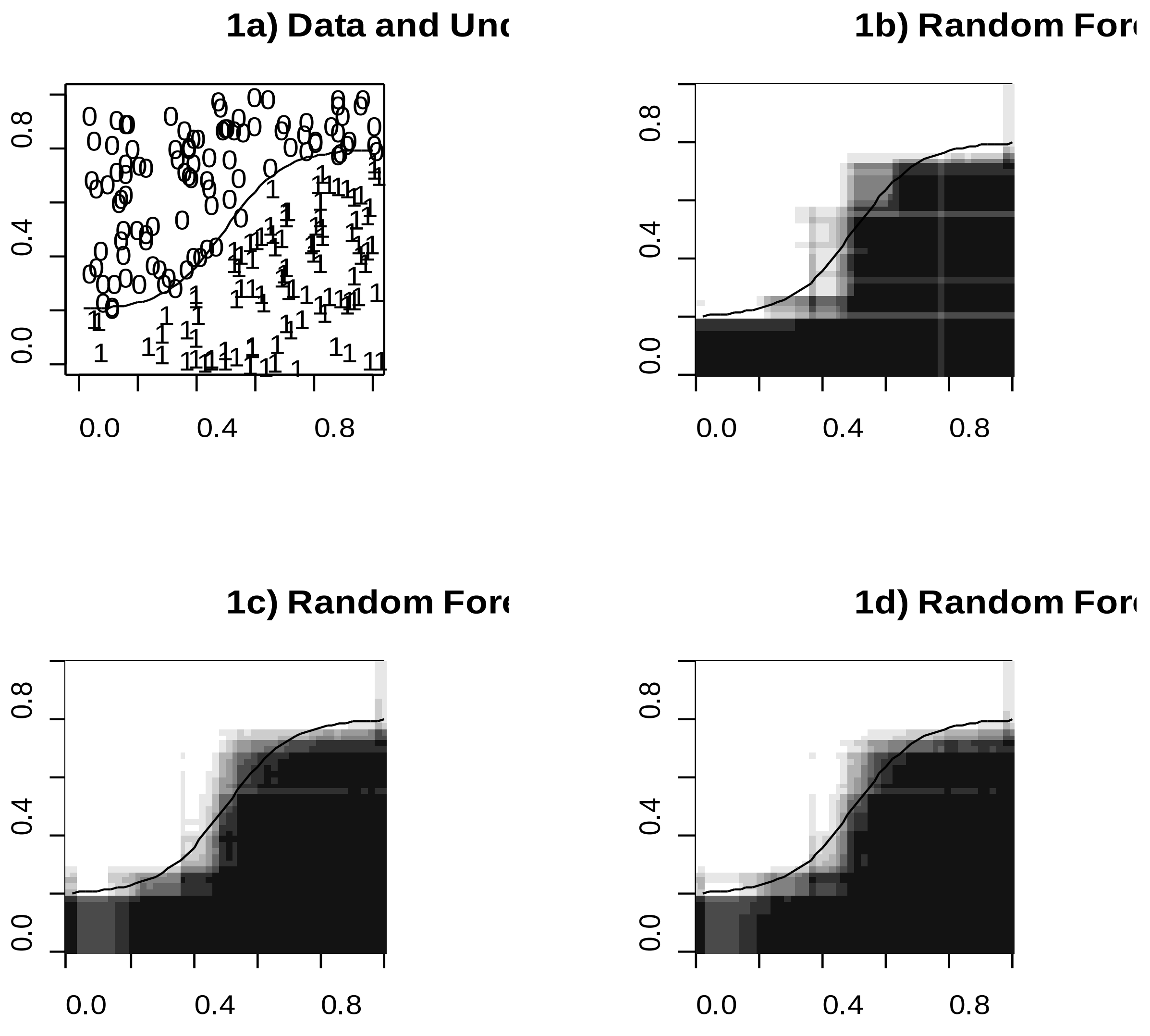


The Message

We win big by averaging i.i.d. randomized predictors as long as their correlation and bias are low.

Random forests are examples of i.i.d. randomization applied to binary classification/regression trees…

Random Forests



What is/are Random Forests?

A random forest (RF) is a collection of tree predictors

$$f(\mathbf{x}, \mathbf{T}, \Theta_k), \quad k = 1,\ldots,K$$

where the Θ_k are i.i.d random vectors.

The trees are combined by

- voting (for classification)
- averaging (for regression).

How do we Grow the Trees?

The key to accuracy is *low bias* and *low correlation*.

Low bias: grow trees to maximum depth.

Low correlation:

- Grow each tree on an independent bootstrap sample from the training data.
- At each node:
    1. Randomly select **m** variables out of all **M** possible variables.
    2. Find the best split on the selected **m** variables.

Overfitting

The Law of Large Numbers insures convergence as $k \to \infty$.

The test set error rates (modulo a little noise) are monotonically decreasing and converge to a limit.

There is no overfitting as the number of trees increases!

Properties as a Classification Machine

1. Excellent accuracy.
   - In tests on collections of data sets it has accuracy at least as good as Adaboost and support vector machines.

2. Fast.
   - With 100 variables, 100 trees in a forest can be grown in the same time as growing 3 single CART trees.

3. Does not overfit.

(ctnd)

4. Handles
   - thousands of variables
   - many-valued categoricals
   - extensive missing values
   - badly unbalanced data sets.

5. Gives an internal unbiased estimate of test set error as trees are added to the ensemble.

Part 2

Classification in Depth

Random Forests

Grow each tree on an independent bootstrap sample from the training data.

At each node:

1. Randomly select **m** variables out of all **M** possible variables (independently for each node).
2. Find the best split on the selected **m** variables.

Grow the tree to maximum depth.

Vote or average the trees to get predictions.

Two Natural Questions

1. ***Why bootstrap?***

   Fitting our randomized trees to the training data is about as accurate, so why bootstrap?

   OUT-OF-BAG DATA

2. ***Why trees?***

   There are many weak learners which could work, so why use trees?

   PROXIMITIES

Out-of-bag Data

For each tree in the forest, we select a bootstrap sample from the data.

The bootstrap sample is used to grow the tree.

The remaining data are said to be “out-of-bag” (about one-third of the cases).

The out-of-bag data can serve as a test set for the tree grown on the bootstrap sample.

The out-of-bag Error Estimate

Think of a *single case* in the training set.

It will be out-of-bag in about one-third of the trees.

Each time it is out of bag, pass it down the tree and get a predicted class.

The RF prediction is the class that is chosen the most often.

For each case, the RF prediction is either correct or incorrect.

- Average over the cases within each class to get a *classwise* out-of-bag error rate.
- Average over all cases to get an *overall* out-of-bag error rate.

The out-of-bag error Estimate

For example, suppose we fit 1000 trees, and a case is out-of-bag in 339 of them, of which:

283 say “class 1”

56 say “class 2”

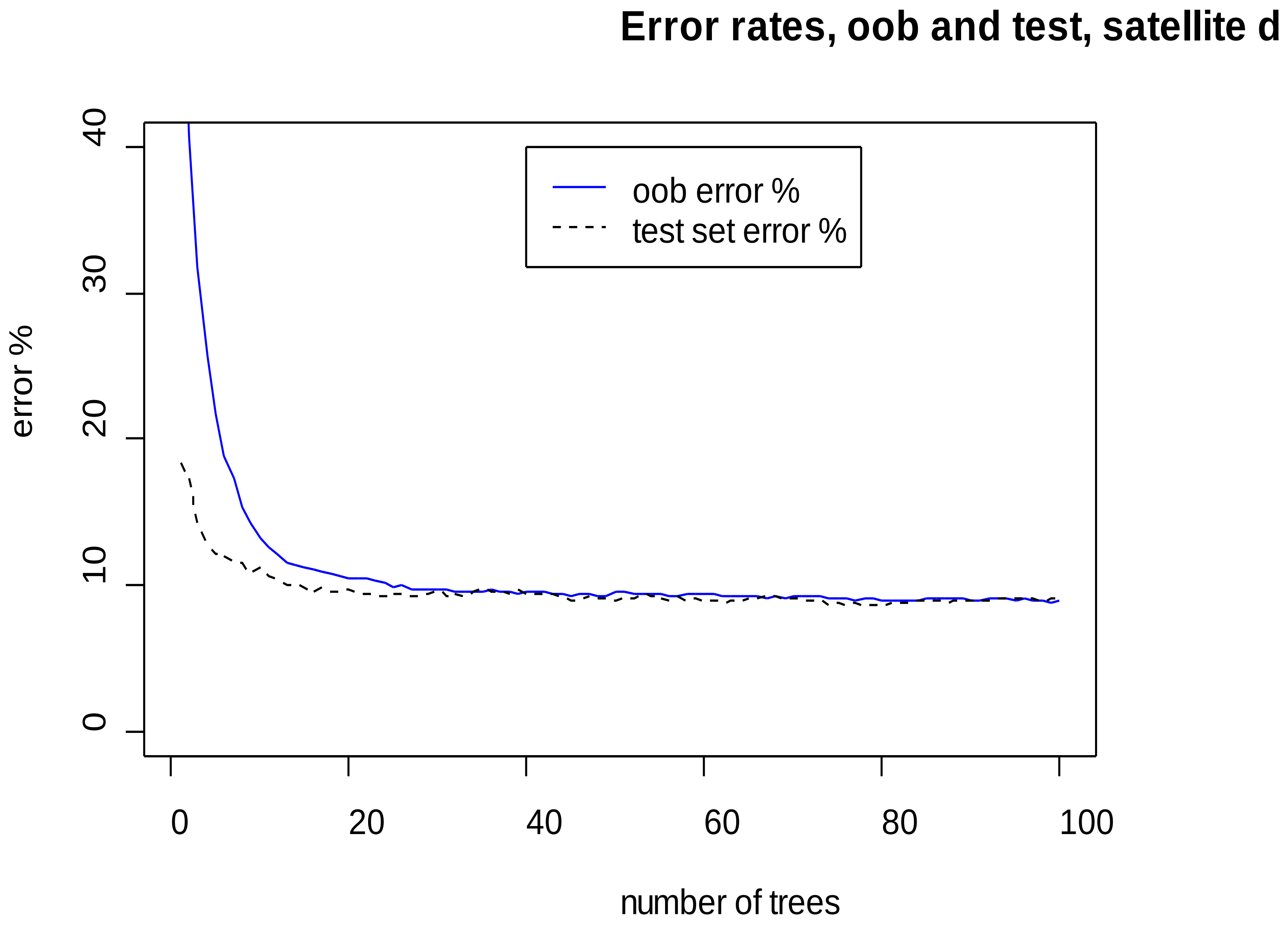
The RF predictor is class 1.

The out-of-bag error rate (%) is the percentage of the time that the RF predictor is correct.

Illustration – Satellite data

- Training set: 4435 cases.
- Test set: 2000 cases.
- 36 variables.
- 100 trees.

The out-of-bag error rate is larger at the beginning because there are not enough trees for good predictions.



Using out-of-bag Data to Choose **m**

The out-of-bag error rate is used to select **m**.

Here's how:

1. Start with **m** = √**M**.
2. Run a few trees, recording the out-of-bag error rate.
3. Increase **m**, decrease **m**, until you are reasonably confident you've found a value with minimum out-of-bag error rate.

Out-of-bag Data and Variable Importance

Consider a single tree (fit to a bootstrap sample).

1. Use the tree to predict the class of each out-of-bag case.

2. Randomly permute the values of the variable of interest in all the out-of-bag cases, and use the tree to predict the class for these perturbed out-of-bag cases.

The variable importance is the increase in the misclassification rate between steps 1 and 2, averaged over all trees in the forest.

P-values for Variable Importance

For each tree, we get an increase in the misclassification rate when the variable of interest is perturbed in the out-of-bag data.

The average increase, over the trees in the forest, is the variable importance.

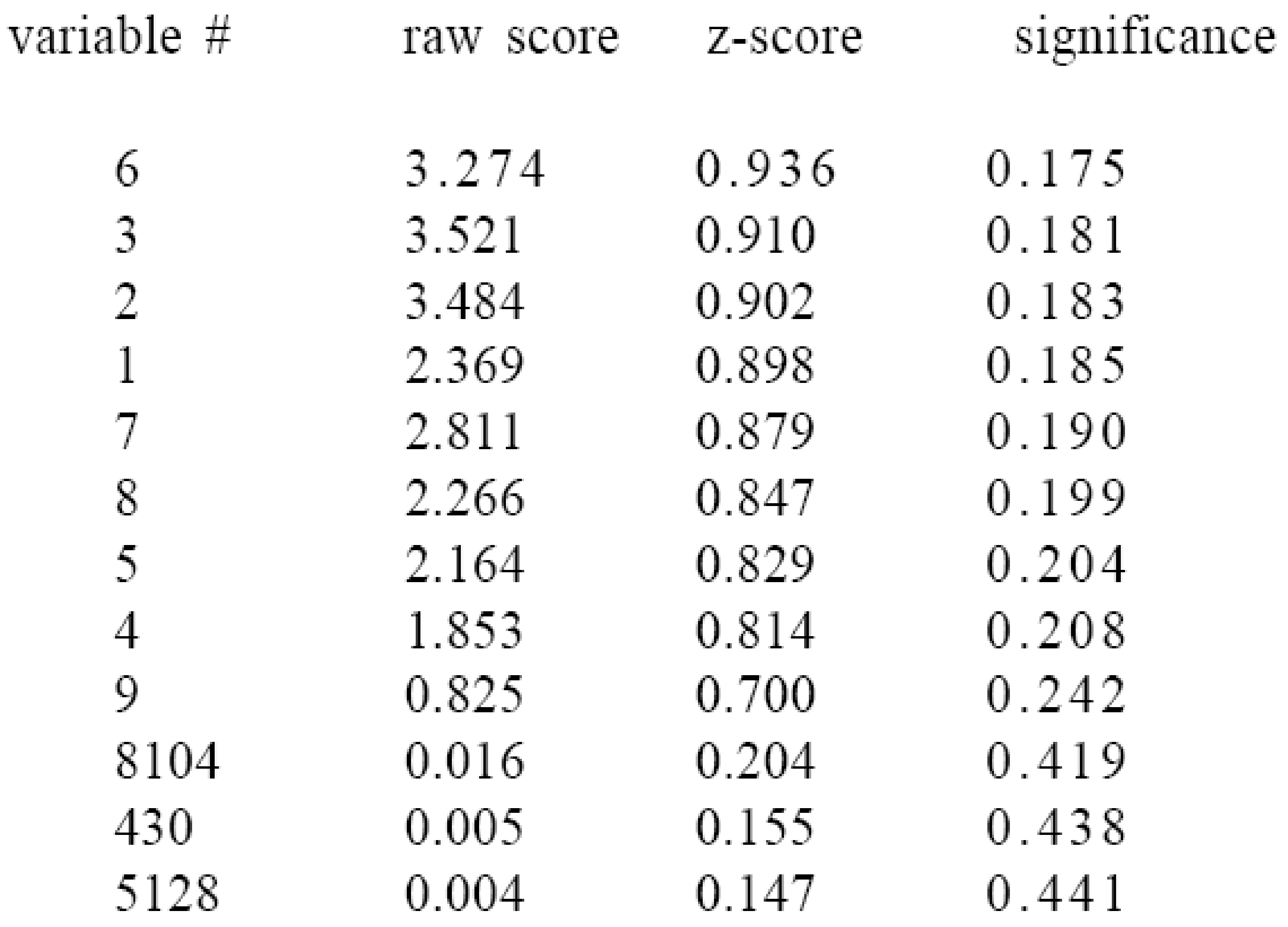
From the SD we can find a standard error, and assuming normality, a z-score and a P-value.

Illustration – Breast Cancer data

- 699 cases, 9 variables, two classes
- Initial out-of-bag error rate is 3.3%

Added 10,000 independent standard normals to each case, making 10,009 variables.

The twelve most important variables, chosen by RF:

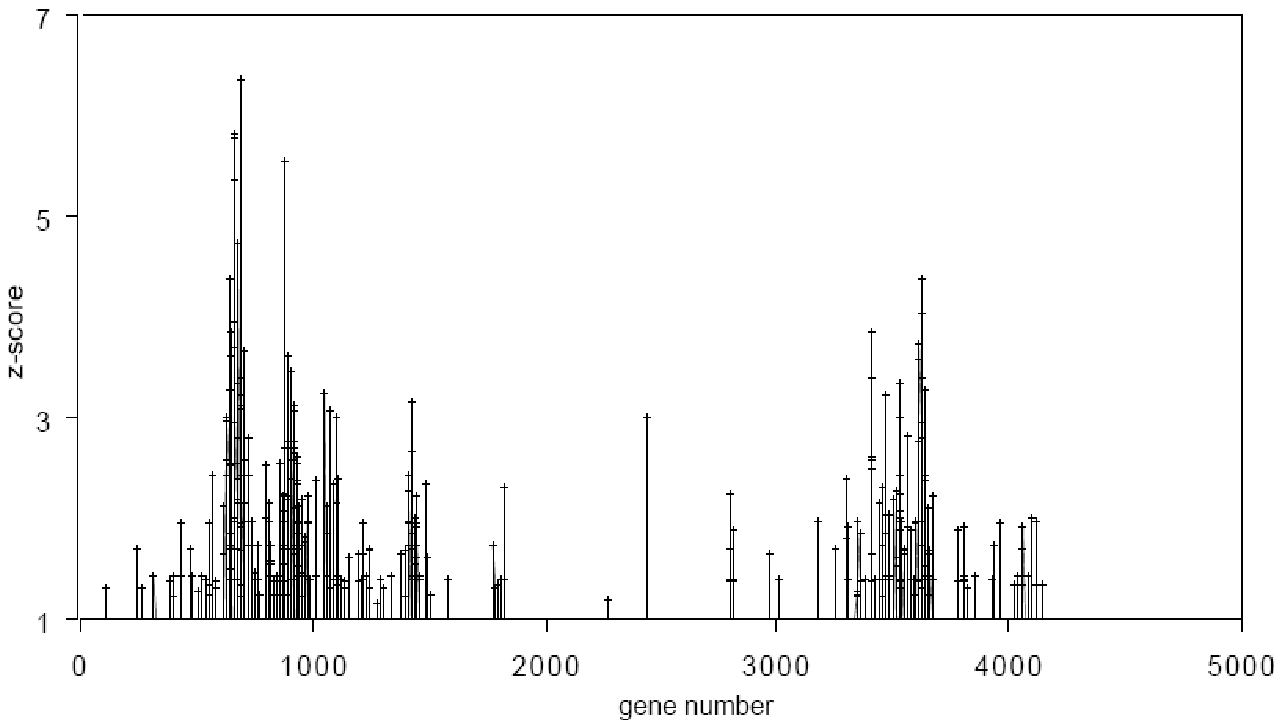


variable #	raw score	z-score	significance
6	3.274	0.936	0.175
3	3.521	0.910	0.181
2	3.484	0.902	0.183
1	2.369	0.898	0.185
7	2.811	0.879	0.190
8	2.266	0.847	0.199
5	2.164	0.829	0.204
4	1.853	0.814	0.208
9	0.825	0.700	0.242
8104	0.016	0.204	0.419
430	0.005	0.155	0.438
5128	0.004	0.147	0.441

Illustration – Microarray data (lymphoma)

- 81 cases, 4682 variables, three classes.
- Initial out-of-bag error rate is 1.25% (one case).

Z-scores for 1000 trees:



Don’t take our word for it!

2003 NIPS competition on feature selection in high-dimensional data (thousands of variables):

- over 1600 entries from some of the most prominent people in machine learning
- top 2nd and 3rd entries used RF for feature selection.

Microarrays: March, 2005

Ramón Díaz-Uriarte, Sara Alvarez de Andrés

http://ligarto.org/rdiaz

Showed that RF gives good gene selection for microarray classification problems.

Two Natural Questions

1. *Why bootstrap?*

   Fitting our randomized trees to the training data is about as accurate, so why bootstrap?

   OUT-OF-BAG DATA

2. *Why trees?*

   There are many weak learners which could work, so why use trees?

   PROXIMITIES

Proximities

Pass all the data down all the trees in the forest.

Define the proximity between cases n and k as the average number of trees for which cases n and k occupy the same terminal node.

The proximities form an intrinsic similarity measure between pairs of cases.

Proximities (ctnd)

The proximities are not directly related to Euclidean distance. Instead, they are based on the way the trees deal with the data.

- Two cases that are far apart in Euclidean space might have high proximity because they stay together in all the trees.

- Two cases that are quite close together in Euclidean space might have relatively small proximity if they are near the classification boundary.

Proximities for Visualization

The values 1-prox(n,k) are squared distances in a high-dimensional Euclidean space.

They can be projected onto a low-dimensional space using metric multidimensional scaling.

Metric multidimensional scaling uses eigenvectors of a modified version of the proximity matrix to get scaling coordinates.

Illustration – Microarray data

- 81 cases, 4682 variables, three classes.
- Initial out-of-bag error rate is 1.25% (one case).

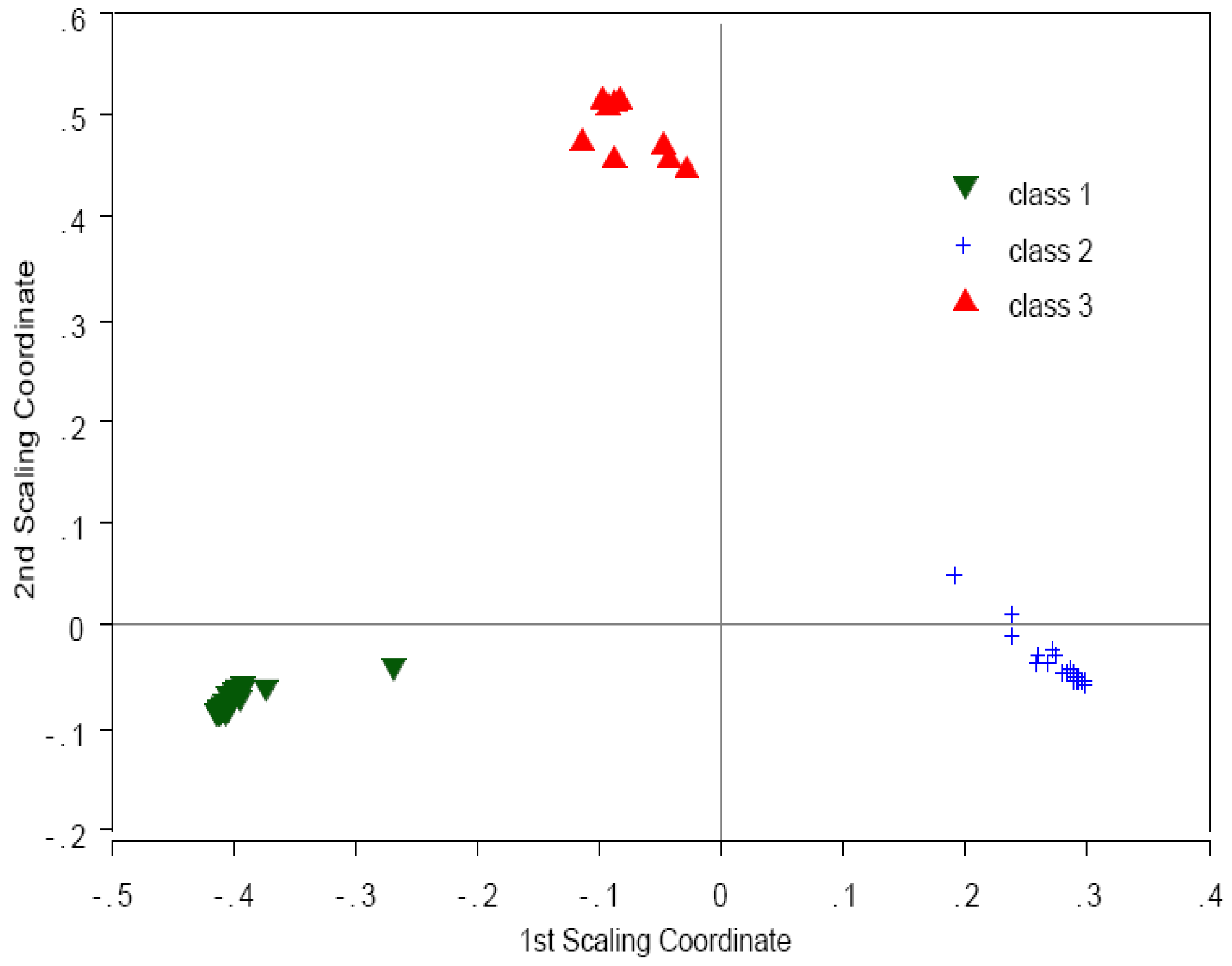
Other methods get similar results, but more is needed for the science:

1. What do the data look like? How do they cluster?
2. Which genes are active in discrimination?
3. What multivariate levels of gene expressions discriminate between classes?

The second question can be answered using variable importance. Let’s take a look at 1 and 3.

Picturing the Microarray Data

The image below is a plot of the second scaling coordinate versus the first scaling coordinate.



Replacing Missing Values

Fast way: replace missing values for a given variable using the median of the non-missing values (or the most frequent, if categorical)

Better way (using proximities):

1. Start with the fast way.
2. Get proximities.
3. Replace missing values in case **n** by a weighted average of non-missing values, with weights proportional to the proximity between case **n** and the cases with the non-missing values.

Repeat steps 2 and 3 a few times (5 or 6).

Identifying Unusual Cases

Outliers can be found using proximities.

An outlier is a case whose proximities to all other cases are small.

More precisely, we can compute a measure of outlyingness for each case in the training sample

$$\text{outlyingness of case } n = 1/\sum_{k \neq n}(\text{prox}(n,k)^2)$$

If the measure is unusually large (greater than 10), the case should be carefully inspected.

Outlyingness

This concept differs from the usual one because we use the random forest proximities instead of Euclidean distance based on the original variables.

Outliers in the Euclidean sense may not be outliers in the proximity sense and vice versa.

Illustration – Microarray data

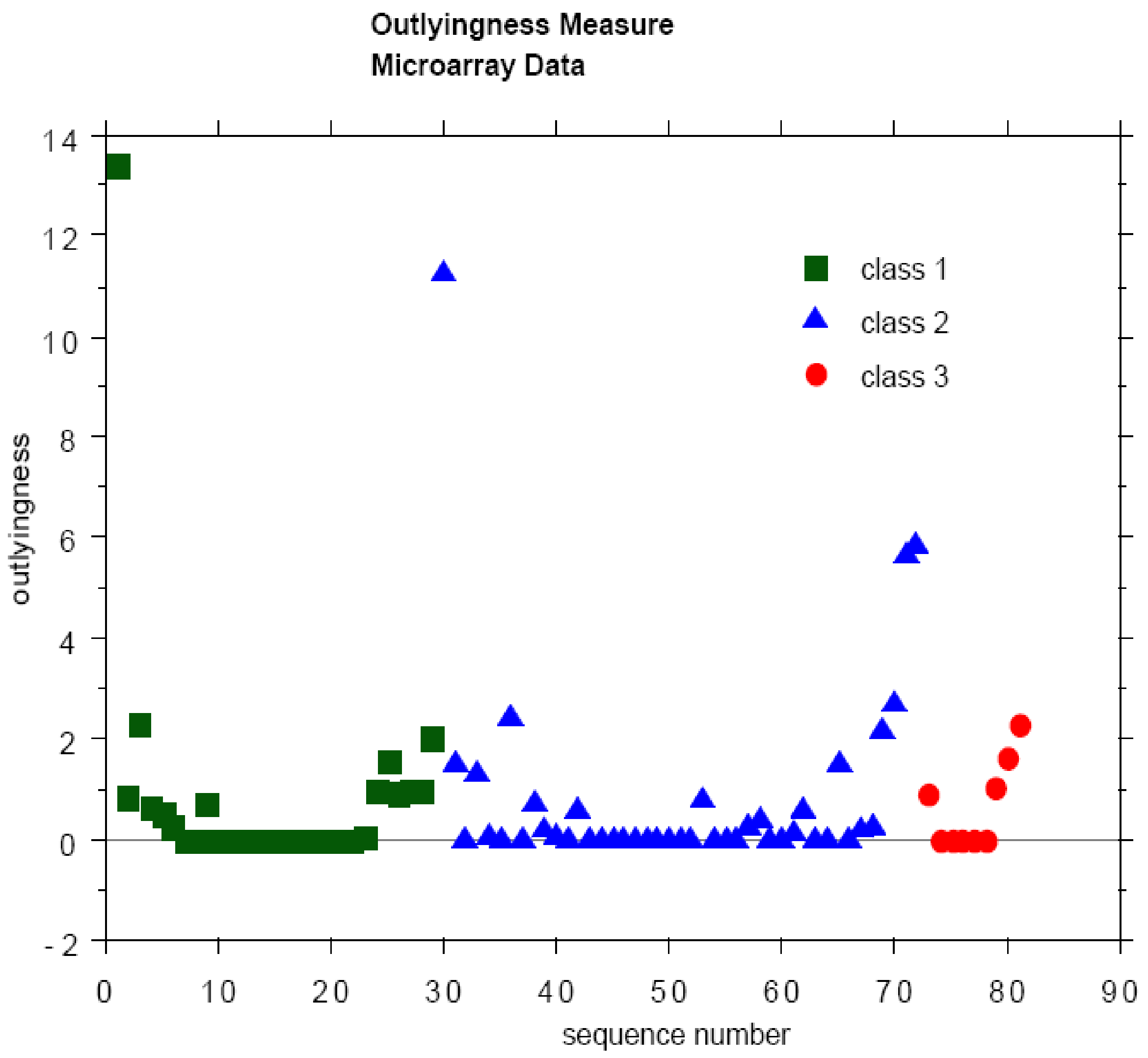
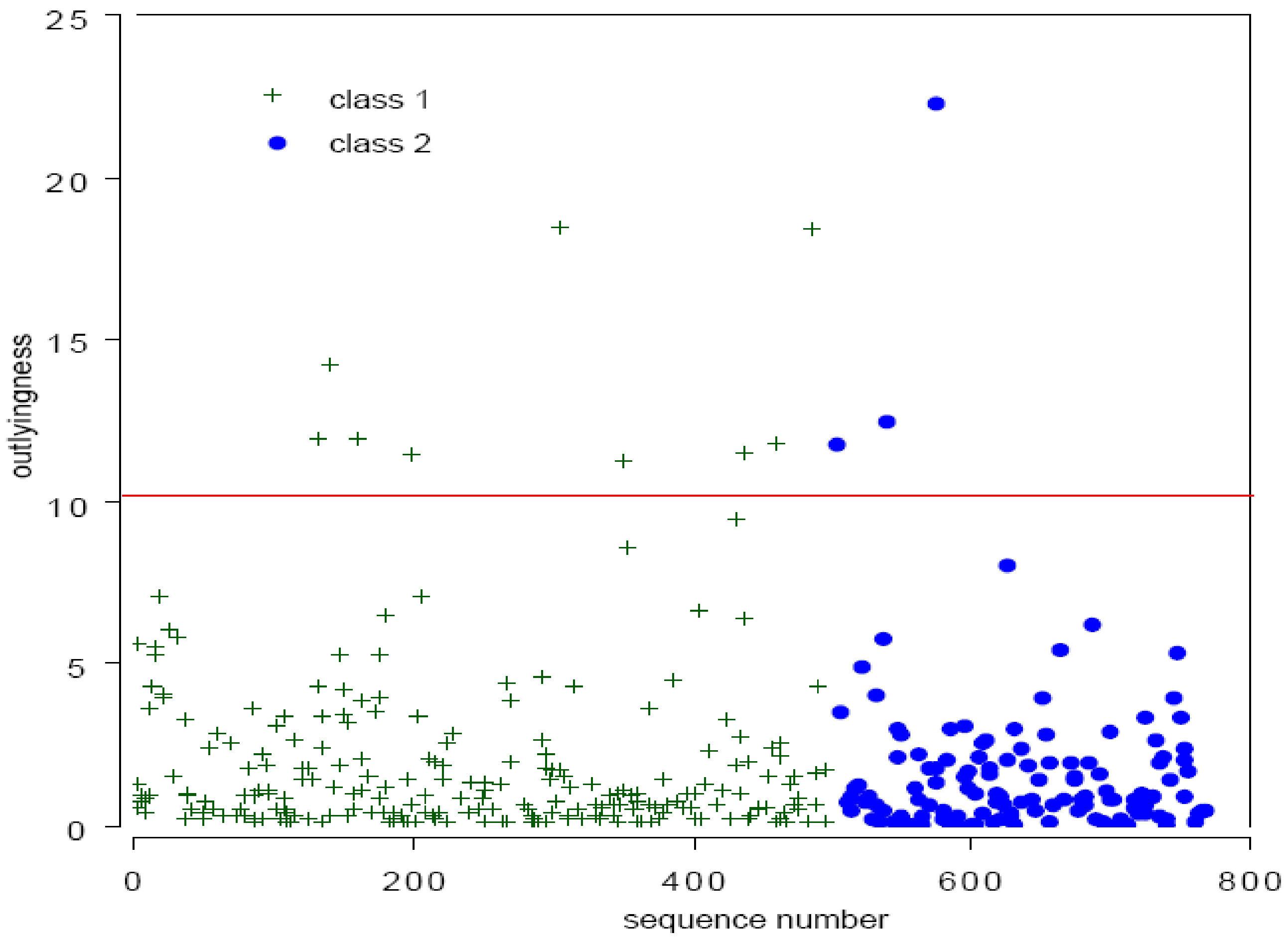


Illustration – Pima Indian Data

- 768 cases, 8 variables, 2 classes
- Machine learning example, known outliers



Mislabeled Data as Outliers

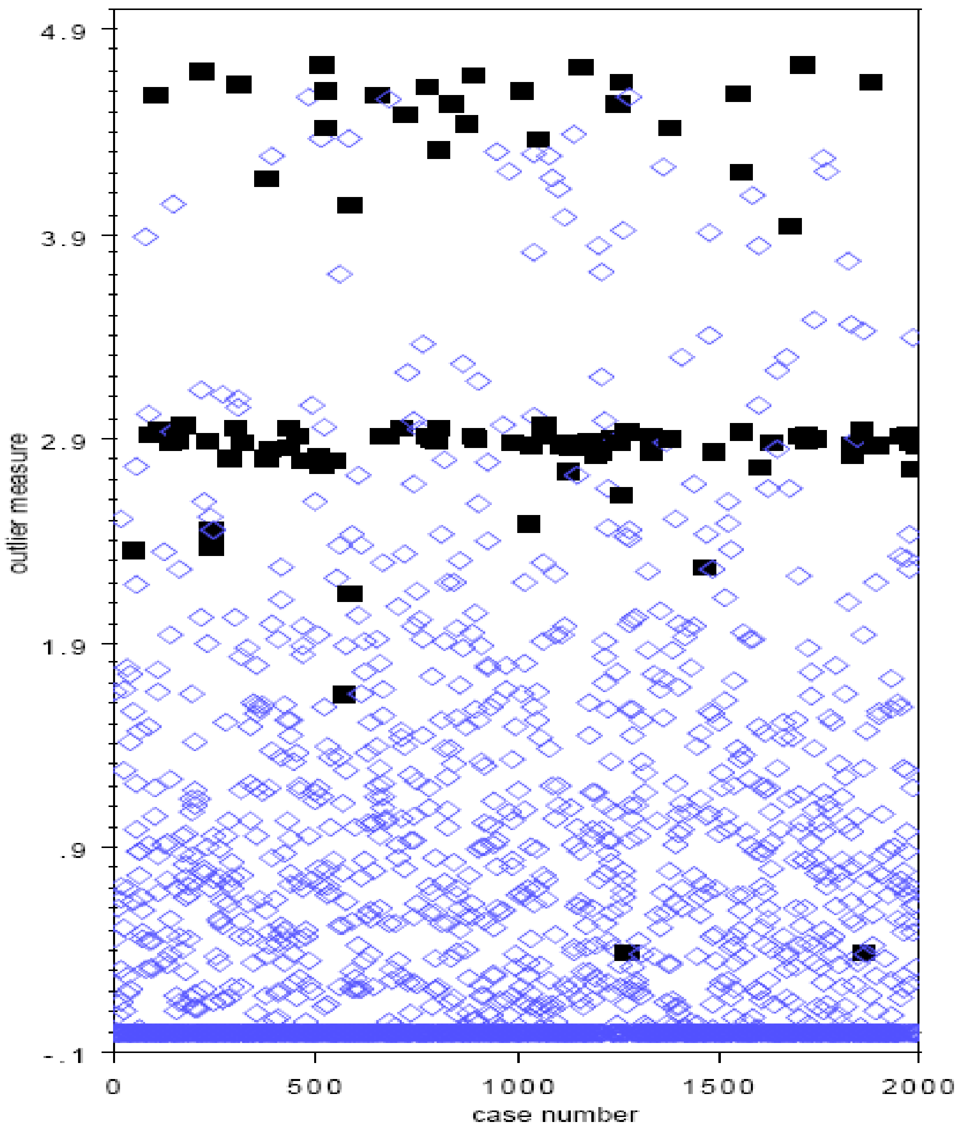
Often the instances in the training dataset are labeled “by hand”. The labels can be ambiguous or downright incorrect.

Consider the DNA data

- 2000 cases, 60 variables, 3 classes
- All variables are 4-valued categoricals (1,2,3,4)

Change 100 labels at random.

Illustration – DNA data



Prototypes

Prototypes are a way of getting a picture of how the variables relate to the classification.

For each class, we seek a small number (2 to 4?) of prototypes to represent the class.

A small, homogeneous class may be summarized by a single prototype, while diverse classes, or classes with clumping will require more prototypes.

Here’s how we get them…

Finding Prototypes

Think of class *j*.

For each case, find its K “nearest” neighbors (using proximities).

Select the case that has the largest weighted number of class *j* nearest neighbors.

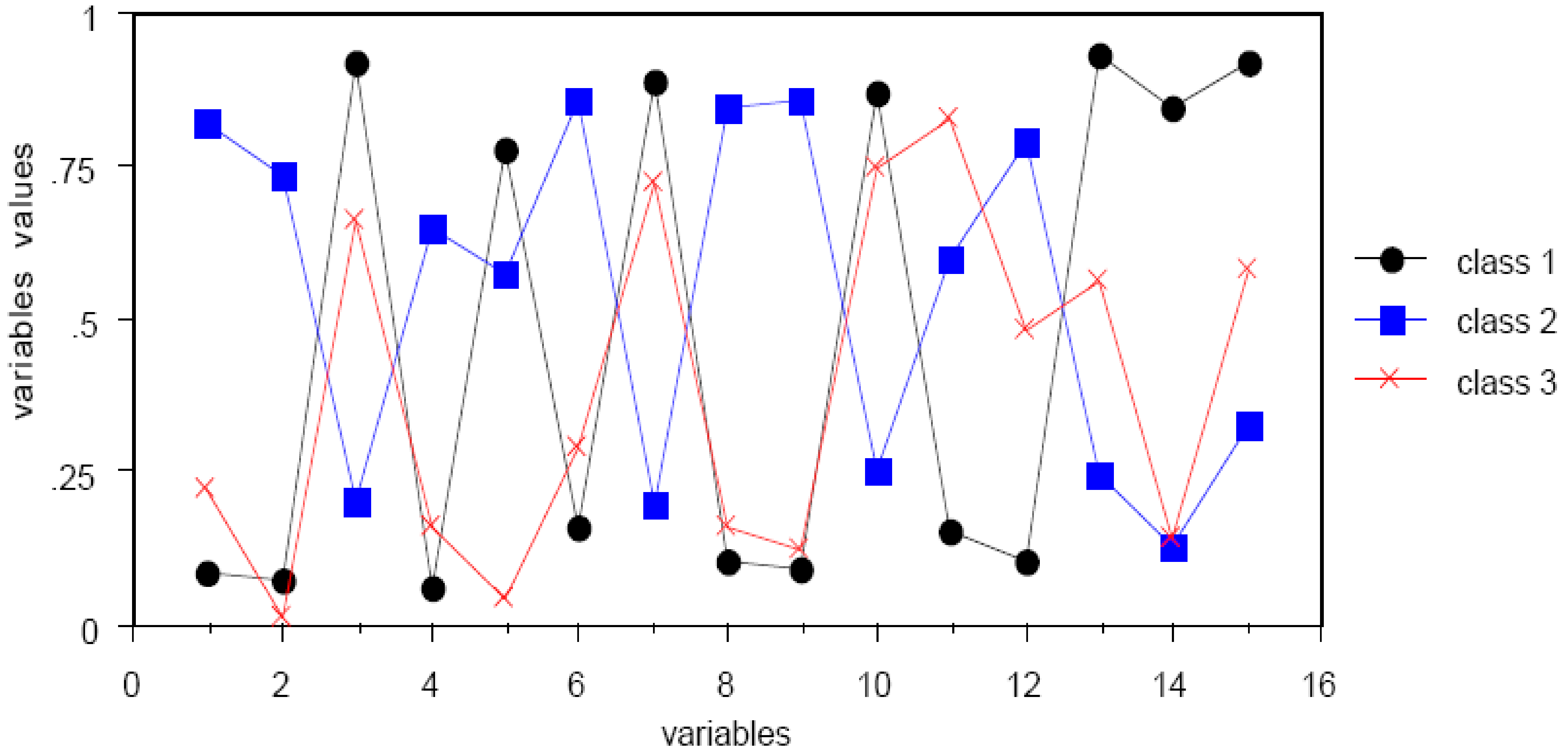
The median of these neighbors is the prototype, and the first and third quartiles give an idea of stability.

Now look for another case that is not one of the previously-chosen neighbors and select the one that has the largest weighted number of class *j* nearest neighbors.

Repeat.

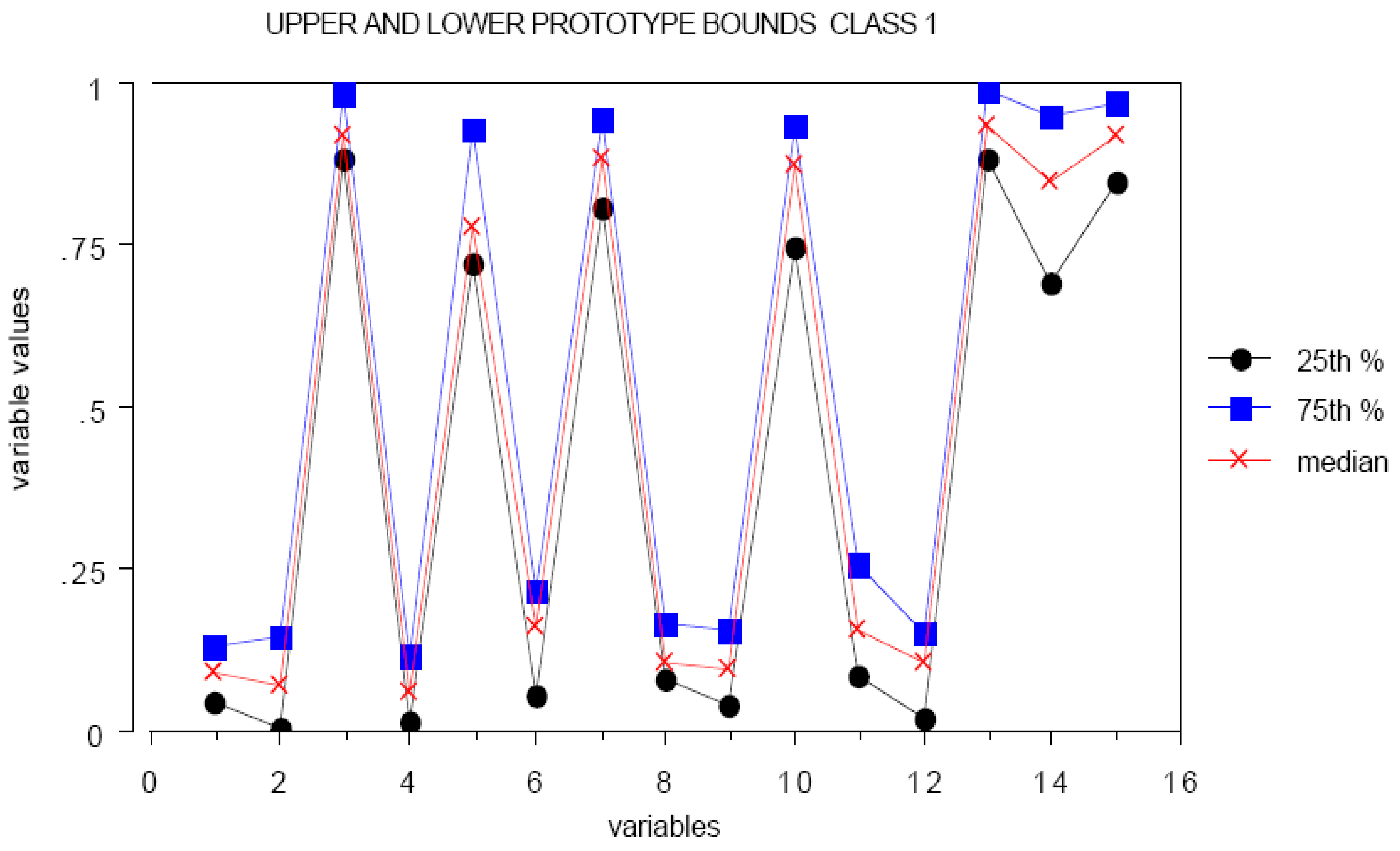
Illustration – Microarray data

There are three classes. Look at a single prototype for each class.



Prototype Variability

Now look at the first and third quartiles for class 1



Learning from Unbalanced Data

Increasingly often, data sets are occurring where the class of interest has a population that is a small fraction of the total population.

Examples

- In document classification, the number of relevant documents may be 1 or 2% of the total number.
- In drug discovery, the number of active drugs may be a small fraction of all the compounds studied.

Unbalanced Data

For such unbalanced data, a classifier can achieve great accuracy by classifying almost all cases into the majority class!

Illustration – Satellite data

4435 cases, 36 variables, 6 classes.

Class 4 has 415 cases. Call this class 2 and combine the rest of the cases into class 1.

Run 1: unbalanced

Overall error rate	5.8%
Class 1 error rate	51.0%
Class 2 error rate	1.2%

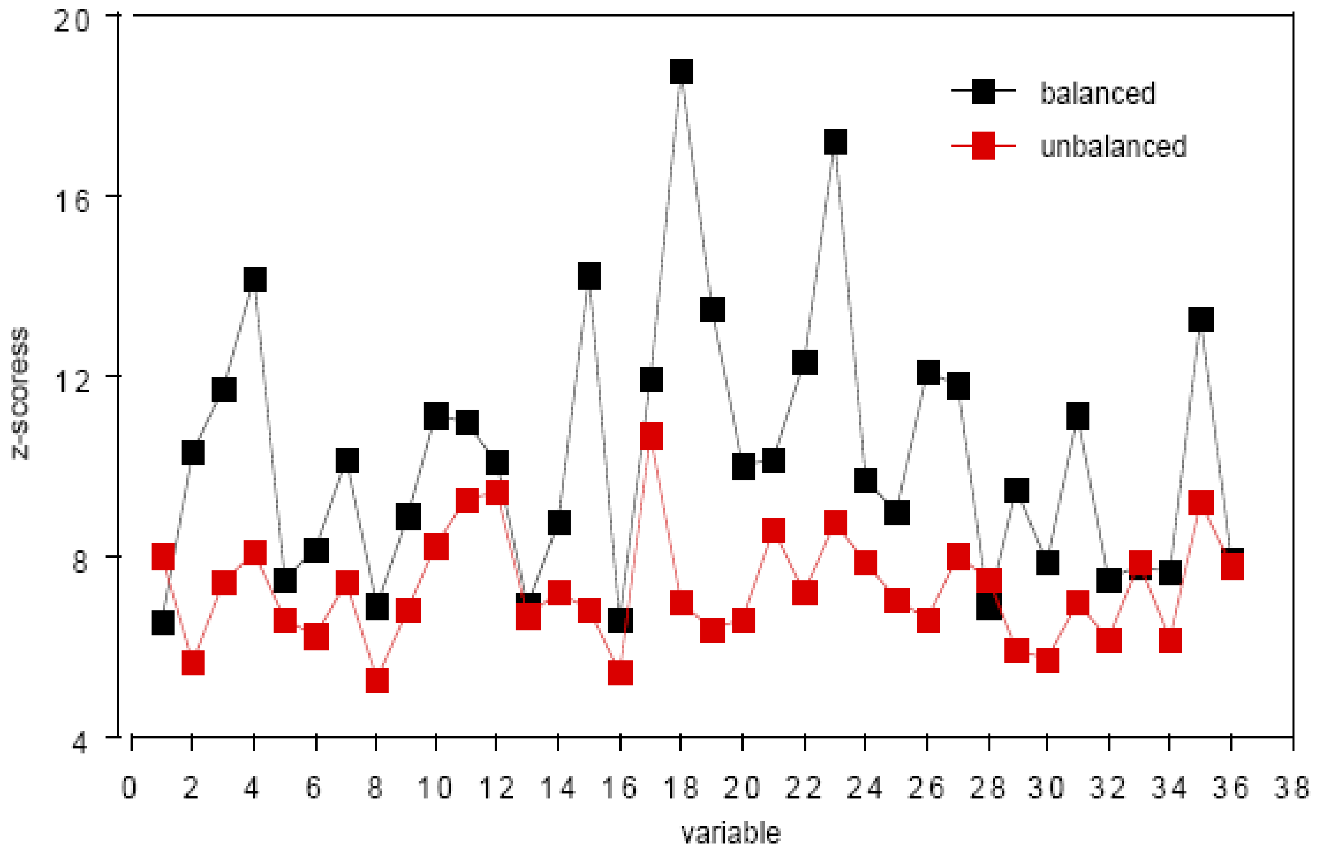
Run 2: Try to equalize the error rate for the two classes

Overall error rate	12.9%
Class 1 error rate	13.1%
Class 2 error rate	11.3%

8:1 weight on class 2

Variable Importances

Here is a graph of the z-scores for the two runs



Variable Importances

In the unbalanced run, the variable importances tend to be more equal (it’s ignoring class 2).

In the balanced case, a small number stand out.

Variable 18 becomes important, as does variable 23.

How did we do the balancing?

Balancing the Error Rate

The out-of-bag error rate is used to balance the error rates when the class sizes are unbalanced or different error rates are desired in the different classes. Here’s how:

1. Start by guessing the class weights (e.g. set them all equal or weight them in inverse proportion to size).
2. Run a few trees, recording the out-of-bag error rate for each class.
3. Increase the weight on classes with high error rates until you get the desired misclassification rates in the groups.

Part 3

Visualization and Case Studies

Random Forests – a personal history

1988-1993

- Archetypal analysis (Leo Breiman)
- Global optimization (Leo Breiman, Bill Baritompa)
- Finite mixture models (Mike Windham, Olga Cordero-Brana)
- Minimum Hellinger distance estimation (Olga Cordero-Brana, Richard Cutler)

1993-1999

- Archetypal analysis of dynamical systems (Emily Stone)
- Neural networks and machine learning
- Non-negative garrote (Leo Breiman)
- PERT (Guohua Zhao)

Current Projects (since 2000)

- Random forests (Leo Breiman)
  - Fortran Code
  - Java visualization
- Applications
  - Autism (Dennis Odell and Ron Torres)
  - Microarrays (Bart Weimer, Yi Xie)
  - Dementia (Chris Corcoran and Leslie Toone)

Random Forests Software

- Free, open-source code (fortran)
  www.stat.berkeley.edu/users/breiman/RandomForests
- Commercial version (academic discounts)
  www.salford-systems.com
- R interface, independent development (Andy Liaw and Matthew Wiener)
- Free, open-source java visualization tool
  www.stat.berkeley.edu/users/breiman/RandomForests

Open Source Code

User

- Free, can read and modify code.
- Fast way to get productive results (applications, new research, etc).

Developer

- Users give valuable insight (inspire developments).
- Fast and easy way to distribute.
- Users do comparisons (sink or swim).

- Hard when software “comes of age” (bravo Salford).
- Piracy.

Visualization for Random Forests

Random forests give

1. *proximities*
2. *variable importance*
   - *casewise*
   - *classwise*
   - *overall*
3. *prototypes*

HOW CAN WE USE THESE EFFICIENTLY?

Proximities and scaling

Proximity of two items is the proportion of the time that they end up in the same node when they are simultaneously out-of-bag.

Proximities $\xrightarrow{\text{MDS}}$ scaling variables

Current fortran code only does metric MDS, but could use other methods.

Visualizing proximities

- at-a-glance information about which classes are close, which classes differ
- find clusters within classes
- find easy/hard/unusual cases

With a good tool we can also

- identify characteristics of unusual points
- see which variables are locally important
- see how clusters or unusual points differ

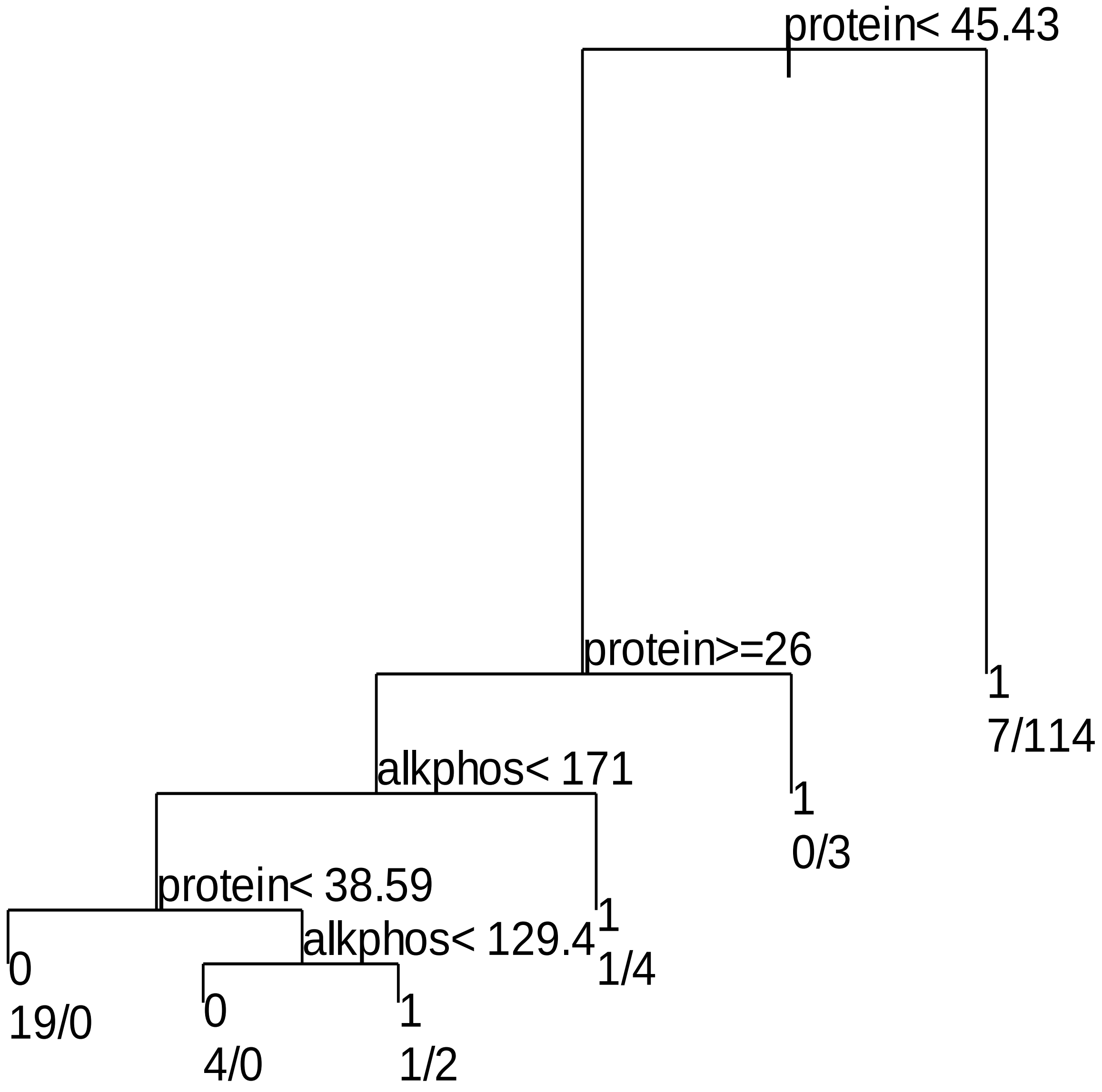
A Closer Look at Variable Importance

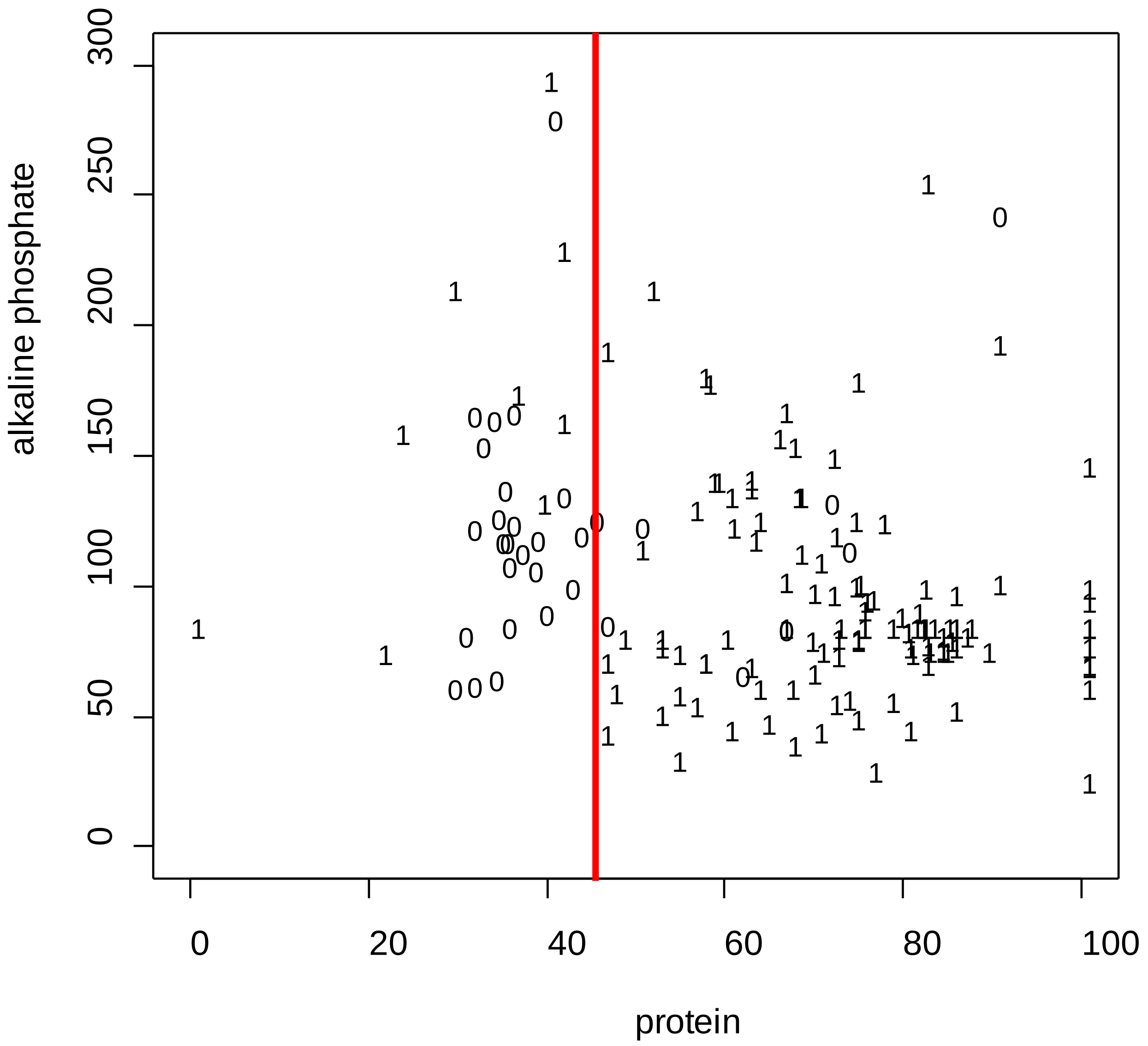
We usually think about variable importance as an overall measure. In part, this is probably because we fit models with global structure (linear regression, logistic regression).

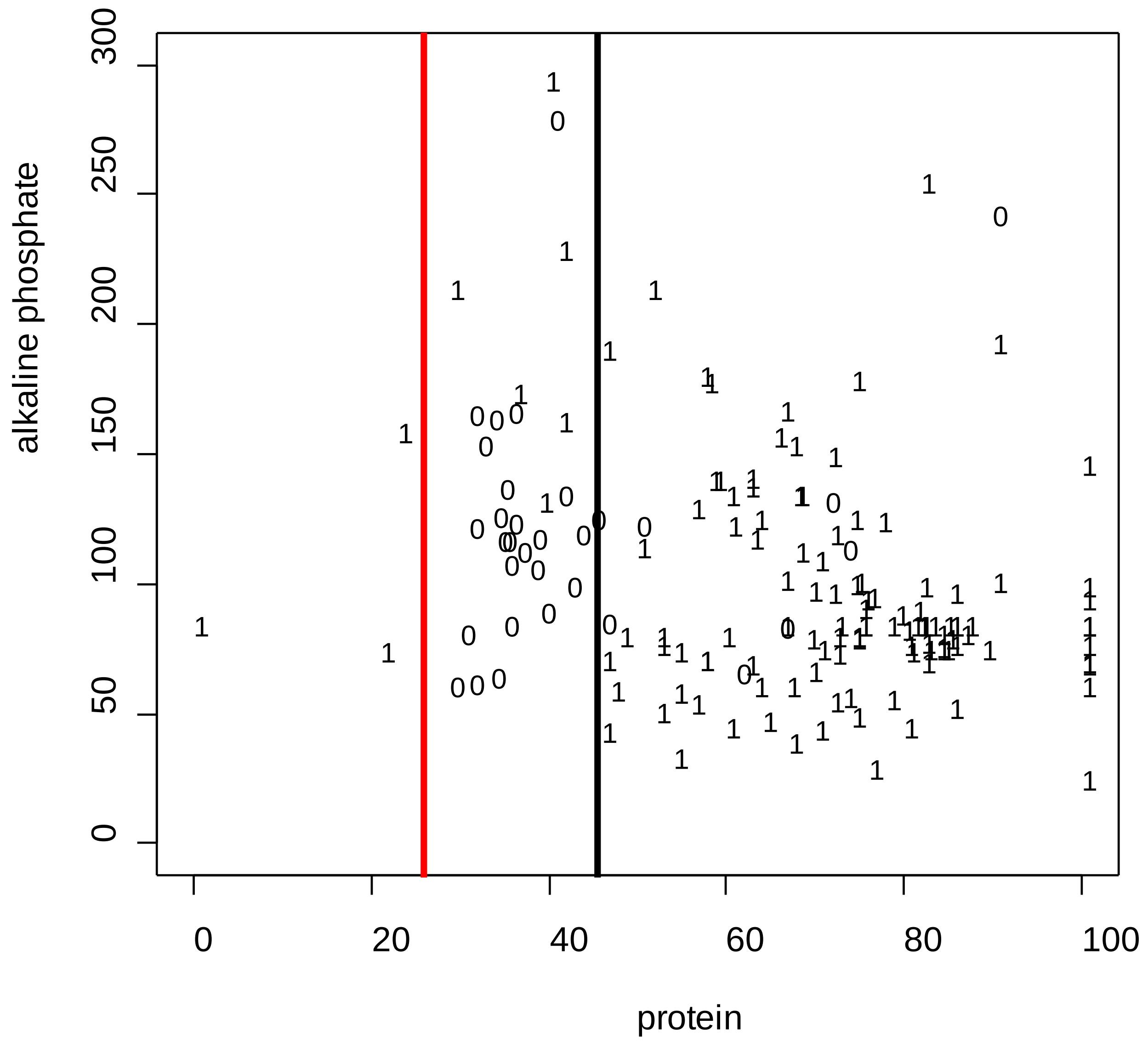
In CART, variable importance is local.

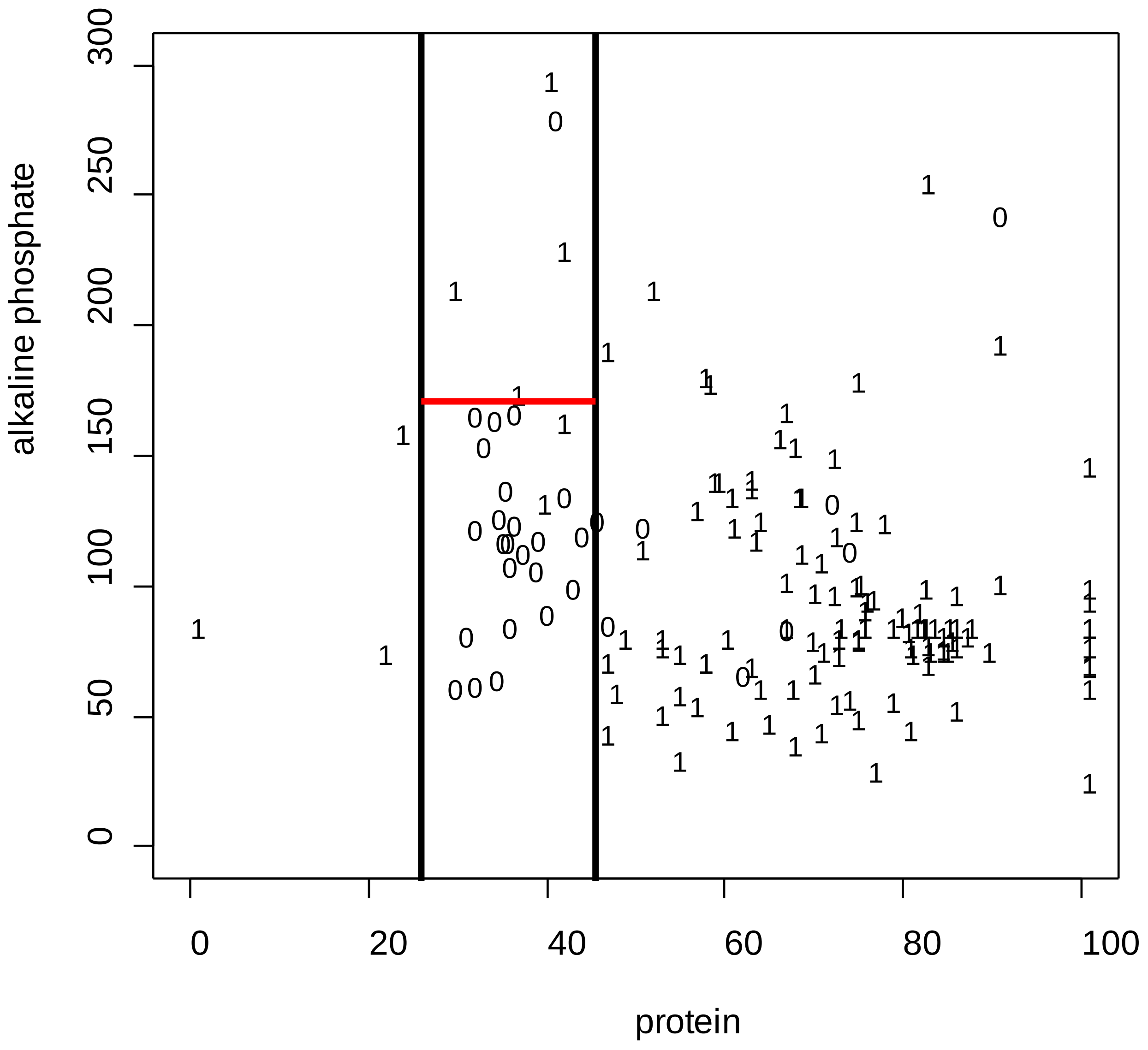
Illustration - Hepatitis

protein and alkaline phosphate

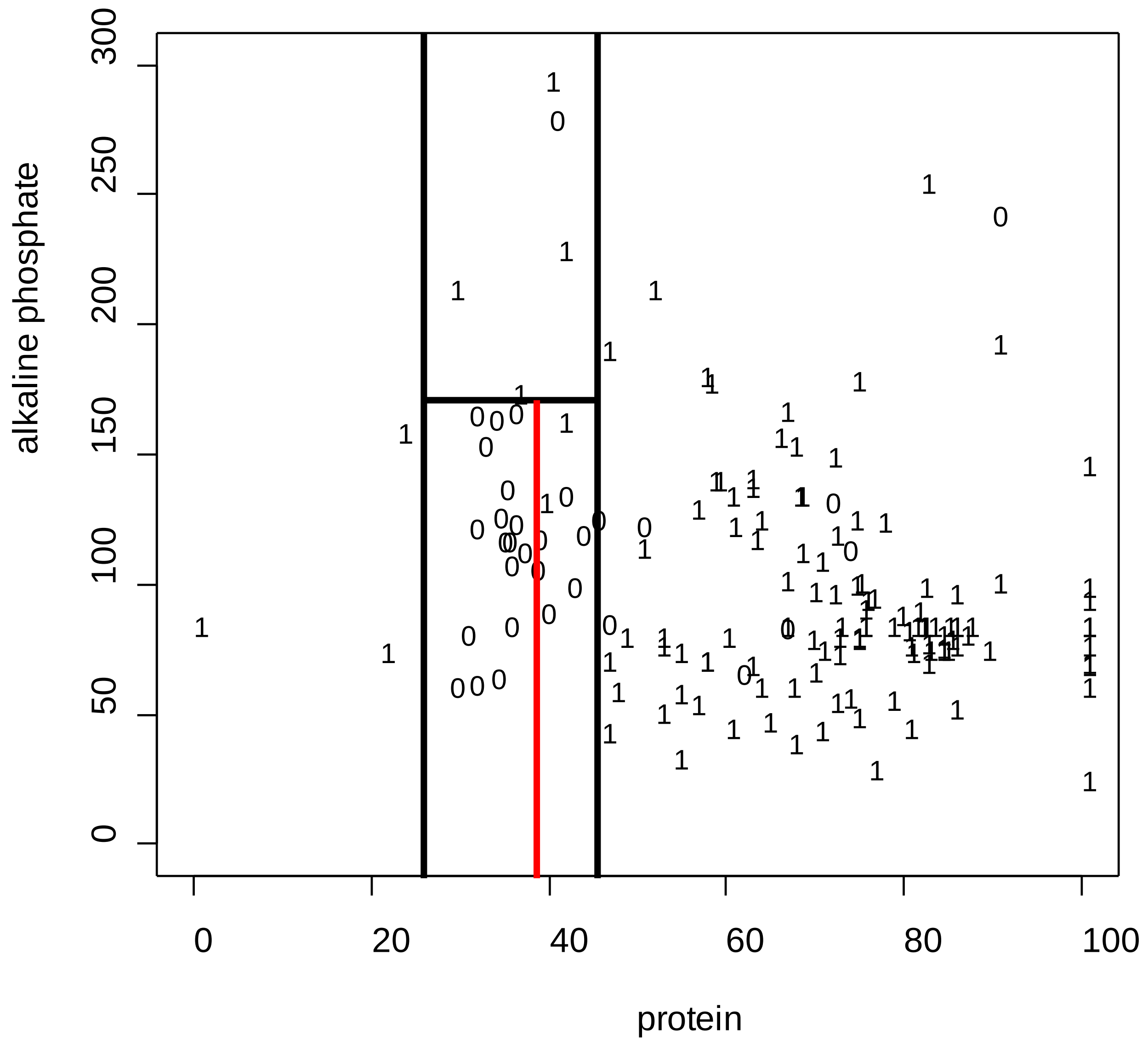


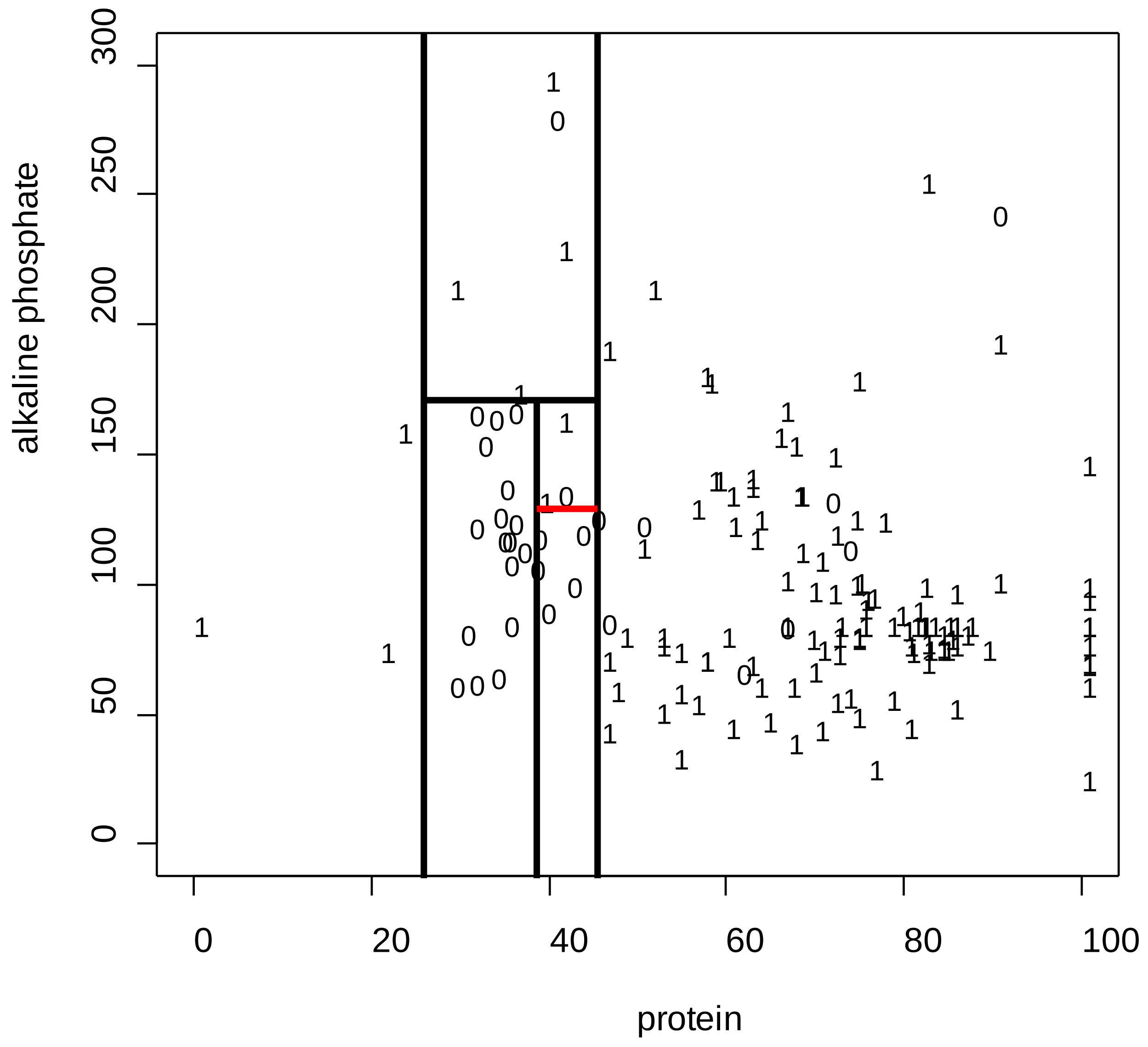

alkaline phosphate
protein
0
50
100
150
200
250
300
0
20
40
60
80
100


alkaline phosphate
protein
0
50
100
150
200
250
300
0
20
40
60
80
100



alkaline phosphate
0
50
100
150
200
250
300
0
20
40
60
80
100
protein

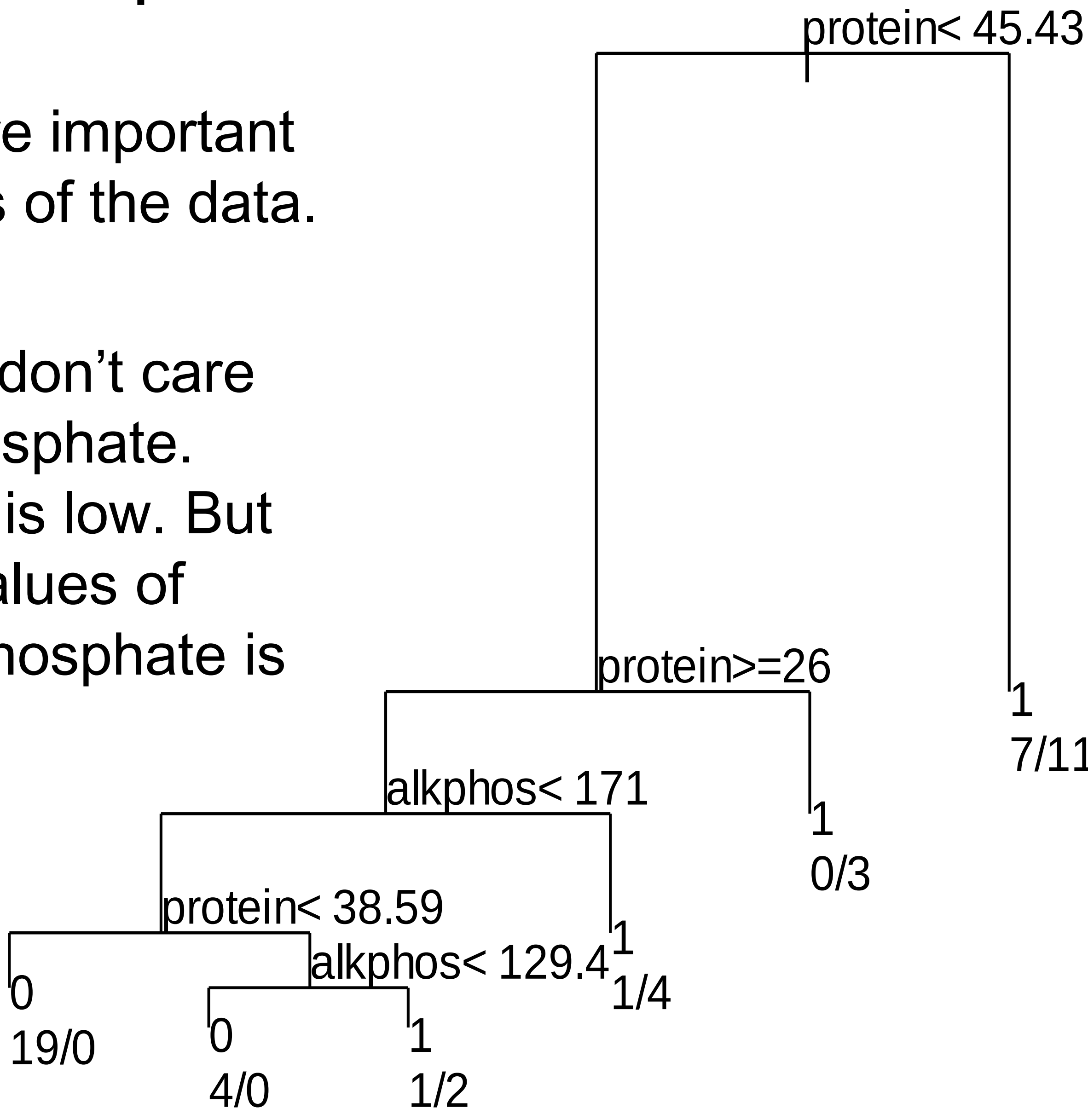

alkaline phosphate
protein
0
50
100
150
200
250
300
0
20
40
60
80
100


alkaline phosphate
protein
0
50
100
150
200
250
300
0
20
40
60
80
100

LOCAL Variable Importance

Different variables are important in different regions of the data.

If protein is high, we don't care about alkaline phosphate. Similarly if protein is low. But for intermediate values of protein, alkaline phosphate is important.



Local Variable Importance

For each tree, look at the out-of-bag data:

- randomly permute the values of variable *j*
- pass these perturbed data down the tree, save the classes.

For case *i* and variable *j* find

$$\begin{array}{c} \text{error rate with} \\ \text{variable } j \text{ permuted} \end{array} - \begin{array}{c} \text{error rate with} \\ \text{no permutation} \end{array}$$

where the error rates are taken over all trees for which case *i* is out-of-bag.

Variable importance for a class 2 case

TREE	No permutation	Permute variable 1	…	Permute variable m
1	2	2	…	1
3	2	2	…	2
4	1	1	…	1
9	2	2	…	1
…	…	…	…	…
992	2	2	…	2
% Error	10%	11%	…	35%

RAFT
RAndom Forests graphics Tool

- java-based, stand-alone application
- uses output files from the fortran code
- download RAFT from www.stat.berkeley.edu/users/breiman/RandomForests/cc_graphics.htm

Java Development:

Raft uses VisAD

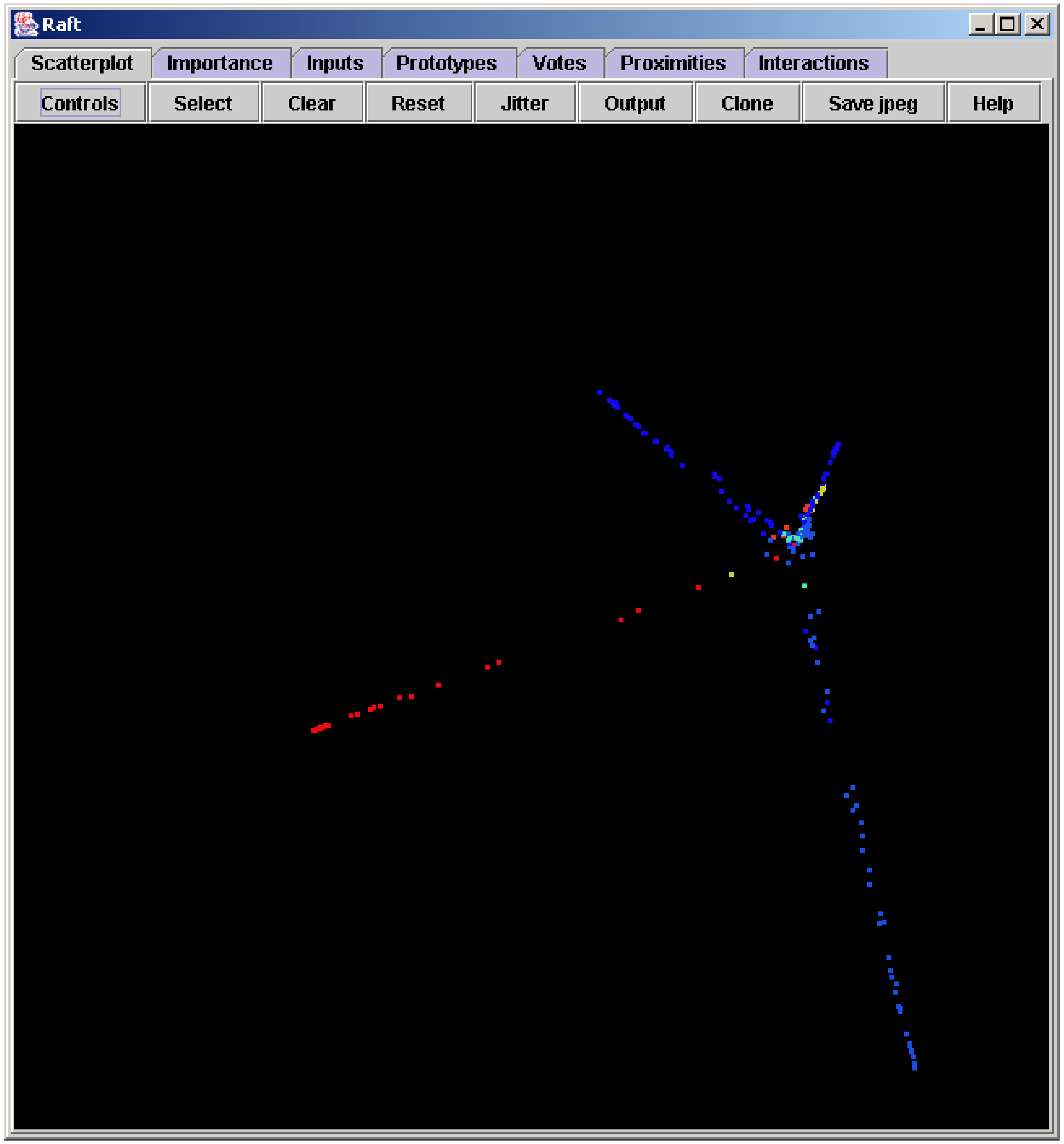
www.ssec.wisc.edu/~billh/visad.html

and ImageJ

http://rsb.info.nih.gov/ij/

These are both open-source projects with great mailing lists and helpful developers.

RAFT Main Window



- Scatterplot
- Importance
- Inputs
- Prototypes
- Votes
- Proximities
- Interactions

General Controls

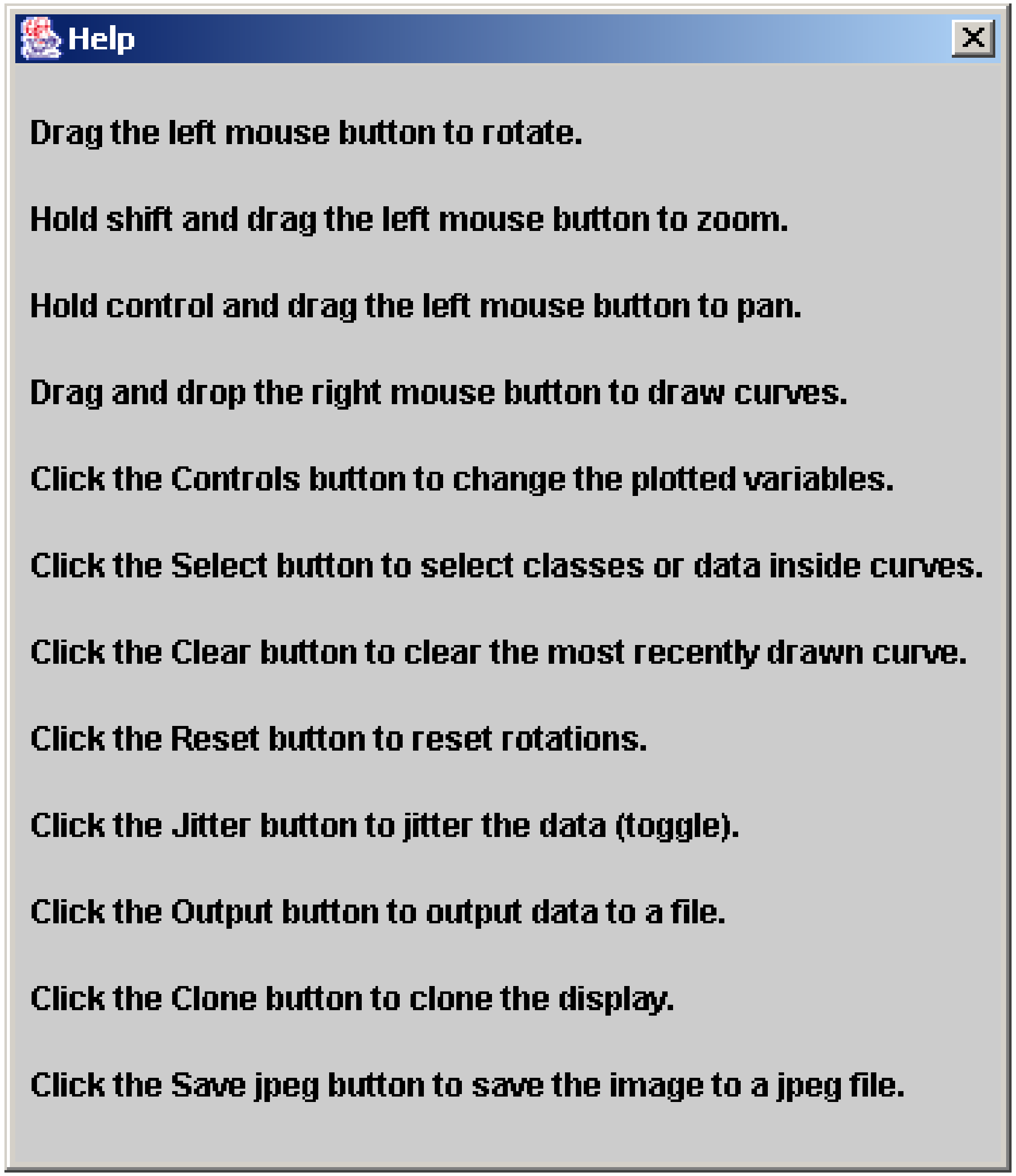
Left mouse rotates

Shift-left zooms

Control-left pans

Right mouse draws (drag and drop)

Help



Help

Drag the left mouse button to rotate.

Hold shift and drag the left mouse button to zoom.

Hold control and drag the left mouse button to pan.

Drag and drop the right mouse button to draw curves.

Click the Controls button to change the plotted variables.

Click the Select button to select classes or data inside curves.

Click the Clear button to clear the most recently drawn curve.

Click the Reset button to reset rotations.

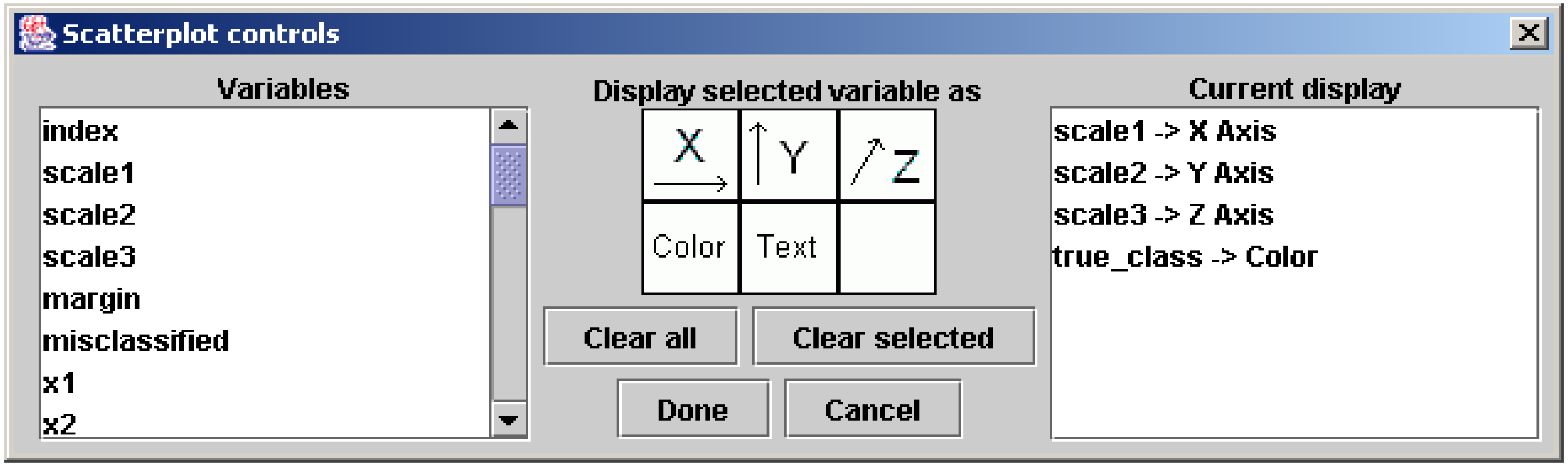
Click the Jitter button to jitter the data (toggle).

Click the Output button to output data to a file.

Click the Clone button to clone the display.

Click the Save jpeg button to save the image to a jpeg file.

Scatterplot Controls



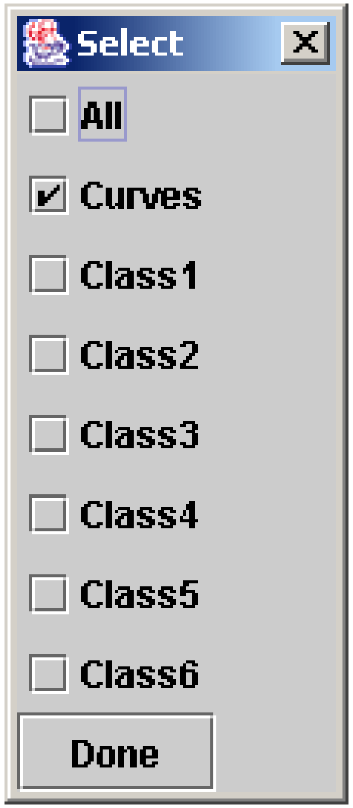
scale1 -> X Axis
scale2 -> Y Axis
scale3 -> Z Axis
true_class -> Color

Can change the x, y, z variables, color and text.

Select a variable on the left panel, click button in middle panel.

Right panel tells you the current choices.

Scatterplot Selection Tool

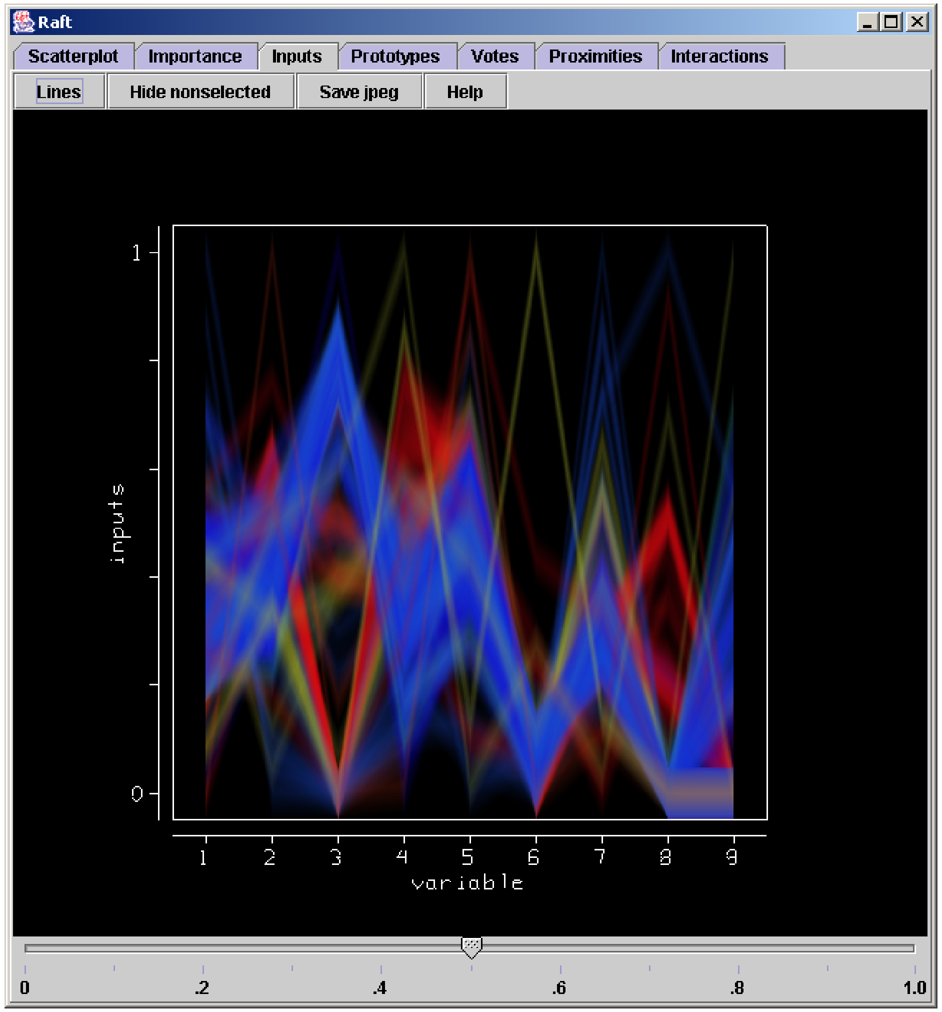


- Select one or more classes.
- Hand-select cases inside curves.

Selected cases will stay bright, others gray.

Illustration – Glass data

- 214 cases, 9 variables, 6 classes



Part 4

Unsupervised Learning
Regression
Survival Analysis
Case Studies

Unsupervised Learning

No class labels.

- Most common approach: try to “cluster” the data to find some “structure”.
  - Ambiguous, ill-defined.
  - Many methods rely on “white space” between “clusters” and are sensitive to outliers and noise.

- However, it should be possible to determine structure, and to discern the ways in which it differs from noise.

Consider the following approach…

The Unsupervised Learning Trick

Label the “real” data as class 1.

Construct a synthetic class, to be labeled class 2. The class 2 cases are constructed by randomly sampling each variable **independently** from the original data.

That is, for each synthetic case (1,…,**N**):

- Randomly select the value for variable 1 from all the observed values of variable 1.
- Independently, randomly select the value for variable 2 from all the observed values of variable 2.
- Similarly for variables 3,…,**M**.

The Synthetic Second Class

The synthetic second class has the same marginal distributions as the “real” data, but we have destroyed all the dependencies between the variables.

Now we have a 2-class classification problem.

Run random forests !

Unsupervised learning

If the out-of-bag misclassification rate is around 50%, random forests cannot distinguish between the classes.

This means that (according to random forests) the “real” data look like random sampling from **M** independent random variables.

But if the separation is good, then all the tools in random forests can be used to learn about the structure of the data…

Unsupervised learning ≠ clustering

We are trying to discover structure, which may or may not include what we usually think of as “clusters”.

Random forests may allow us to discover structure and may allow us to discover “clusters”.

One of the problems with “clustering” is that there is no single objective figure of merit.

A Proposed Test of Clustering Methods

- Consider real data with class labels.
- Erase the labels.
- Cluster the data.
- Do the clusters correspond to the original classes?

Let’s try it!

Illustration – Microarray data (10% error)

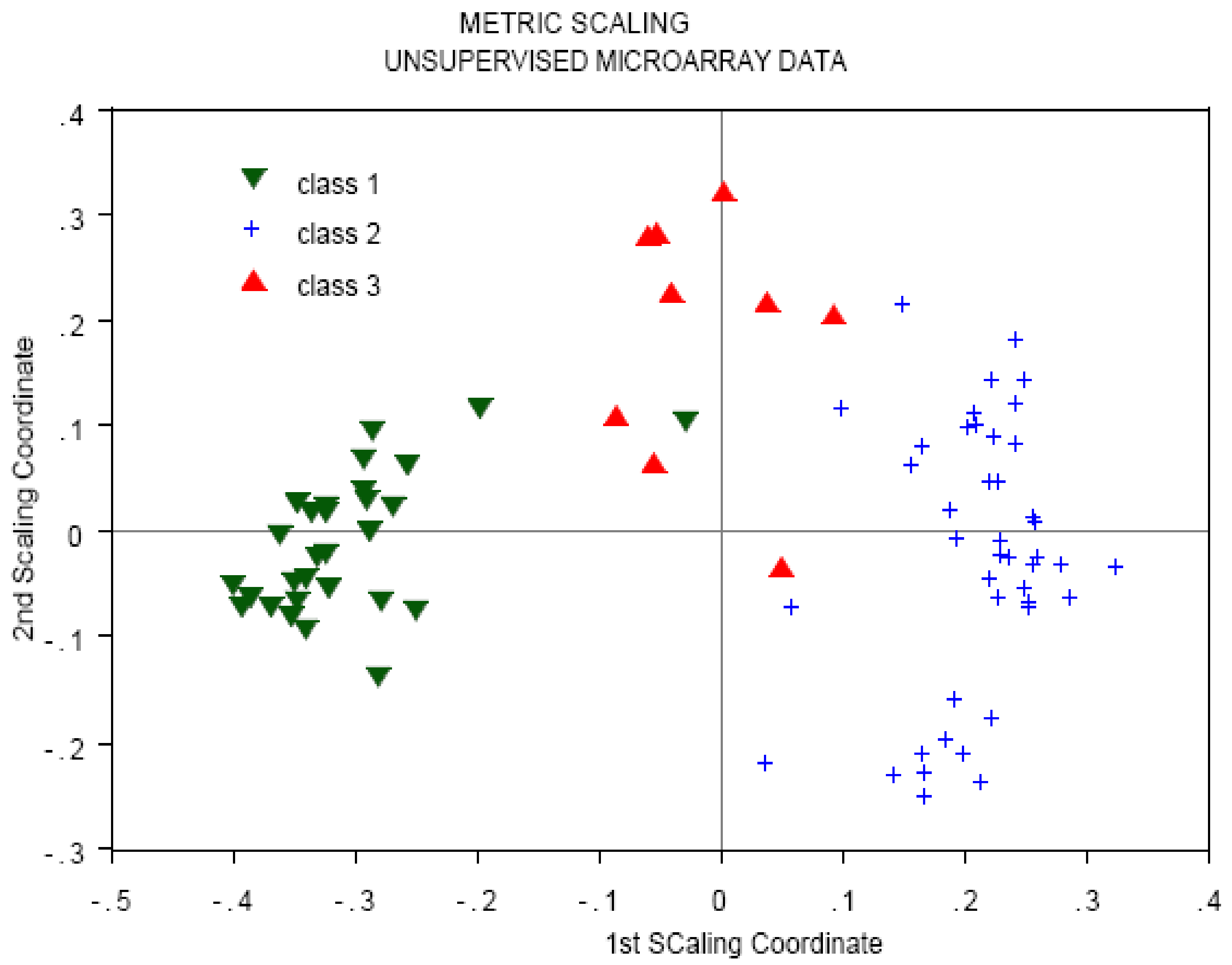


Illustration – Breast Cancer data

- 699 cases, 9 variables, two classes

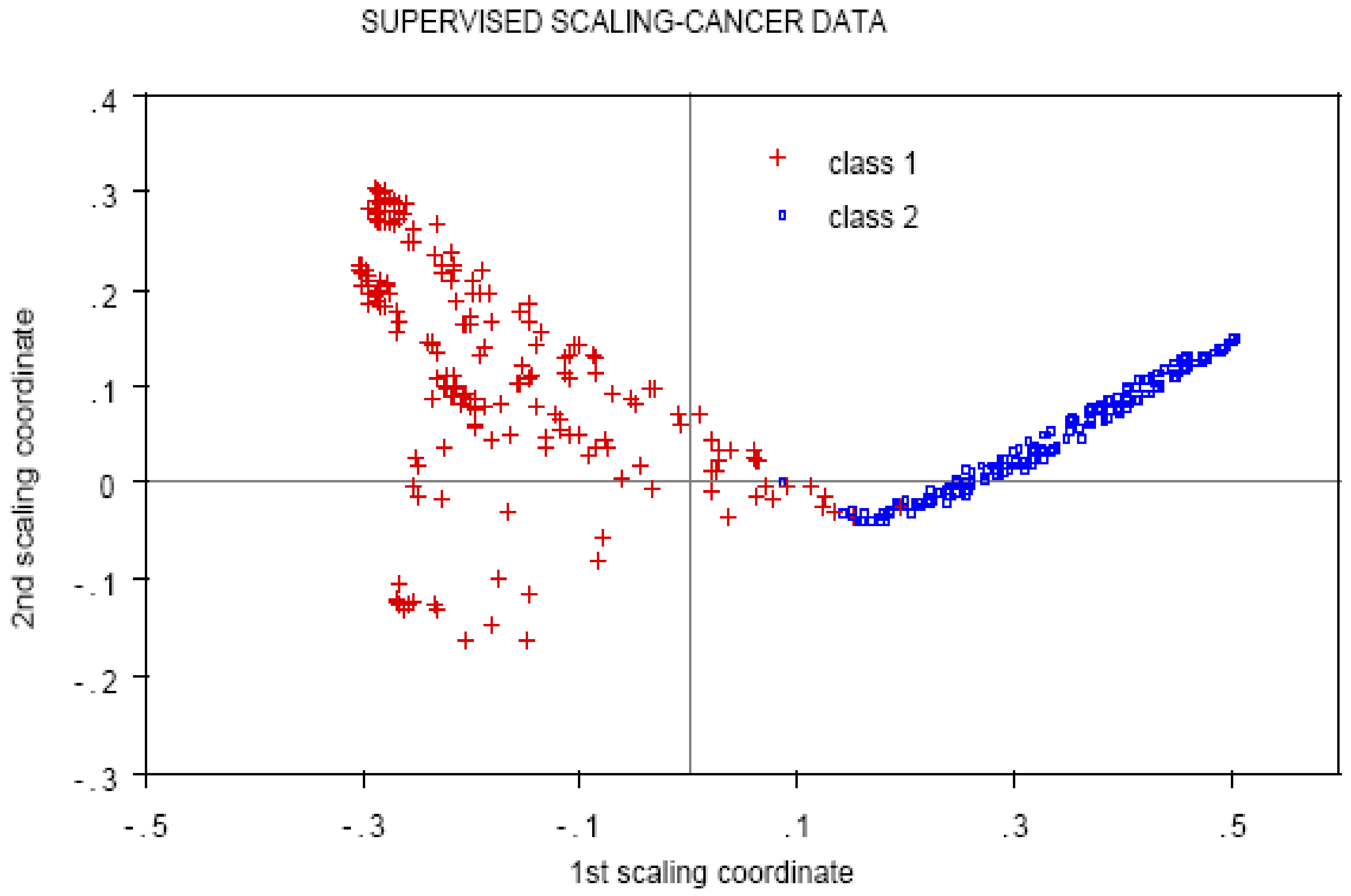


Illustration – Breast Cancer data

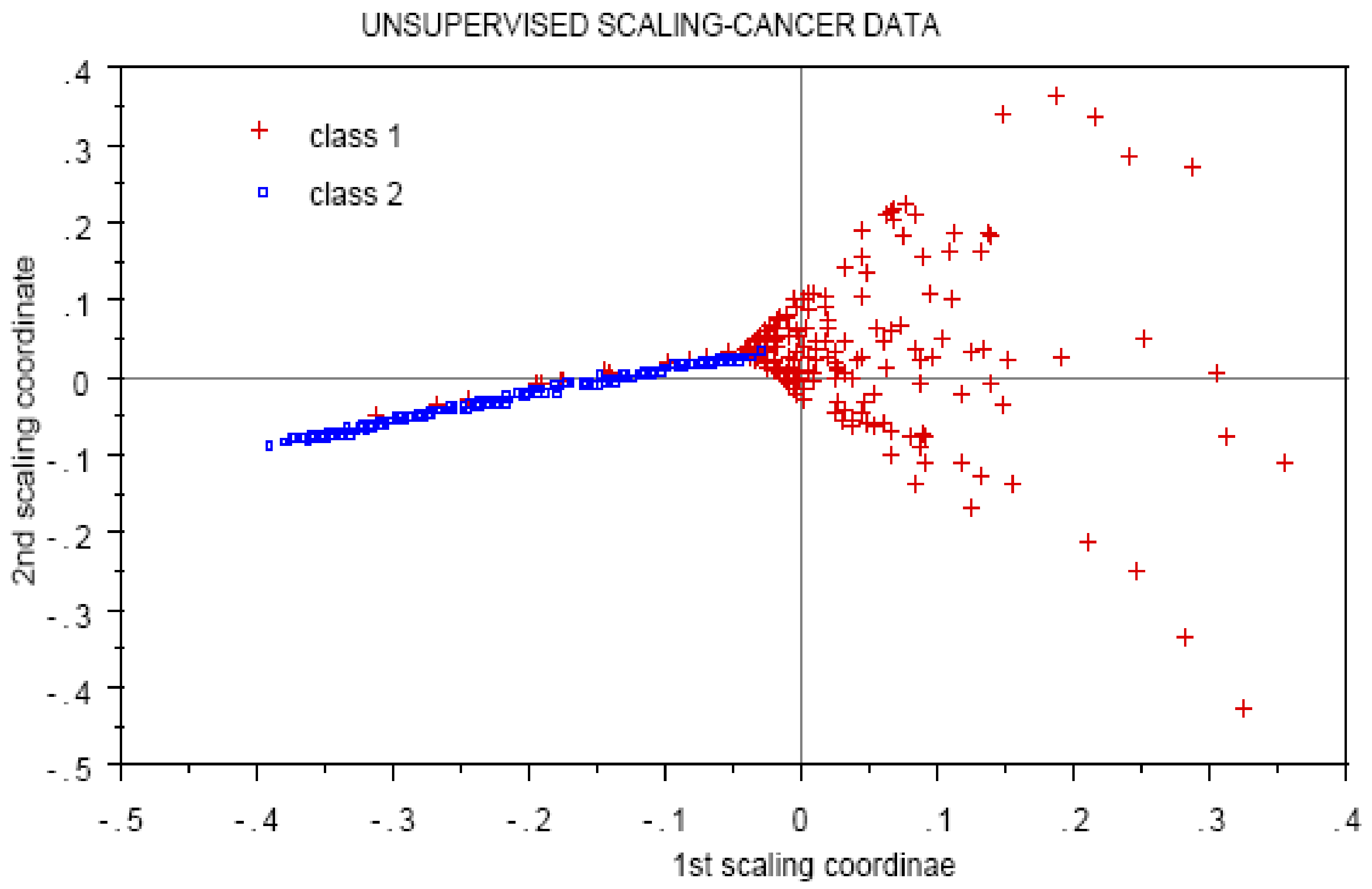


Illustration – Satellite data

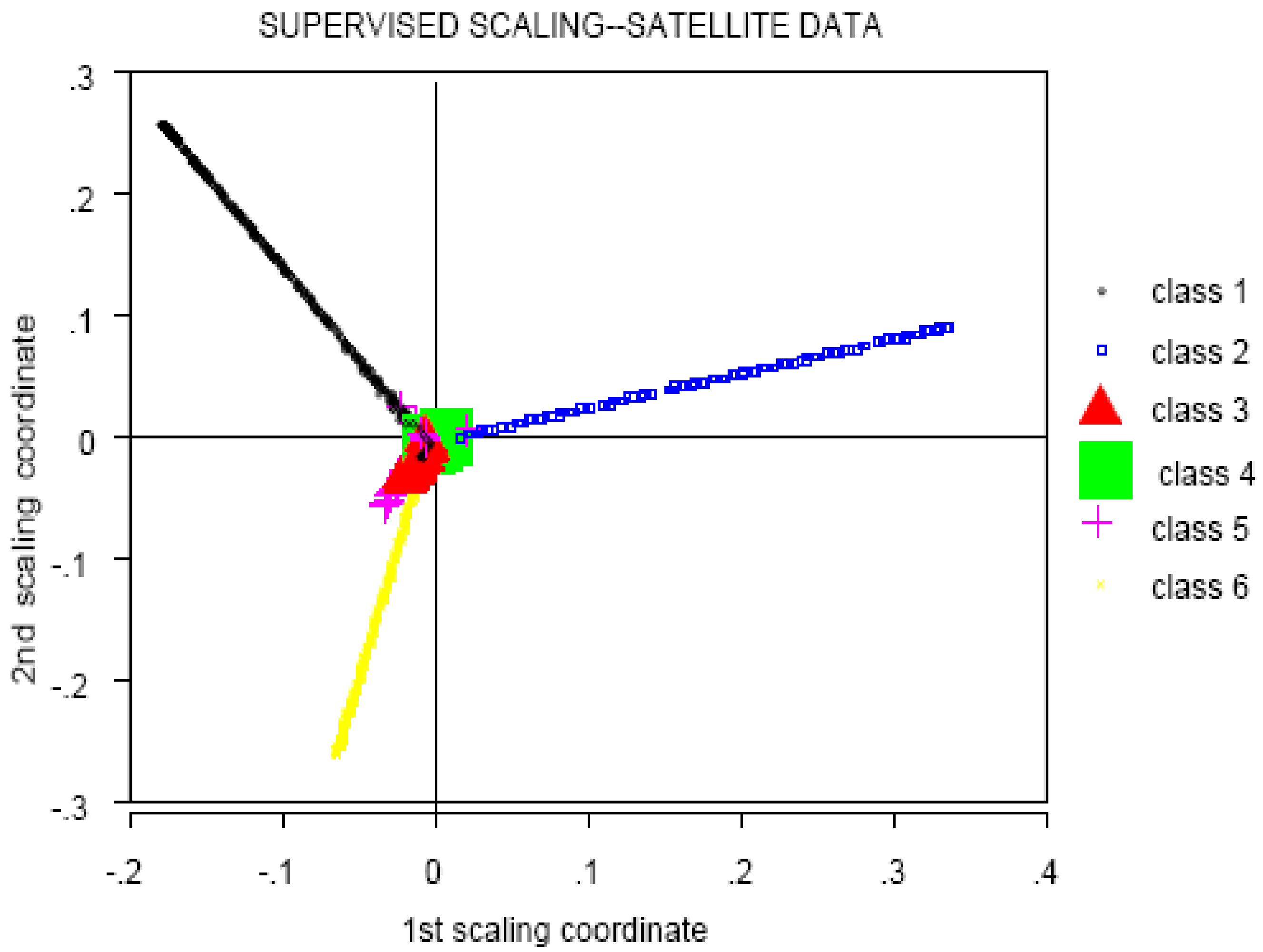


Illustration – Satellite data

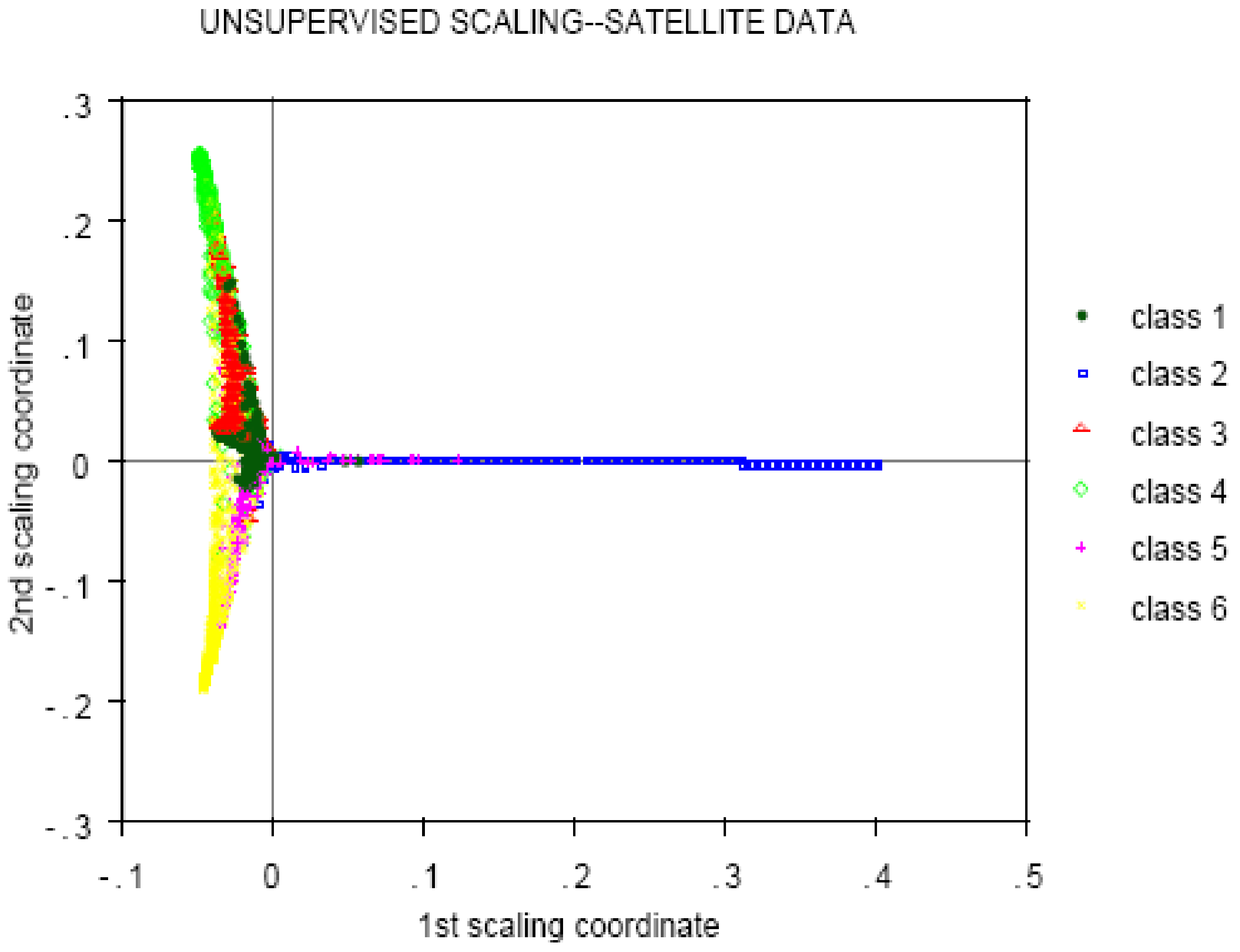


Illustration – Spectra data

The first 468 spectral intensities in the spectra of 764 compounds. Courtesy of Merck.

The challenge: find small cohesive groups of outlying cases.

Excellent separation: error rate of 0.5%.

No outliers were evident in the outlier plots.

However, the scaling picture…

Illustration – Spectra data

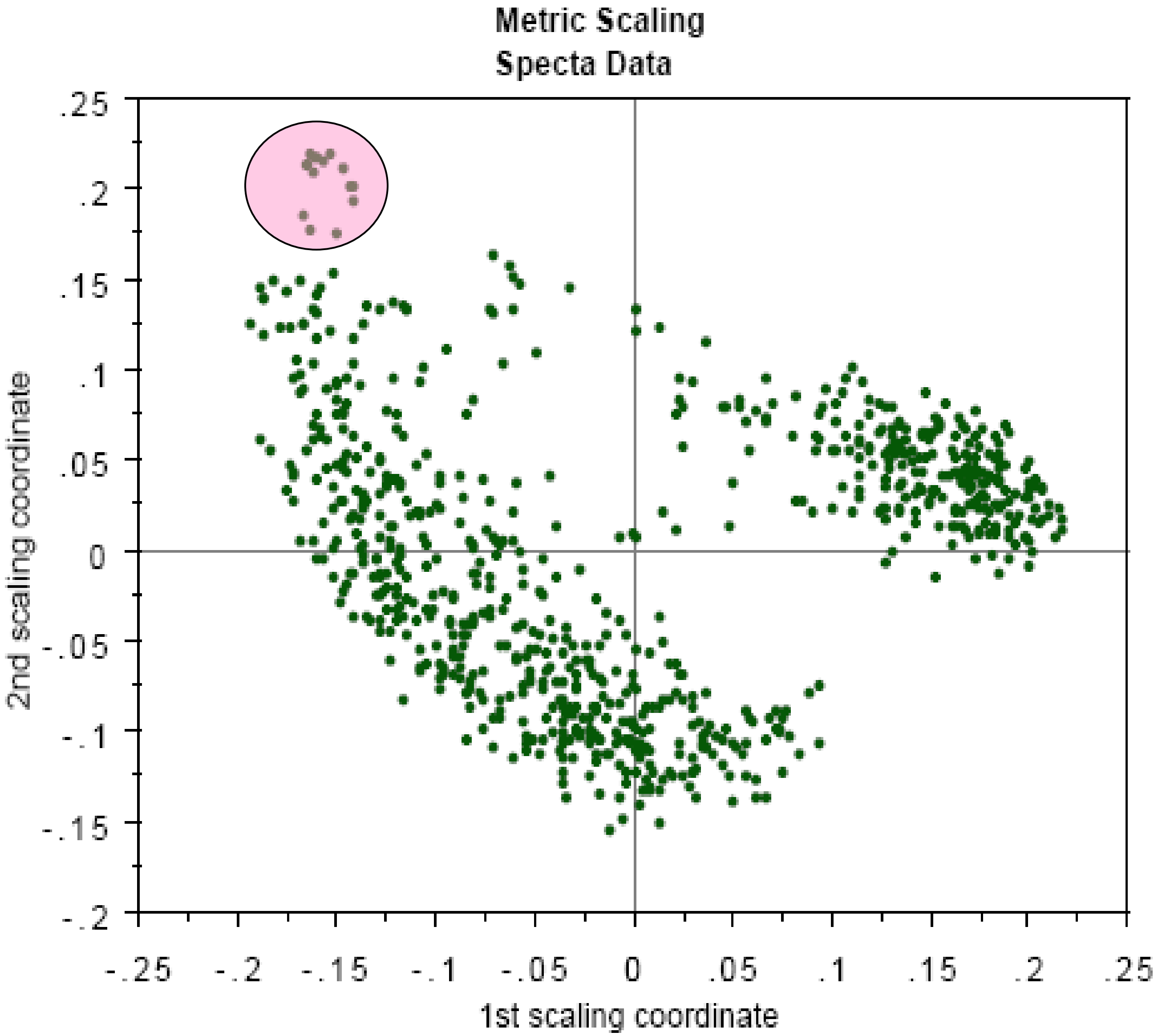
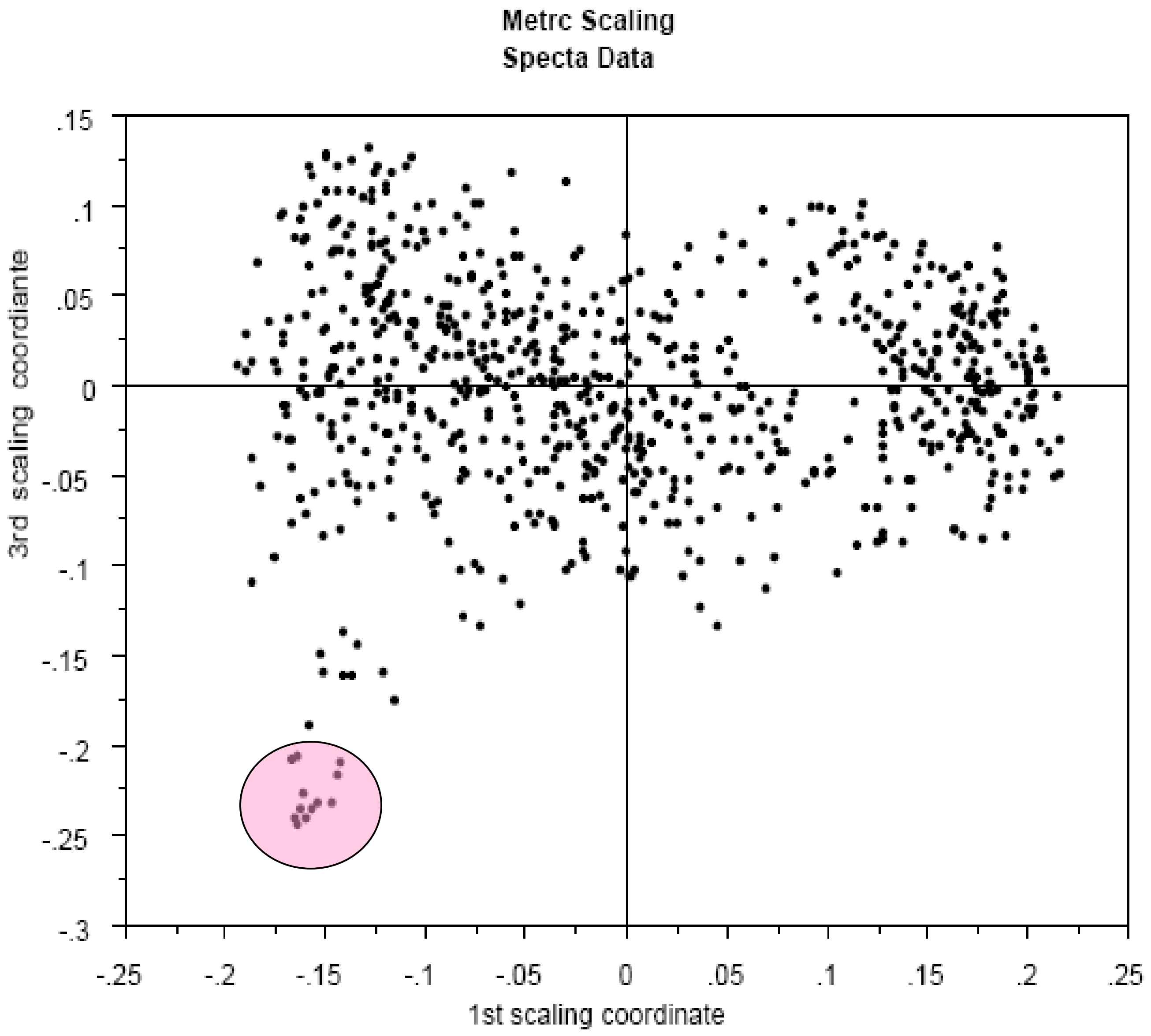


Illustration – Spectra data



Random Forests for Regression

Instead of fitting classification trees, fit regression trees.

The following carry over in the natural way (replacing misclassification rate with residual sum of squares)

- Variable importance
- Proximities
- Outlier detection
- Missing value replacement

Weighted Response

One useful capability is to give different weights to different ranges of the response variable.

Think about splitting up the response into a categorical with lots of classes (corresponding to different ranges of the response). Now it’s a classification problem, so the classes can be differentially weighted and we can find classwise variable importance. Similarly, local variable importance.

Random Forests for Regression

Expect a new version out soon!

Meanwhile, try classification with lots of classes.

Random Forests for Survival Analysis

- Model-free, unorthodox, experimental!
- Estimate individual survival curves.
- Cox model assumes that the effects of covariates are constant over time. Other models assume a fixed form of the time-dependence.

Random Forests for Survival Analysis

We can explore the evolution of the covariate effects over time. Do this in two ways:

- Correlation between $\log(\hat{S}(t,x_n))$ and covariates.
- Correlation between $\log(\hat{S}(t,x_n))$ and the “best” monotone transformation of the covariates.

Illustration – Vet-lung data

- 136 cases, 7% censoring, 6 covariates
  - Treatment (standard/test)
  - Cell type (squamous/small cell/adeno/large)
  - Karnofsky score
  - Months from diagnosis
  - Age in years
  - Prior Therapy (yes/no)

  - Survival in days
  - Status (1=dead, 0=censored)

Illustration – Vet-lung data

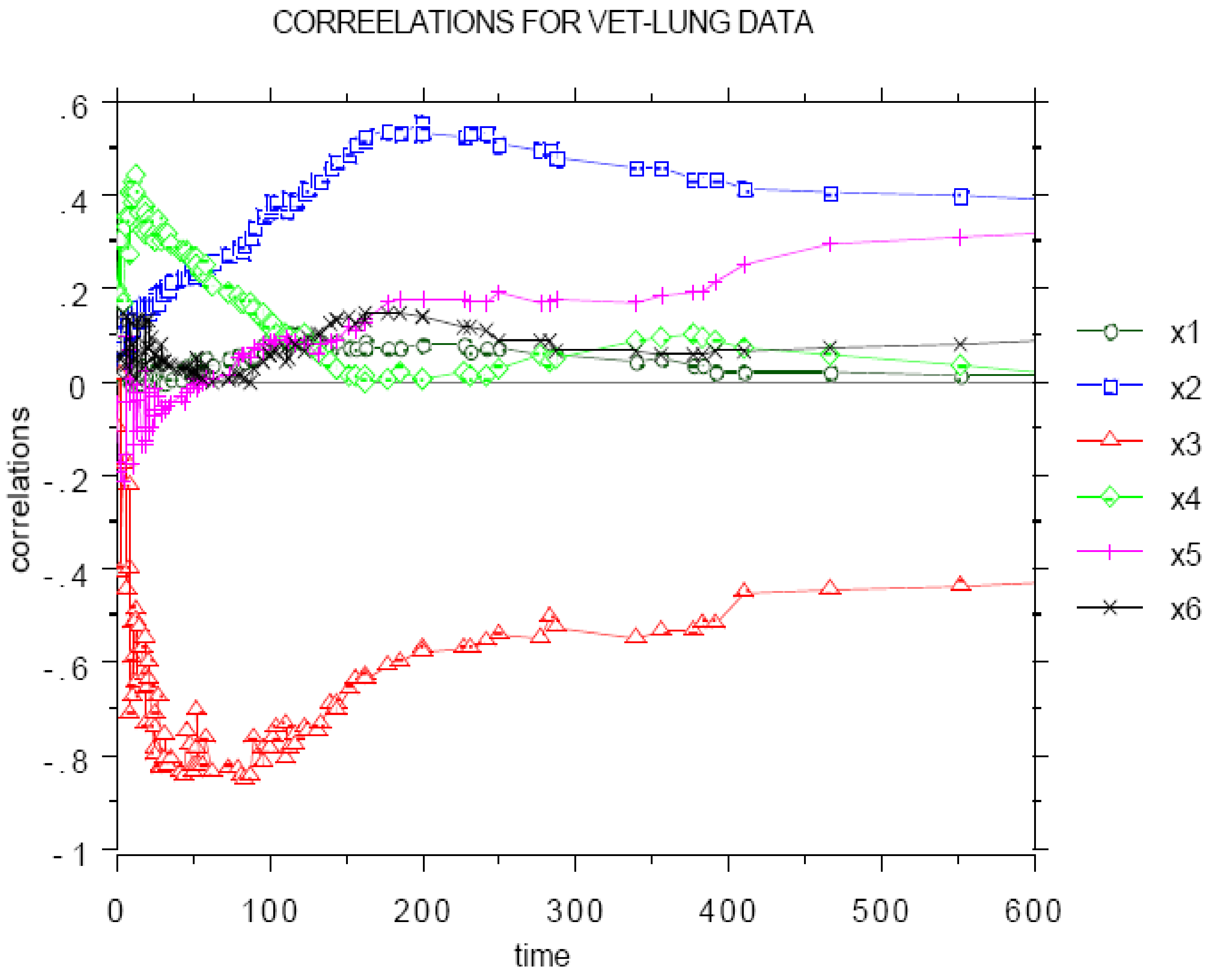


Illustration – Vet-lung data

- Variable 2 (Celltype) is weakening over time.
- Variable 3 (Karnofsky score) increases then settles.

Similarly interesting results have been obtained for other datasets.

Case Studies

Case study - Brain Cancer Microarrays

Pomeroy et al. Nature, 2002.

Dettling and Bühlmann, Genome Biology, 2002.

42 cases, 5,597 genes, 5 tumor types:

- 10 medulloblastomas BLUE
- 10 malignant gliomas PALE BLUE
- 10 atypical teratoid/rhabdoid tumors (AT/RTs) GREEN
- 4 human cerebella ORANGE
- 8 PNETs RED

Case study - Dementia

Data courtesy of J.T. Tschanz, USU

516 subjects

28 variables

2 classes:

- no cognitive impairment, blue (372 people)
- Alzheimer’s, red (144 people)

Case study - Autism

Data courtesy of J.D.Odell and R. Torres, USU

154 subjects (308 chromosomes)

7 variables, all categorical (up to 30 categories)

2 classes:

- Normal, blue (69 subjects)
- Autistic, red (85 subjects)

Case study - Metabolomics

(Lou Gehrig’s disease)
data courtesy of Metabolon (Chris Beecham)

63 subjects
317 variables
3 classes:
- blue (22 subjects)
- green (9 subjects)
- red (32 subjects)

Microarrays

March, 2005

Ramón Díaz-Uriarte, Sara Alvarez de Andrés

http://ligarto.org/rdiaz

Compared

- SVM, linear kernel
- KNN/crossvalidation (Dudoit et al. JASA 2002)
- DLDA
- Shrunken Centroids (Tibshirani et al. PNAS 2002)
- Random forests

Microarray Error Rates (.632+)

Data	*M*	*N*	*Class*	*SVM*	*KNN*	*DLDA*	*SC*	*RF*
Leukemia	3051	38	2	.014	.029	.020	.025	.051
Breast 2	4869	78	2	.325	.337	.331	.324	.342
Breast 3	4869	96	3	.380	.449	.370	.396	.351
NCI60	5244	61	8	.256	.317	.286	.256	.252
Adenocar	9868	76	2	.203	.174	.194	.177	.125
Brain	5597	42	5	.138	.174	.183	.163	.154
Colon	2000	62	2	.147	.152	.137	.123	.127
Lymphoma	4026	62	3	.010	.008	.021	.028	.009
Prostate	6033	102	2	.064	.100	.149	.088	.077
Srbct	2308	63	4	.017	.023	.011	.012	.021

Microarray Error Rates (.632+)

Data	*SVM*	*KNN*	*DLDA*	*SC*	*RF*	rank
Leukemia	.014	.029	.020	.025	.051	5
Breast 2	.325	.337	.331	.324	.342	5
Breast 3	.380	.449	.370	.396	.351	1
NCI60	.256	.317	.286	.256	.252	1
Adenocar	.203	.174	.194	.177	.125	1
Brain	.138	.174	.183	.163	.154	2
Colon	.147	.152	.137	.123	.127	2
Lymphoma	.010	.008	.021	.028	.009	2
Prostate	.064	.100	.149	.088	.077	2
Srbct	.017	.023	.011	.012	.021	4
Mean	.155	.176	.170	.159	.151	

Current and Future Work – Leo and Adele

- Proximities
- Regression and Survival Analysis
- Visualization – regression

  Rggobi?

Other Work on Random Forests

- Andy Liaw (Merck) - R package
- Jerome Friedman (Stanford) - Small trees on subsets
- Chao Chen et al. (UC Berkeley) - Unbalanced data
- Steve Horvath et al. (UCLA) – Unsupervised learning
- Ramón Díaz-Uriarte, Sara Alvarez de Andrés (CNIO Spain) - Variable Selection
- Grant Izmirlian (NIH/NCI) - Theory

Appendix

Nuts and Bolts of Fortran Code

Sections:

1. Read in the data
2. Essential settings
3. Common settings
4. Special purpose settings
5. Output control

1. Read in the data

Go to line 250:

```
open(16, file=‘data.train', status='old')
do n=1,nsample0
      read(16,*) (x(m,n),m=1,mdim),cl(n)
enddo
close(16)
if(ntest.gt.1) then
      open(17, file=‘data.test', status='old')
```

Control parameters

```
c         DESCRIBE DATA
   1               mdim=36, nsample0=4435, nclass=6, maxcat=1,
   1               ntest=0, labelts=0, labeltr=1,
c
c         SET RUN PARAMETERS
   2               mtry0=6, ndsize=1, jbt=100, look=100, lookcls=1,
   2               jclasswt=0, mdim2nd=0, mselect=0, iseed=4351,
c
c         SET IMPORTANCE OPTIONS
   3               imp=0, interact=0, impn=0, impfast=0,
c
c         SET PROXIMITY COMPUTATIONS
   4               nprox=0, nrnn=200,
c
c         SET OPTIONS BASED ON PROXIMITIES
   5               noutlier=0, nscale=0, nprot=0,
c
c         REPLACE MISSING VALUES
   6               code=-999, missfill=0, mfixrep=0,
c
c         GRAPHICS
   7               iviz=0,
c
c         SAVING A FOREST
   8               isaverf=0, isavepar=0, isavefill=0, isaveprox=0,
c
c         RUNNING A SAVED FOREST
   9               irunrf=0, ireadpar=0, ireadfill=0, ireadprox=0)
```

2. Essential settings

mdim = number of input variables

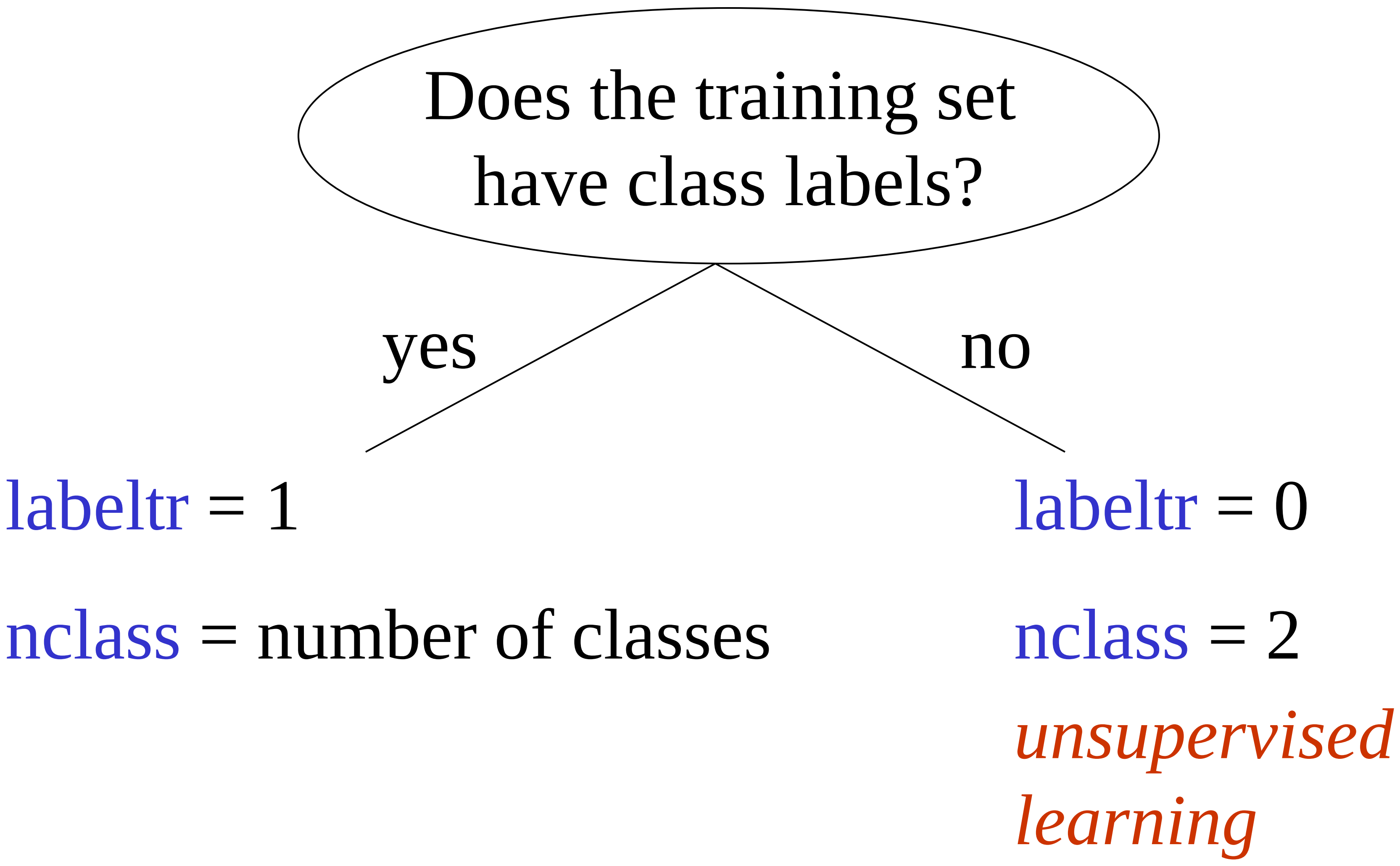
nsample0 = training set size

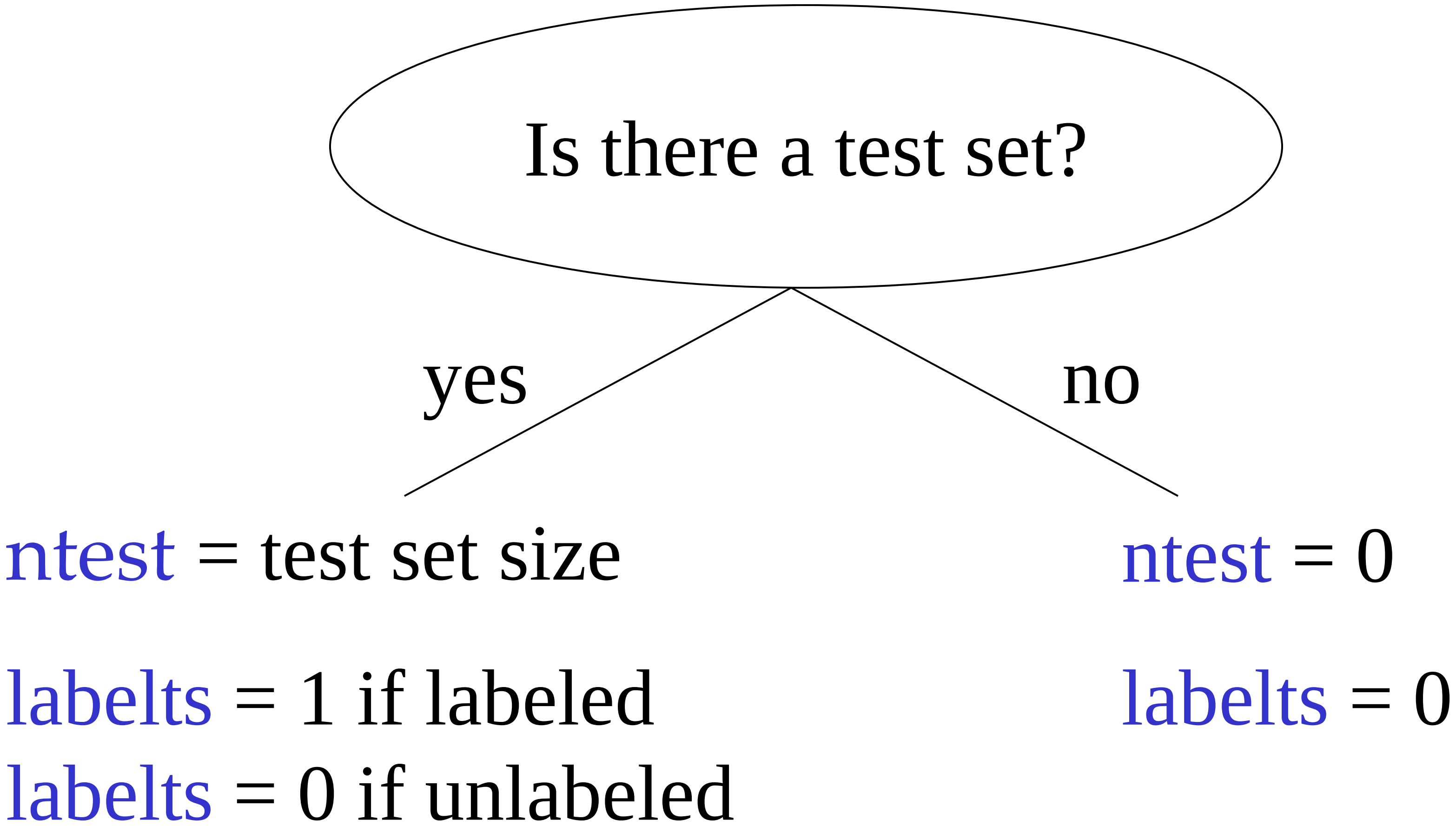
jbt = number of trees in the forest

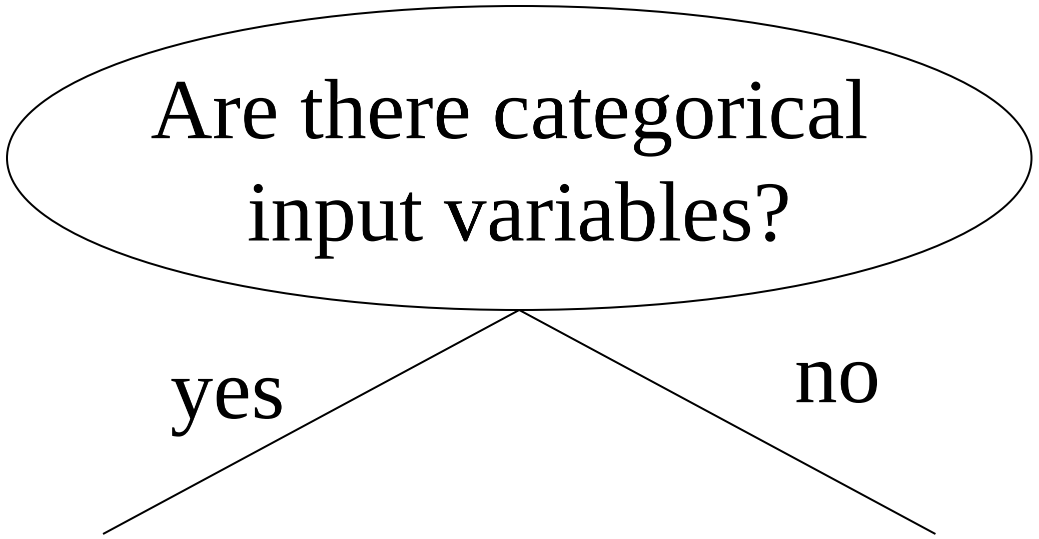
mtry0 = number of random splits

Choosing mtry

- Start with mtry = √mdim
- Try doubling and halving mtry
- If oob error rate is better with large (small) mtry, go larger (smaller) until the error rate starts to increase


Does the training set have class labels?
yes
no
labeltr = 1
nclass = number of classes
labeltr = 0
nclass = 2
unsupervised learning


Is there a test set?
yes
no
ntest = test set size
labelts = 1 if labeled
labelts = 0 if unlabeled
ntest = 0
labelts = 0



maxcat = max number of categories　　　　maxcat = 1

Go to line 280:

```
c       SET CATEGORICAL VALUES
        do m=1,mdim
                cat(m)=1
        enddo
```

3. Common settings

- Importance
- Class weights
- Proximities, scaling, prototypes, outliers
- Missing values
- Graphics

Importance

- imp = 0 → imp = 1 for variable importance
- mdim2nd = m does a second run with m most important variables (requires imp = 1)

Class weights

- jclasswt = 0 equal class weights
- jclasswt = 1 unequal class weights

Go to line 290:

```
c          SET CLASS WEIGHTS
           do j=1,nclass
                     classwt(j)=1
           enddo
```

note: setting class weights may require inspection of the oob error rate for each class, and suitable adjustment of the class weights

Proximities, scaling, prototypes, outliers

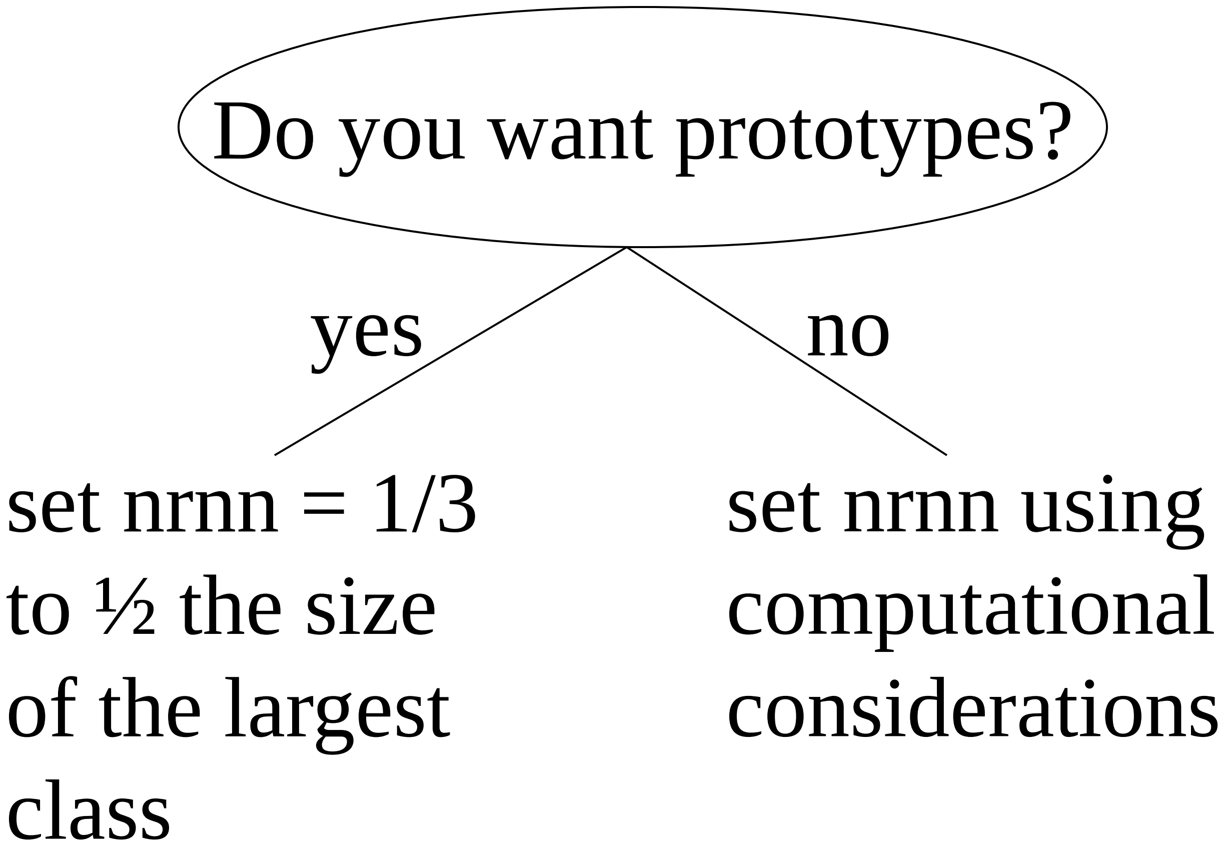
- nprox = 0 no proximities
- nprox = 1 training set proximities
- nprox = 2 training and test set proximities

The following options require nprox>0:

- noutlier = 1 outlier detection (2 adds test set)
- nscale = number of MDS variables
- nprot = number of prototypes for each class

Choosing nrnn

nrnn = the number of nearest neighbors for which to compute proximities


Do you want prototypes?
yes
no
set nrnn = 1/3 to ½ the size of the largest class
set nrnn using computational considerations

Missing values

- missfill = 1 does a fast, rough missing value replacement
- missfill = 2 does a careful missing value replacement (requires nprox>0)

Both of the above options require setting

code = missing value code

Note: the out-of-bag error rate is biased when missfill=2

Graphics

requires:

- iviz = 1
- impn=1
- imp=1
- nscale=3
- nprox=1

optionally: nprot=p and int=1 give additional plots in the display

4. Special purpose settings

ndsize = number of cases in terminal node

look = prints intermediate results

lookcls = 1 prints individual class results

interact = 1 computes variable interactions

impn = casewise variable importance (requires imp = 1)

impfast = 1 fast variable importance (doesn't require imp=1)

Special purpose settings (continued)

- mselect = 1 allows a subset of variables to be used throughout

  Go to line 265:

```
c SELECT SUBSET OF VARIABLES TO USE
  if(mselect.eq.0) then
    mdimt=mdim
    do k=1,mdim
            msm(k)=k
    enddo
  endif
```

- iseed = 4351 seed for the random number generator
- mfixrep = the number of iterations for careful replacement of missing values

Special purpose settings (continued)

- isaverf, irunrf to save forest and re-run
- isavepar, ireadpar to save parameter settings and recall
- isavefill, ireadfill to save missing value fills and re-use
- isaveprox, ireadprox to save proximities and re-use

5. Output control

Around line 70:

```
parameter(
 &   isumout= 1,             !0/1    1=summary to screen
 &   idataout= 1,            !0/1/2  1=train,2=adds test (7)
 &   impfastout= 1,          !0/1    1=gini fastimp         (8)
 &   impout= 1,              !0/1/2  1=imp,2=to screen (9)
 &   impnout= 1,             !0/1    1=impn (10)
 &   interout= 1,            !0/1/2  1=interaction,2=screen (11)
 &   iprotout= 1,            !0/1/2  1=prototypes,2=screen (12)
 &   iproxout= 1,            !0/1/2  1=prox,2=adds test (13)
 &   iscaleout= 1,           !0/1    1=scaling coords (14)
 &   ioutlierout= 1)         !0/1/2  1=train,2=adds test (15)
```