Lecture 2

Classification and Regression Trees

Mladen Kolar (mkolar@chicagobooth.edu)

Classification

Thus far, regression: predict a continuous value given some input

Classification is a predictive task in which the response takes values across discrete categories (i.e., not continuous), and in the most fundamental case, two categories (response variable is binary: $Y = 0$ or 1).

- Buy or not buy.
- Win or lose.
- Sick or healthy.
- Pay or default.
- Thumbs up or down.

We want to learn a mapping $f : X \mapsto Y$

- X are predictor (input) variables
- Y is the target class or class label

Similar to our regression setup, we observe pairs (x_i, y_i), $i = 1, \dots, n$, where y_i gives the class of the i-th observation, and $x_i \in \mathbb{R}^p$ are the measurements of p predictor variables

Though the class labels may actually be $y_i \in \{\text{healthy, sick}\}$ or $y_i \in \{\text{up, down, } \dots\}$, but we can always encode them as $y_i \in \{0, 1, \dots, K-1\}$ where K is the total number of classes.

Constructed from training data $(x_i, y_i), i = 1, \ldots, n$, we denote our classification rule by $\hat{f}(x)$; given any $x \in \mathbb{R}^p$, this returns a class label $\hat{f}(x) \in \{1, \ldots, K\}$

As before, we will see that there are two different ways of assessing the quality of $\hat{f}$: its predictive ability and interpretative ability

- In what situations would we care more about prediction error?
- And in what situations more about interpretation?

Binary classification and linear regression

Let's start off by supposing that $K = 2$, so that the response is $y_i \in \{0, 1\}$, for $i = 1, \ldots, n$

You already know a tool that you could potentially use in this case for classification: linear regression. Simply treat the response as if it were continuous, and find the linear regression coefficients of the response vector $y \in \mathbb{R}^n$ onto the predictors, that is

$$\hat{\beta}_0, \hat{\beta} = \arg\min \sum_i (y_i - \beta_0 - x_i^T \beta)^2$$

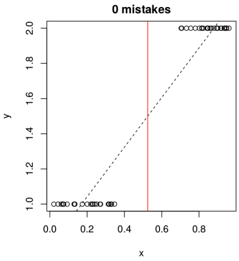
Then, given a new input $x_0 \in \mathbb{R}^p$, we predict the class to be

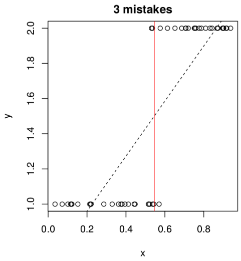
$$\hat{f}^{\mathrm{LS}}(x_0) = \begin{cases} 1 & \text{if } \hat{\beta}_0 - x_0^T \hat{\beta} \geq 0.5 \\ 0 & \text{if } \hat{\beta}_0 - x_0^T \hat{\beta} < 0.5 \end{cases}$$

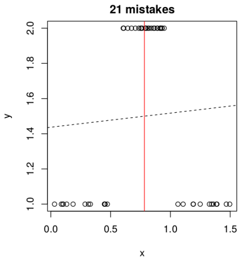
Can we use linear regression for this task?

(Note: since we included an intercept term in the regression, it doesn't matter whether we code the class labels as $\{1, 2\}$ or $\{0, 1\}$ or $\{-1, +1\}$, etc.)

In many instances, this actually works reasonably well. Examples:





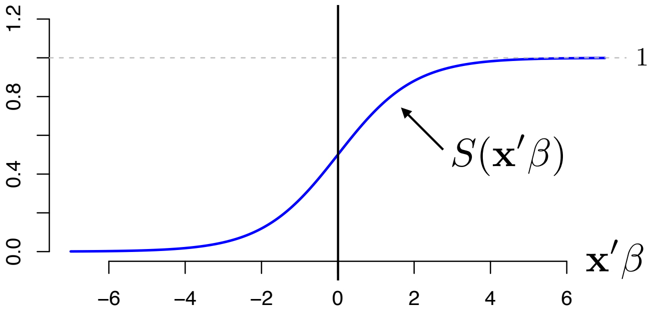


Overall, using linear regression in this way for binary classification is not a crazy idea.

Logistic regression

Modeling directly probability $P(Y = 1 \mid X)$

$$P(Y = 1 \mid X) = S(x'\beta), \quad S(a) = \frac{\exp(a)}{1 + \exp(a)}$$



However, in this class we want to learn a more generic specification for $Y \mid X = x$.

Given x, many methods will give

$$P(Y = y \mid x), \;\; y \in \{1, \ldots, K\}.$$

the conditional distribution of Y given x.

Some methods just give a predicted y in $\{1, \ldots, K\}$.

kNN

Given test x and training (x_i, y_i):

Numeric Y:

- find the k training observations with x_i closest to x.
- predict y with the average of the y values for the neighbors.

Categorical Y:

- find the k training observations with x_i closest to x.
- predict Y with the most frequent of the y values for the neighbors.
- estimate $P(Y = y \mid x)$ with the proportion on neighbors having $Y = y$.

Example: Forensic Glass

Can you tell what kind of glass it was from measurements on the broken shards??

Y: glass type, 3 categories.

$Y \in \{\text{WinF}, \text{WinNF}, \text{Other}\}$

- WinF: float glass window
- inNF: non-float window
- Other.

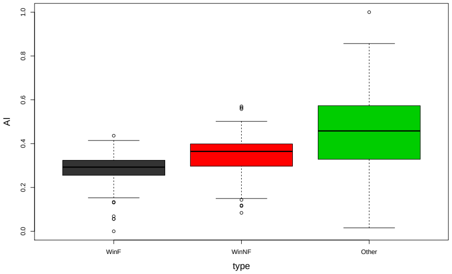
x: 3 numeric x's:

- $x_1 =$ RI: refractive index
- $x_2 =$ Al
- $x_3 =$ Na

Is Y related to $x_2 =$Al?

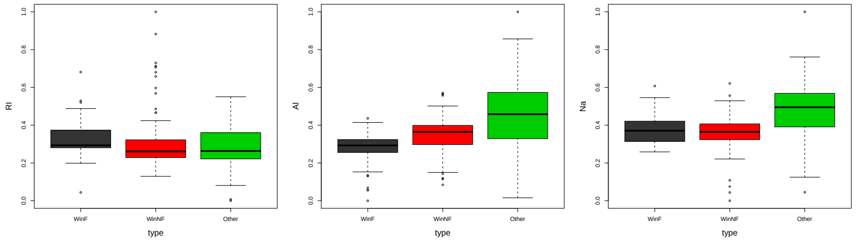
How do we plot a categorical Y vs a numeric x?

For each level of Y, pick off the subset of the data such that Y is at that level and then display the x values using a boxplot.



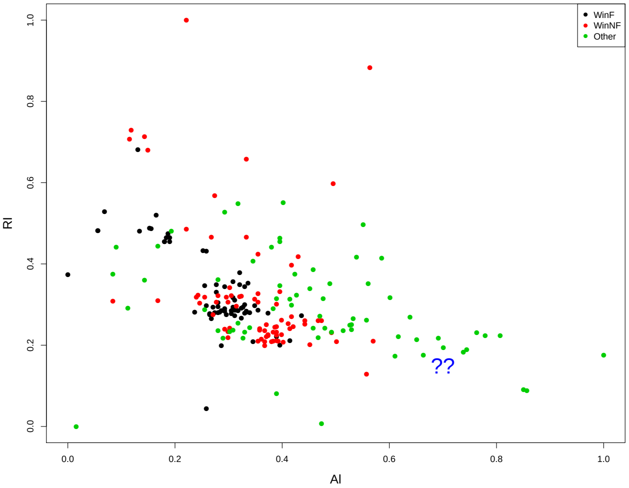
For big $x_2 =$Al, “Other” seems more likely for Y.

Plot y vs. each x.



x_2 =Al looks like a winner, but there may also be information about whether Y =WinF in x_1 =RI, and information about whether Y =Other in x_3 =Na.

How does kNN use the x's to predict the categorical Y? With just Al and RI:



If (Al,RI) = ??, what is prediction for Y?

Since we only have 214 observations we are just going to look at in-sample fit.

We used kNN with $k = 10$ and stored the result in `near`.

```
near$fitted[1:50]
 [1] WinF  WinNF WinNF WinF  WinF  WinNF WinF  WinF  WinF  WinF  WinNF WinF
[13] WinNF WinF  WinF  WinF  WinF  WinF  WinF  WinNF WinNF WinF  WinF  WinF
[25] WinF  WinF  WinF  WinF  WinF  WinF  WinF  WinF  WinF  WinF  WinF  Other
[37] WinF  WinF  WinF  WinF  WinF  WinF  WinF  WinF  WinF  WinNF WinNF WinF
[49] WinF  WinF
Levels: WinF WinNF Other
```

```
> near$prob[1:5,]
     WinF WinNF Other
[1,]  0.6   0.3   0.1
[2,]  0.4   0.4   0.2
[3,]  0.1   0.9   0.0
[4,]  0.7   0.3   0.0
[5,]  0.8   0.2   0.0
```

The two-way table relating the observed Y with the predicted Y is called the confusion matrix.

Data label on columns, "predicted" label on rows.

So, there are 58+11+1 observations with Y =WinF.
Of those 11 were predicted to be WinNF.

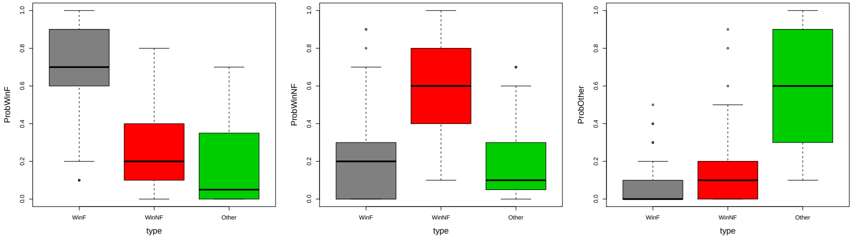
```
       WinF WinNF Other
  WinF    58    13    14
  WinNF   11    57    12
  Other    1     6    42
```

We like the diagonals big!
Missclassification rate: (214-(58+57+42))/214 = 0.27

pretty good !!

How good are the probabilities??

The first plot is $P(Y = WinF \mid x)$ vs. y=glass type.
The second plot is $P(Y = WinNF \mid x)$ vs. y=glass type.
The third plot is $P(Y = Other \mid x)$ vs. y=glass type.



pretty good !!

Tree-based methods

Trees

Tree based methods are a major player in data-mining.

Good:

- flexible fitters, capture non-linearity and interactions.
- do not have to think about scale of x variables.
- handles categorical and numeric y and x very nicely.
- fast.
- interpretable (when small).

Bad:

- Not the best in out-of-sample predictive performance (but not bad!).

But, if we bag or boost trees, we can get the best off-the-shelf prediction available. Bagging and Boosting are ensemble methods that combine the fit from many of tree models to get an overall predictor.

Tree-based methods

Tree-based based methods for predicting y from a feature vector x divide up the feature space into rectangles, and then fit a very simple model in each rectangle. This works both when y is discrete and continuous, i.e., both for classification and regression.

Rectangles can be achieved by making successive binary splits on the predictors variables $X_1, \ldots, X_p$. That is, we choose a variable X_j, $j = 1, \ldots, p$, divide up the feature space according to

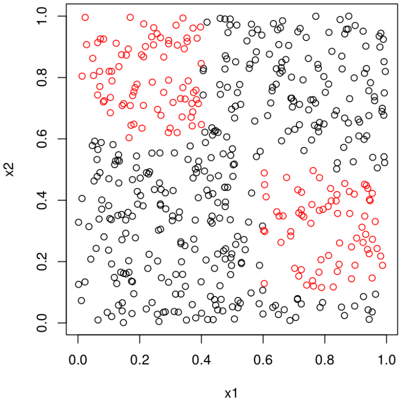
$$X_j \leq c \qquad \text{and} \qquad X_j > c$$

Then we proceed on each half

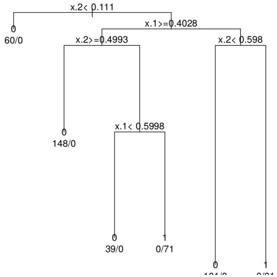
For simplicity, consider classification first (regression later). If a half is "pure", meaning that it mostly contains points from one class, then we don't need to continue splitting; otherwise, we continue splitting.

Example: simple classification tree

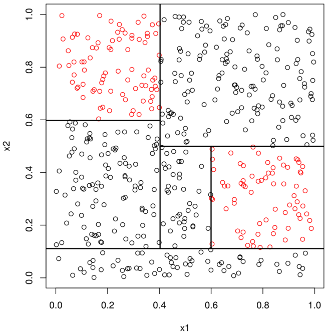
Example: n = 500 points in p = 2 dimensions, falling into classes 0 and 1, as marked by colors



Does dividing up the feature space into rectangles look like it would work here?

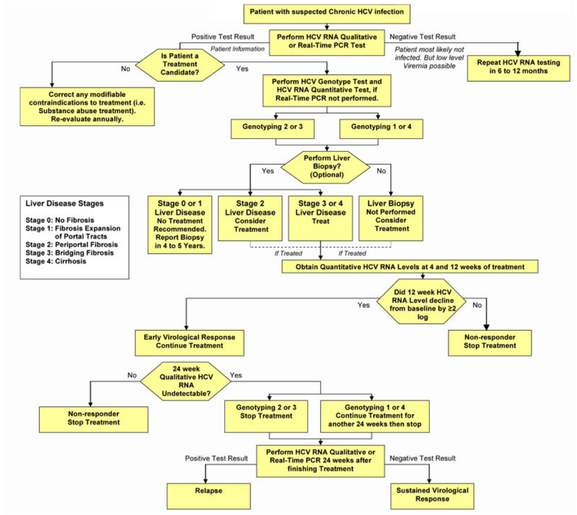


x.2< 0.111
0
60/0
x.1>=0.4028
x.2>=0.4993
x.2< 0.598
0
148/0
x.1< 0.5998
0
39/0
1
0/71
0
1



x2
x1
0.0
0.2
0.4
0.6
0.8
1.0

Example: HCV treatment flow chart


Patient with suspected Chronic HCV infection
Perform HCV RNA Qualitative or Real-Time PCR Test
Positive Test Result
Patient Information
Negative Test Result
Patient most likely not infected. But low level Viremia possible
Repeat HCV RNA testing in 6 to 12 months
Is Patient a Treatment Candidate?
No
Yes
Correct any modifiable contraindications to treatment (i.e. Substance abuse treatment). Re-evaluate annually.
Perform HCV Genotype Test and HCV RNA Quantitative Test, if Real-Time PCR not performed.
Genotyping 2 or 3
Genotyping 1 or 4
Perform Liver Biopsy? (Optional)
Yes
No
Liver Disease Stages
Stage 0: No Fibrosis
Stage 1: Fibrosis Expansion of Portal Tracts
Stage 2: Periportal Fibrosis
Stage 3: Bridging Fibrosis
Stage 4: Cirrhosis
Stage 0 or 1 Liver Disease No Treatment Recommended. Report Biopsy in 4 to 5 Years.
Stage 2 Liver Disease Consider Treatment
Stage 3 or 4 Liver Disease Treat
Liver Biopsy Not Performed Consider Treatment
If Treated
If Treated
Obtain Quantitative HCV RNA Levels at 4 and 12 weeks of treatment
Did 12 week HCV RNA Level decline from baseline by ≥2 log
Yes
No
Early Virological Response Continue Treatment
Non-responder Stop Treatment
24 week Qualitative HCV RNA Undetectable?
No
Yes
Non-responder Stop Treatment
Genotyping 2 or 3 Stop Treatment
Genotyping 1 or 4 Continue Treatment for another 24 weeks then stop
Perform HCV RNA Qualitative or Real-Time PCR 24 weeks after finishing Treatment
Positive Test Result
Negative Test Result
Relapse
Sustained Virological Response

(From `http://hcv.org.nz/wordpress/?tag=treatment-flow-chart`)

Classification trees

Classification trees are popular because they are interpretable, and maybe also because they mimic the way (some) decisions are made

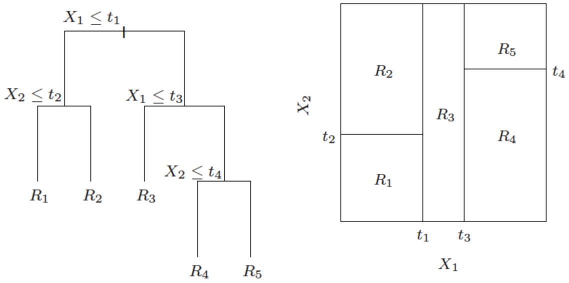
Let (x_i, y_i), $i = 1, \ldots n$ be the training data, where $y_i \in \{1, \ldots K\}$ are the class labels, and $x_i \in \mathbb{R}^p$ measure the p predictor variables. The classification tree can be thought of as defining m regions (rectangles) $R_1, \ldots R_m$, each corresponding to a leaf of the tree

We assign each R_j a class label $c_j \in \{1, \ldots K\}$. We then classify a new point $x \in \mathbb{R}^p$ by

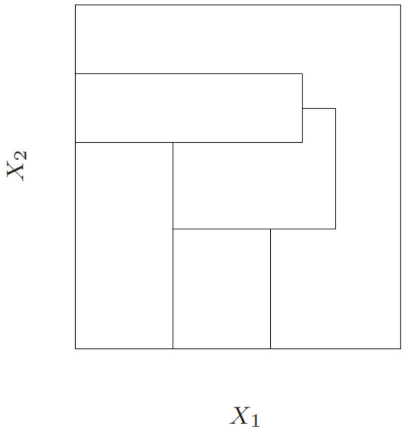
$$\hat{f}^{\text{tree}}(x) \;=\; \sum_{j=1}^{m} c_j \cdot 1\{x \in R_j\} \;=\; c_j \;\text{ such that }\; x \in R_j$$

Finding out which region a given point x belongs to is easy since the regions R_j are defined by a tree—we just scan down the tree. Otherwise, it would be a lot harder (need to look at each region)

Example: regions defined by a tree



Example: regions not defined by a tree



Predicted class probabilities

With classification trees, we can also get not only the predicted classes for new points but also the predicted class probabilties

Note that each region R_j contains some subset of the training data (x_i, y_i), $i = 1, \dots n$, say, n_j points. The predicted class c_j is just most common occuring class among these points. Further, for each class $k = 1, \dots K$, we can estimate the probability that the class label is k given that the feature vector lies in region R_j, $\mathrm{P}(C = k | X \in R_j)$, by

$$\hat{p}_k(R_j) = \frac{1}{n_j} \sum_{x_i \in R_j} 1\{y_i = k\}$$

the proportion of points in the region that are of class k. We can now express the predicted class as

$$c_j = \operatorname*{argmax}_{k=1,\dots K} \; \hat{p}_k(R_j)$$

Trees provide a good balance

	Model assumptions?	Estimated probabilities?	Interpretable?	Flexible?
LDA	Yes	Yes	Yes	No
LR	Yes	Yes	Yes	No
k-NN	No	No	No	Yes
Trees	No	Yes	Yes	Somewhat

	Predicts well?
LDA	Depends on X
LR	Depends on X
k-NN	If properly tuned
Trees	?

How to build trees?

There are two main issues to consider in building a tree:

1. How to choose the splits?
2. How big to grow the tree?

Think first about varying the depth of the tree . . . which is more complex, a big tree or a small tree? What tradeoff is at play here? How might we eventually consider choosing the depth?

Now for a fixed depth, consider choosing the splits. If the tree has depth d (and is balanced), then it has $\approx 2^d$ nodes. At each node we could choose any of p the variables for the split—this means that the number of possibilities is

$$p \cdot 2^d$$

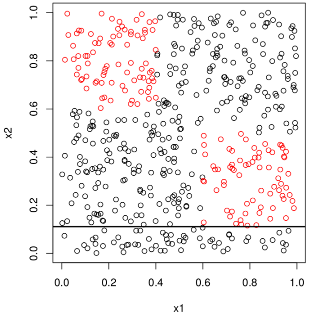
This is huge even for moderate d! And we haven't even counted the actual split points themselves.

The CART algorithm

The CART algorithm[1] chooses the splits in a top down fashion: then chooses the first variable to at the root, then the variables at the second level, etc.

At each stage we choose the split to achieve the biggest drop in misclassification error—this is called a greedy strategy. In terms of tree depth, the strategy is to grow a large tree and then prune at the end

Why grow a large tree and prune, instead of just stopping at some point? Because any stopping rule may be short-sighted, in that a split may look bad but it may lead to a good split below it



[1]Breiman et al. (1984), "Classification and Regression Trees"

Recall that in a region R_m, the proportion of points in class k is

$$\hat{p}_k(R_m) = \frac{1}{n_m} \sum_{x_i \in R_m} 1\{y_i = k\}.$$

The CART algorithm begins by considering splitting on variable j and split point s, and defines the regions

$$R_1 = \{X \in \mathbb{R}^p : X_j \leq s\}, \quad R_2 = \{X \in \mathbb{R}^p : X_j > s\}$$

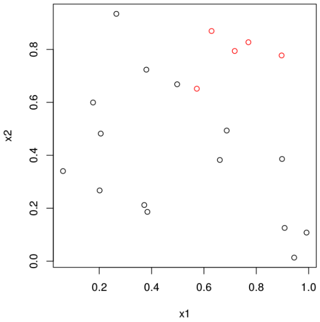
We then greedily chooses j, s by minimizing the misclassification error

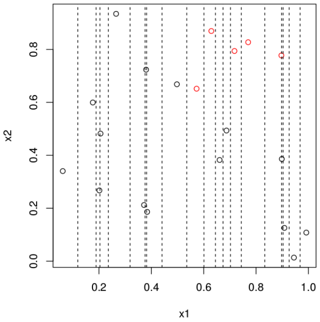
$$\underset{j,s}{\operatorname{argmin}} \left(\left[1 - \hat{p}_{c_1}(R_1)\right] + \left[1 - \hat{p}_{c_2}(R_2)\right] \right)$$

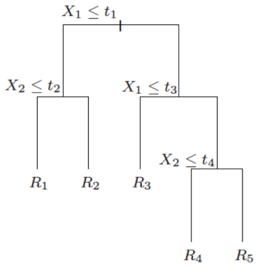
Here $c_1 = \operatorname{argmax}_{k=1,\ldots K} \hat{p}_k(R_1)$ is the most common class in R_1, and $c_2 = \operatorname{argmax}_{k=1,\ldots K} \hat{p}_k(R_2)$ is the most common class in R_2

Having done this, we now repeat this within each of the newly defined regions R_1, R_2. That is, it again considers splitting all variables j and split points s, within each of R_1, R_2, this time greedily choosing the pair that provides us with the biggest improvement in misclassification error.

How do we find the best split s? Aren't there infinitely many to consider? No, to split a region R_m on a variable j, we really only need to consider n_m splits (or $n_m - 1$ splits)







Continuing on in this manner, we will get a big tree T_0. Its leaves define regions $R_1, \dots R_m$. We then prune this tree, meaning that we collapse some of its leaves into the parent nodes

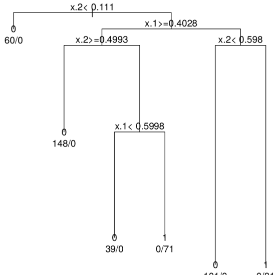
For any tree T, let $|T|$ denote its number of leaves. We define

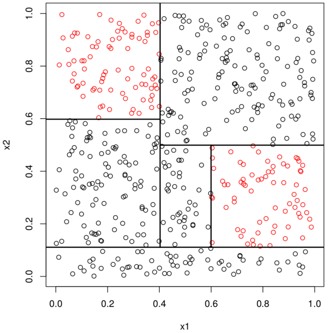
$$C_\alpha(T) = \sum_{j=1}^{|T|} \left[1 - \hat{p}_{c_j}(R_j)\right] + \alpha |T|$$

We seek the tree $T \subseteq T_0$ that minimizes $C_\alpha(T)$. It turns out that this can be done by pruning the weakest leaf one at a time. Note that α is a tuning parameter, and a larger α yields a smaller tree. CART picks α by 5- or 10-fold cross-validation

Example: simple classification tree

Example: n = 500, p = 2, and K = 2. We ran CART:





To use CART in R, you can use either of the functions rpart or tree, in the packages of those same names. When you call `rpart`, cross-validation is performed automatically; when you call `tree`, you must then call `cv.tree` for cross-validation.

Other impurity measures

We used misclassification error as a measure of the impurity of region R_j,

$$1 - \hat{p}_{c_j}(R_j)$$

But there are other useful measures too: the Gini index:

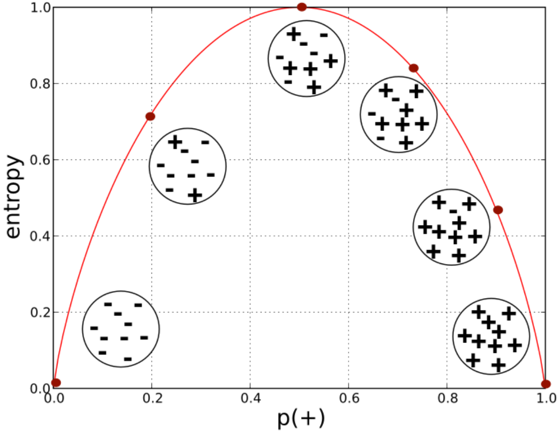
$$\sum_{k=1}^{K} \hat{p}_k(R_j)[1 - \hat{p}_k(R_j)]$$

and the cross-entropy or deviance:

$$-\sum_{k=1}^{K} \hat{p}_k(R_j) \log[\hat{p}_k(R_j)]$$

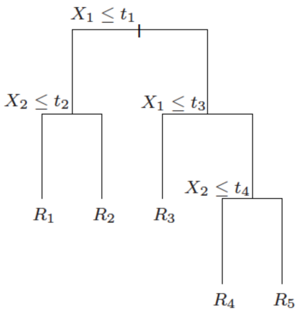
Using these measures instead of misclassification error is sometimes preferable because they are more sensitive to changes in class probabilities.

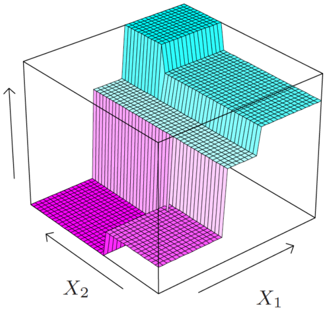
What does an impurity measure do?



Regression trees

Suppose that now we want to predict a continuous outcome instead of a class label. Essentially, everything follows as before, but now we just fit a constant inside each rectangle





The estimated regression function has the form

$$\hat{f}^{\mathrm{tree}}(x) = \sum_{j=1}^{m} c_j \cdot 1\{x \in R_j\} = c_j \quad \text{such that } x \in R_j$$

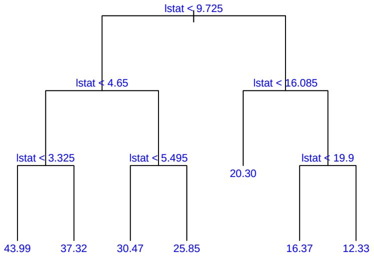
just as it did with classification. The quantities c_j are no longer predicted classes, but instead they are real numbers. How would we choose these? Simple: just take the average response of all of the points in the region,

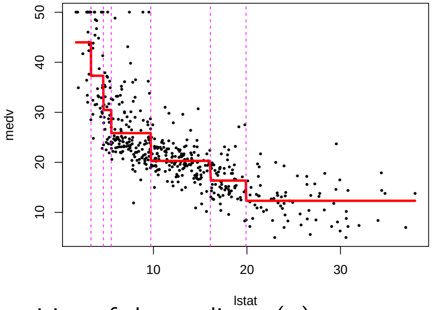
$$c_j = \frac{1}{n_j} \sum_{x_i \in R_j} y_i$$

The main difference in building the tree is that we use squared error loss instead of misclassification error (or Gini index or deviance) to decide which region to split. Also, with squared error loss, choosing c_j as above is optimal

Example: Boston housing

At left is the tree fit to the data. At each interior node there is a decision rule of the form $\{x < c\}$. If $x < c$ you go left, otherwise you go right. Each observation is sent down the tree until it hits a bottom node or leaf of the tree.

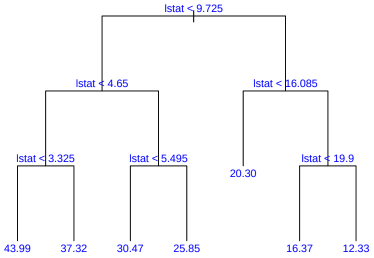


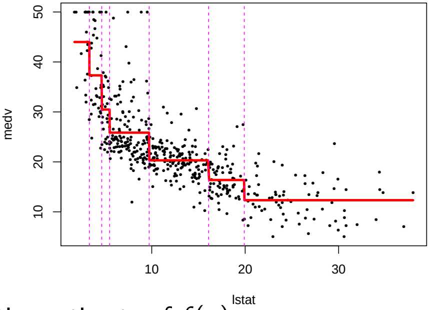


The set of bottom nodes gives us a partition of the predictor (x) space into disjoint regions. At right, the vertical lines display the partition. With just one x, this is just a set of intervals.

Example: Boston housing

Within each region (interval) we compute the average of the y values for the subset of training data in the region. This gives us the step function which is our $\hat{f}$. The $\bar{y}$ values are also printed at the bottom nodes (left plot).



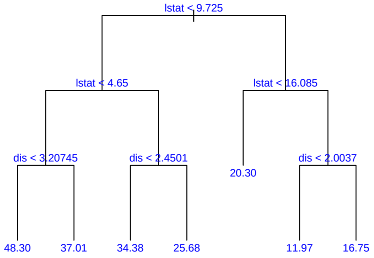


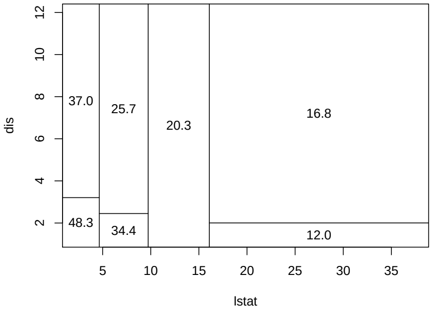
To predict, we just use our step function estimate of $f(x)$.

Equivalently, we drop x down the tree until it lands in a leaf and then predict the average of the y values for the training observations in the same leaf.

Example: Boston housing — two explanatory variables

Here is a tree with $x = (x_1, x_2) =$ (lstat,dis) and y=medv.





At right is the *partition* of the x space corresponding to the set of bottom nodes (leaves).

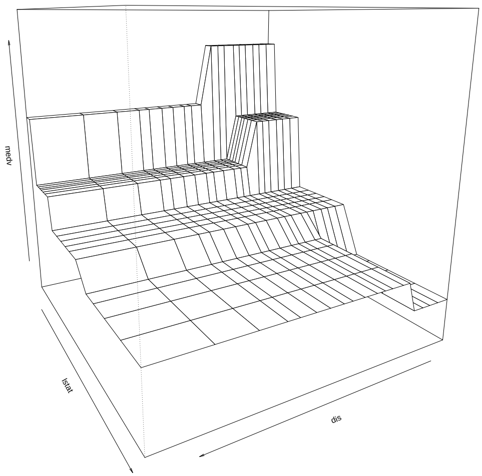
The average y for training observations assigned to a region is printed in each region and at the bottom nodes.

This is the regression function given by the tree.

It is a step function which can seem dumb, but it delivers non-linearity *and* interactions in a simple way and works with a lot of variables.

Notice the interaction.

The effect of `dis` depends on `lstat`!!



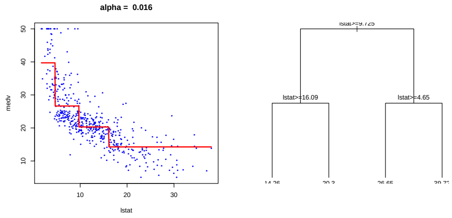
Choosing α

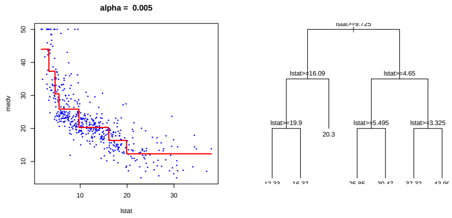
At right are three different tree fits we get from three different α values (using all the data).

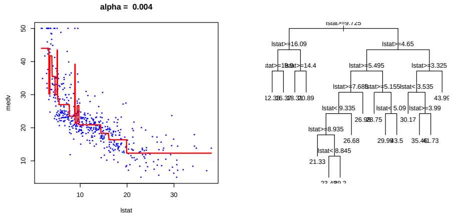
The smaller α is, the lower the penalty for complexity is, the bigger tree you get.

The top tree is a sub-tree of the middle tree, and the middle tree is a sub-tree of the bottom tree.

The middle α is the one suggested by CV.

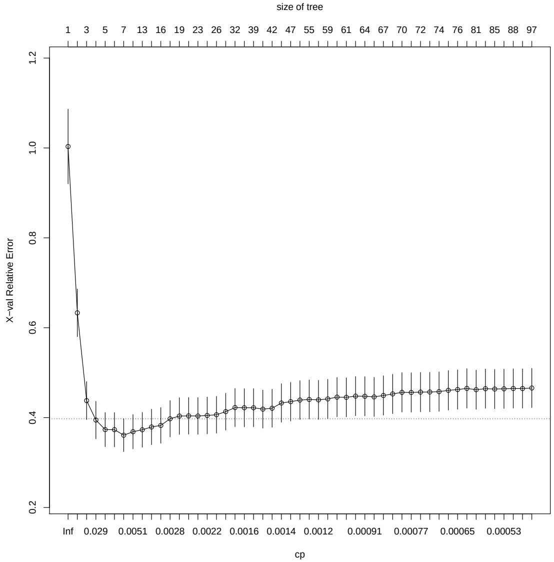






This is the CV plot giving by the R package `rpart` for `y=medv` `x=lstat`.

The error is relative to the error obtained with a single node (fit is $y = \bar{y}$, $\alpha = \infty$).



How well do trees predict?

Trees seem to have a lot of things going in the favor. So how is their predictive ability?

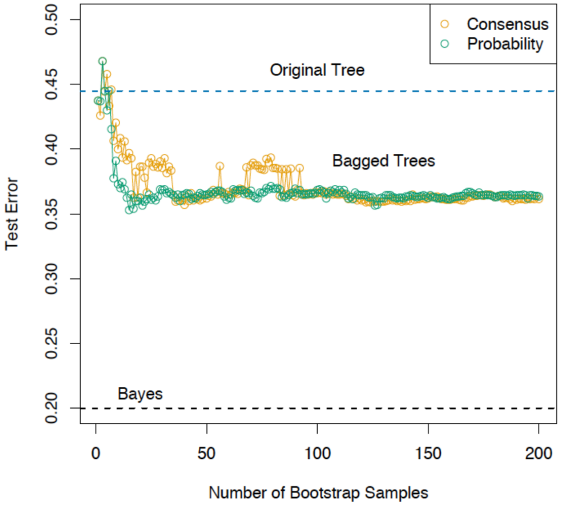
Unfortunately, the answer is not great. Of course, at a high level, the prediction error is governed by bias and variance, which in turn have some relationship with the size of the tree (number of nodes). A larger size means smaller bias and higher variance, and a smaller tree means larger bias and smaller variance.

But trees generally suffer from high variance because they are quite instable: a smaller change in the observed data can lead to a dramatically different sequence of splits, and hence a different prediction. This instability comes from their hierarchical nature; once a split is made, it is permanent and can never be "unmade" further down in the tree.

We'll learn some variations of trees have much better predictive abilities. However, their predictions rules aren't as transparent.

Combining trees

Fitting small trees on bootstrapped data sets, and averaging predictions at the end, can greatly reduce prediction error.


Consensus
Probability
Original Tree
Bagged Trees
Bayes
Test Error
Number of Bootstrap Samples