MY474: Applied Machine Learning for Social Science

Lecture 6: Regularization, Decision Trees

Blake Miller

13 November 2019

Agenda

1. Shrinkage Methods or Regularization
    - Ridge regression
    - Lasso regression
2. Trees

Shrinkage Methods or Regularization

Regularization

- **ridge regression** and **lasso** are regularized linear models
- The subset selection methods use **least squares** to fit a linear model that contains a subset of the predictors.
- As an alternative, we can fit a model containing all p predictors using a technique that **constrains** or **regularizes** the coefficient estimates, or equivalently, that **shrinks** the coefficient estimates towards zero.
- It may not be immediately obvious why such a constraint should improve the fit, but it turns out that shrinking the coefficient estimates can significantly reduce their variance.

Ridge regression

- Recall that the least squares fitting procedure estimates $\beta_0, \beta_1, \ldots, \beta_p$ using the values that minimize:

$$\mathrm{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2$$

- Ridge regression estimates add a **penalty** for large coefficient estimates, makes them **shrink** toward zero
- The goal is a balance between fit and size of coefficients
- The ridge regression coefficient estimates $\hat{\beta}^R$ are the values that minimize

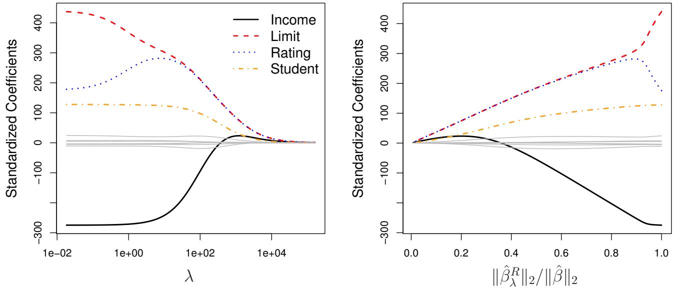
$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{i=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{i=1}^{p} \beta_j^2 = \mathrm{RSS} + \lambda \sum_{j=1}^{p} \beta_j^2$$

where $\lambda \geq 0$ is a **hyperparameter** chosen using cross-validation) that controls how much we want to shrink coefficients.

Ridge regression: continued

- As with least squares, ridge regression seeks coefficient estimates that fit the data well, by making the RSS small.
- However, the second term, $\lambda \sum_{j=1}^{p} \beta_j^2$, called a **shrinkage penalty**, is small when $\beta_1, \ldots, \beta_p$ are close to zero, and so it has the effect of shrinking the estimates of β_j towards zero, controlling the complexity of the model
- The tuning parameter λ controls the tradeoff between fit and complexity
  - When λ is large, we "shrink" coefficients towards zero
  - When λ is small, it moves closer to **ordinary least squares** estimates
- Selecting a good value for λ is critical; cross-validation is used for this.
- **Regularization** also lets us fit large models with more parameters than observations

Credit data example



Left: Least squares estimate on the left where $\lambda = 0$, Right: Least squares estimate on the right where the ridge ℓ_2 norm is the same as the OLS ℓ_2 norm.

- λ of 0 means coefficients are least squares estimates
- As λ increases in size, coefficients shrink toward zero (but they never reach zero).

Details of Previous Figure

- In the left-hand panel, each curve corresponds to the ridge regression coefficient estimate for one of the ten variables, plotted as a function of λ.
- The right-hand panel displays the same ridge coefficient estimates as the left-hand panel, but instead of displaying λ on the x-axis, we now display $\|\hat{\beta}_\lambda^R\|_2/\|\hat{\beta}\|_2$, where $\hat{\beta}$ denotes the vector of least squares coefficient estimates.
- The notation $\|\beta\|_2$ denotes the ℓ_2 norm (pronounced “ell 2”) of a vector, and is defined as $\|\beta\|_2 = \sqrt{\sum_{j=1}^p \beta_j^2}$.

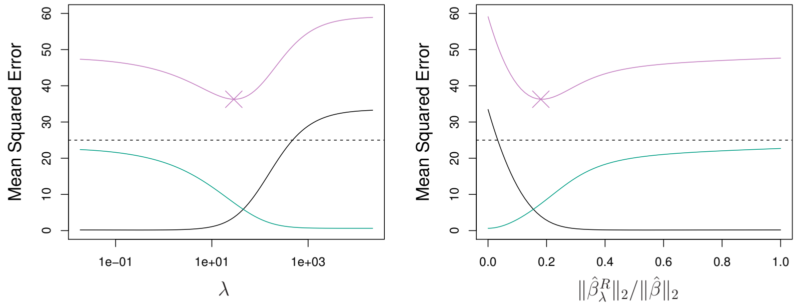
Ridge regression: scaling of predictors

- The standard least squares coefficient estimates are scale equivariant: multiplying X_j by a constant c simply leads to a scaling of the least squares coefficient estimates by a factor of $1/c$. In other words, regardless of how the jth predictor is scaled, $X_j\hat{\beta}_j$ will remain the same.
- In contrast, the ridge regression coefficient estimates can change *substantially* when multiplying a given predictor by a constant, due to the sum of squared coefficients term in the penalty part of the ridge regression objective function.
- Therefore, it is best to apply ridge regression after **standardizing the predictors**, or dividing that feature by its **standard deviation**:

$$\tilde{x}_{ij} = \frac{x_{ij}}{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2}$$

Why Does Ridge Regression Improve Over Least Squares?

- Regularization improves predictions by **trading variance for bias**, need **cross-validation** to determine the best tradeoff



Simulated data with $n = 50$ observations, $p = 45$ predictors, all having nonzero coefficients. Squared bias (black), variance (green), and test mean squared error (purple) for the ridge regression predictions on a simulated data set, as a function of λ and $\|\hat{\beta}^R_\lambda\|_2/\|\hat{\beta}\|_2$. The horizontal dashed lines indicate the minimum possible MSE. The purple crosses indicate the ridge regression models for which the MSE is smallest.

The Lasso

- Ridge regression does have one obvious disadvantage: it does not select models that involve just a subset of the variables
- Ridge regression will include all p predictors in the final model
- The Lasso is a relatively recent alternative to ridge regression that overcomes this disadvantage. The lasso coefficients, $\hat{\beta}_\lambda^L$, minimize the quantity

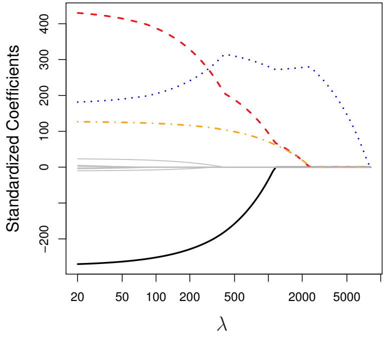
$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{i=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{i=1}^{p}|\beta_j| = \text{RSS} + \lambda\sum_{j=1}^{p}|\beta_j|$$

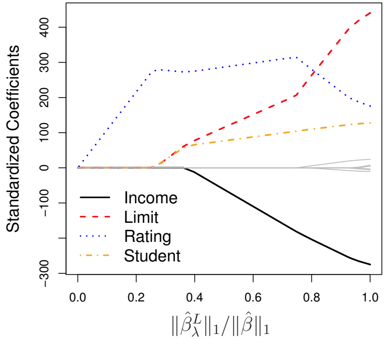
- In statistical parlance, the lasso uses an ℓ_1 (pronounced "ell 1") penalty instead of an ℓ_2 penalty. The ℓ_1 norm of a coefficient vector β is given by $\|\beta\|_1 = \sum|\beta_j|$.

The Lasso: continued

- As with ridge regression, the lasso shrinks the coefficient estimates towards zero.
- However, in the case of the lasso, the ℓ_1 penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero when the tuning parameter λ is sufficiently large.
- Hence, the lasso performs **variable selection**.
- We say that the lasso yields **sparse** models — that is, models that involve only a subset of the variables.
- As in ridge regression, selecting a good value of λ for the lasso is critical; cross-validation is again the method of choice.

Example: Credit dataset





The Variable Selection Property of the Lasso

- Why is it that the lasso, unlike ridge regression, results in coefficient estimates that are exactly equal to zero?
- One can show that the lasso and ridge regression coefficient estimates solve the problems below (λ may now look familiar if you recall lagrange multipliers from calculus)

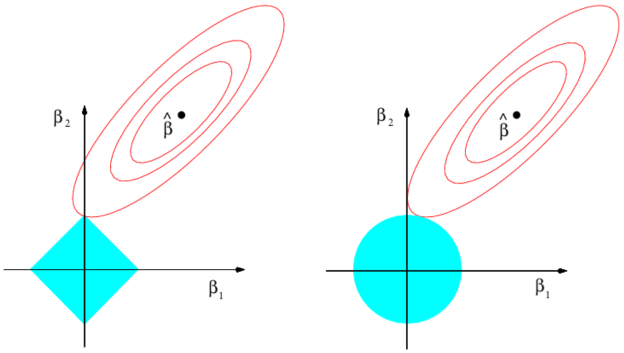
$$\underset{\beta}{\text{minimize}} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \quad \text{subject to} \quad \sum_{j=1}^{p} |\beta_j| < s$$

and

$$\underset{\beta}{\text{minimize}} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \quad \text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 < s$$

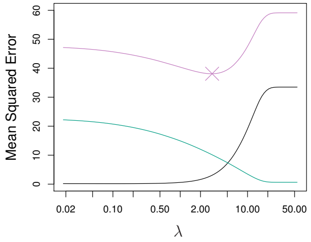
respectively.

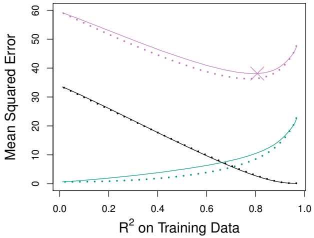
The Lasso Picture



- Left is lasso, right is ridge
- Contours represent RSS as we move away from the minimum least squares estimate of $\hat{\beta}$.
- The circle represents the constraint region (our budget s): ℓ_2 norm is circular, ℓ_1 is a diamond.
- The objective is then to find the minimum value of RSS that satisfies the constraint.

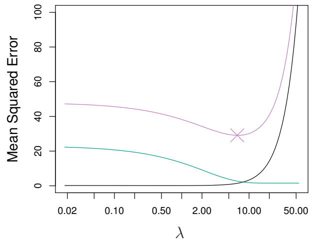
Comparing the Lasso and Ridge Regression

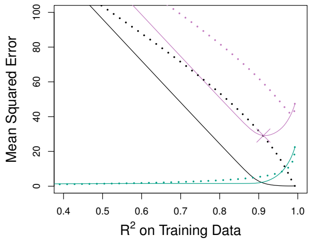




- Left: Plots of squared bias (black), variance (green), and test MSE (purple) for the lasso on simulated data set from the previous slide.
- Right: Comparison of squared bias, variance and test MSE between lasso (solid) and ridge (dashed). Both are plotted against their R^2 on the training data, as a common form of indexing. The crosses in both plots indicate the lasso model for which the MSE is smallest.

Comparing the Lasso and Ridge Regression: continued





- Left: Plots of squared bias (black), variance (green), and test MSE (purple) for the lasso. The simulated data is similar to that in the previous slide, except that now only two predictors are related to the response.
- Right: Comparison of squared bias, variance and test MSE between lasso (solid) and ridge (dashed). Both are plotted against their R^2 on the training data, as a common form of indexing. The crosses in both plots indicate the lasso model for which the MSE is smallest.

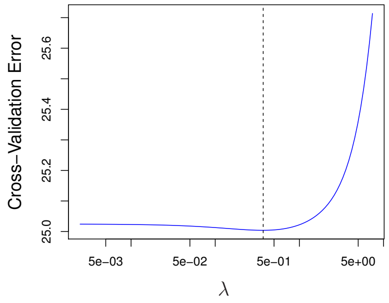
Conclusions

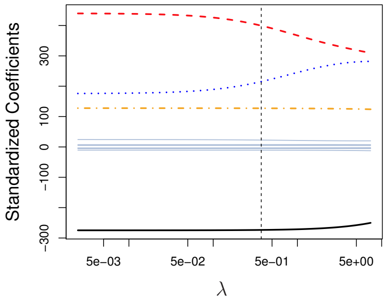
- These two examples illustrate that neither ridge regression nor the lasso will universally dominate the other.
- In general, one might expect the lasso to perform better when the response is a function of only a relatively small number of predictors.
- However, the number of predictors that is related to the response is never known a priori for real data sets.
- A technique such as cross-validation can be used in order to determine which approach is better on a particular data set.

Selecting the Tuning Parameter for Ridge Regression and Lasso

- Ridge regression and lasso we require a method to determine which of the models under consideration is best.
- That is, we require a method selecting a value for the tuning parameter λ or equivalently, the value of the constraint s.
- **Cross-validation** provides a simple way to tackle this problem. We choose a grid of λ values, and compute the cross-validation error rate for each value of λ.
- We then select the tuning parameter value for which the cross-validation error is smallest.
- Finally, the model is re-fit using all of the available observations and the selected value of the tuning parameter.

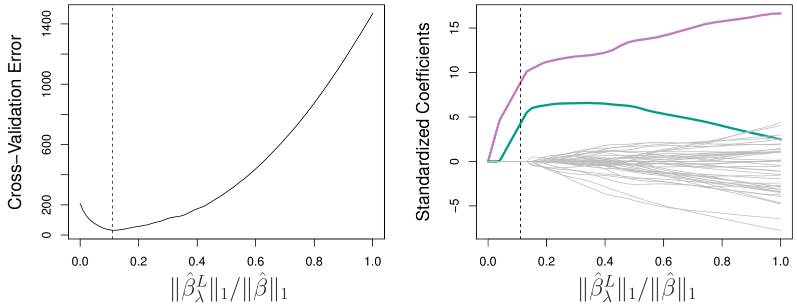
Credit data example





- Left: Cross-validation errors that result from applying ridge regression to the `Credit` data set with various values of λ.
- Right: The coefficient estimates as a function of λ. The vertical dashed lines indicates the value of λ selected by cross-validation.

Simulated data example



- Left: Ten-fold cross-validated MSE for the lasso, applied to the sparse simulated data set from earlier.
- Right: The corresponding lasso coefficient estimates are displayed. The vertical dashed lines indicate the lasso fit for which the cross-validation error is smallest.

Trees

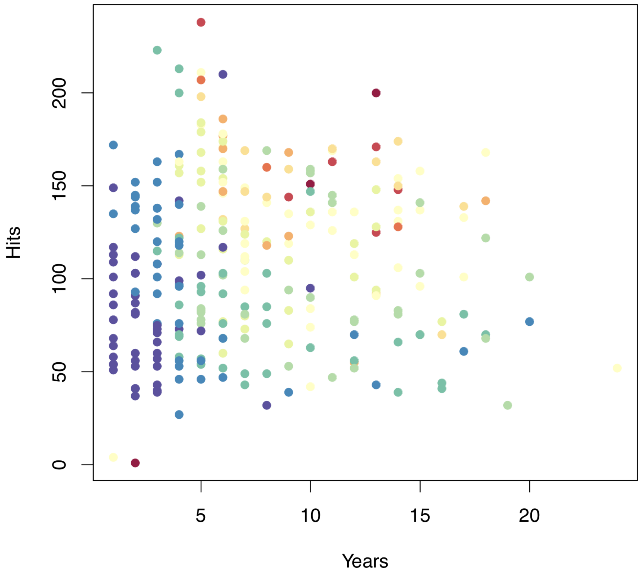
Tree-based Methods

- **Tree-based** methods are used for regression and classification.
- They **stratify** or **segment** the feature space into a number of simple regions.
- These rules can be summarized in tree.

Pros and Cons

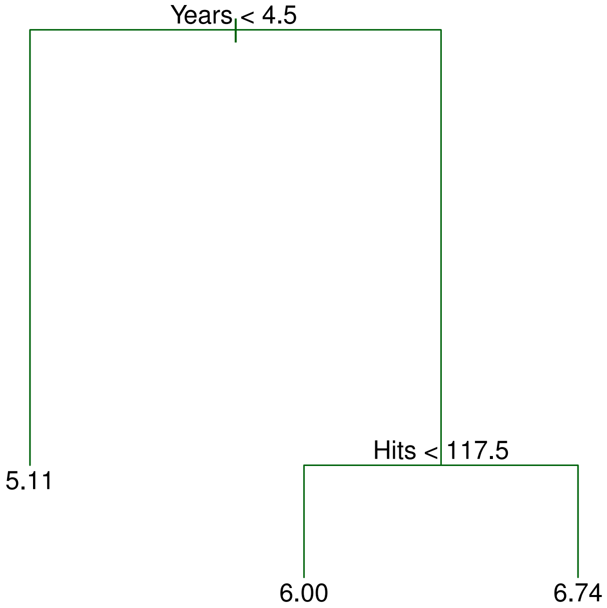
- Tree-based methods are simple and useful for **interpretation**.
- They usually do not perform as well as other models.
- Methods that use multiple trees such as **bagging**, **random forests**, and **boosting** perform much better.
- Methods using multiple trees are a special class of **ensemble methods** that we will discuss next week.

Baseball salary data: how would you stratify it?



Salary is color-coded from low (blue, green) to high (yellow,red)

Decision tree for these data

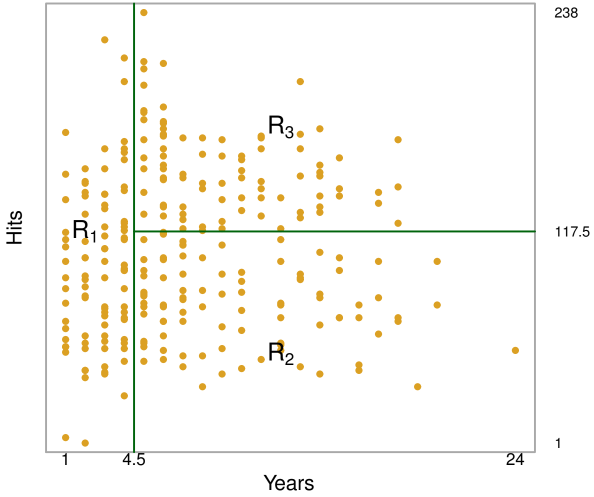


Details of previous figure

- For the Hitters data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year.
- At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$. For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to Years < 4.5, and the right-hand branch corresponds to Years ≥ 4.5.
- The tree has two internal nodes and three terminal nodes, or leaves. The number in each leaf is the mean of the response for the observations that fall there.

Results

- Overall, the tree stratifies or segments the players into three regions of predictor space: $R_1 = \{X | Years < 4.5\}$, $R_2 = \{X|\text{Years} \geq 4.5, \text{Hits} < 117.5\}$, and $R_3 = \{X| Years \geq 4.5, Hits \geq 117.5\}$.



Terminology for Trees

- In keeping with the **tree** analogy, the regions R_1, R_2, and R_3 are known as **terminal nodes**
- Decision trees are typically drawn upside down, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split are referred to as **internal nodes**
- In the hitters tree, the two internal nodes are indicated by the text Years < 4.5 and Hits < 117.5.

Interpretation of Results

- `Years` is the most important factor in determining `Salary`, and players with less experience earn lower salaries than more experienced players.
- Given that a player is less experienced, the number of `Hits` that he made in the previous year seems to play little role in his `Salary`.
- But among players who have been in the major leagues for five or more years, the number of `Hits` made in the previous year does affect `Salary`, and players who made more `Hits` last year tend to have higher salaries.
- Surely an over-simplification, but compared to a regression model, it is easy to display, interpret and explain

Details of the tree-building process

1. We divide the predictor space — that is, the set of possible values for $X_1, X_2, \ldots, X_p$ — into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.
2. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

More details of the tree-building process

- In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model.
- The goal is to find boxes $R_1, \ldots, R_J$ that minimize the RSS, given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

- In other words, the goal is to partition the data into boxes that balance **homogeneity** within boxes and **distinctness** among boxes.

More details of the tree-building process

- Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a **top-down**, **greedy** approach that is known as recursive binary splitting.
- The approach is **top-down** because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is **greedy** because at each step of the tree-building process, the **best** split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

Why Binary Splits?

- Multiway splits into more than two groups fragment the data too quickly, leaving insufficient data at the next level down.
- Since multiway splits can be achieved by a series of binary splits, the latter are preferred.

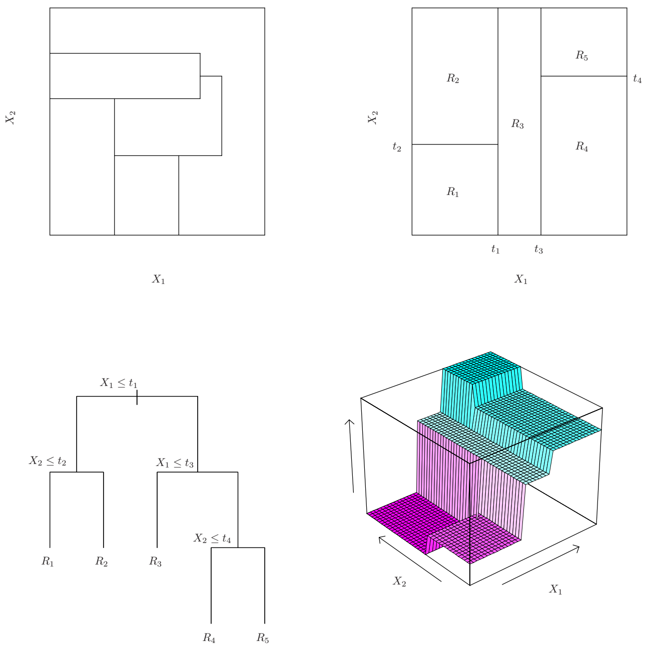
Details— Continued

- We first select the predictor Xj and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in **RSS**.
- Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions.
- However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions. We now have three regions.
- Again, we look to split one of these three regions further, so as to minimize the RSS. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

Predictions

- We predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.
- A five-region example of this approach is shown in the next slide.

Visualizing Tree Regression



Details of previous figure

- Top Left: A partition of two-dimensional feature space that could not result from recursive binary splitting.
- Top Right: The output of recursive binary splitting on a two-dimensional example.
- Bottom Left: A tree corresponding to the partition in the top right panel.
- Bottom Right: A perspective plot of the prediction surface corresponding to that tree.

Pruning a tree

- The process described above may produce good predictions on the training set, but is likely to **overfit** the data, leading to poor test set performance. **Why?**

Pruning a tree

- The process described above may produce good predictions on the training set, but is likely to **overfit** the data, leading to poor test set performance. **Why?**
- Imagine a tree with one observation in each terminal node. **What is our training error?**

Pruning a tree

- The process described above may produce good predictions on the training set, but is likely to **overfit** the data, leading to poor test set performance. **Why?**
- Imagine a tree with one observation in each terminal node. **What is our training error?**
- A smaller tree with fewer splits (that is, fewer regions $R_1, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.

Pruning a tree

- The process described above may produce good predictions on the training set, but is likely to **overfit** the data, leading to poor test set performance. **Why?**
- Imagine a tree with one observation in each terminal node. **What is our training error?**
- A smaller tree with fewer splits (that is, fewer regions $R_1, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.
- One possible alternative to the process described above is to grow the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold.

Pruning a tree

- The process described above may produce good predictions on the training set, but is likely to **overfit** the data, leading to poor test set performance. **Why?**
- Imagine a tree with one observation in each terminal node. **What is our training error?**
- A smaller tree with fewer splits (that is, fewer regions $R_1, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.
- One possible alternative to the process described above is to grow the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold.
- This strategy will result in smaller trees, but is too **short-sighted**: a seemingly worthless split early on in the tree might be followed by a very good split — that is, a split that leads to a large reduction in RSS later on.

Pruning a tree— continued

- A better strategy is to grow a very large tree T_0, and then **prune** it back in order to obtain a **subtree**
- **Cost complexity pruning** — also known as **weakest link pruning** — is used to do this
- we consider a sequence of trees indexed by a nonnegative tuning parameter α. For each value of α there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i: x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

is as small as possible. Here $|T|$ indicates the number of terminal nodes of the tree T, R_m is the rectangle (i.e. the subset of predictor space) corresponding to the mth terminal node, and $\hat{y}_{R_m}$ is the mean of the training observations in R_m.

Choosing the best subtree

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
  - $\alpha = 0$ corresponds to minimizing training error (will choose the largest tree, all pure nodes)
  - Large α results in small trees
- We select an optimal value $\hat{\alpha}$ using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$.

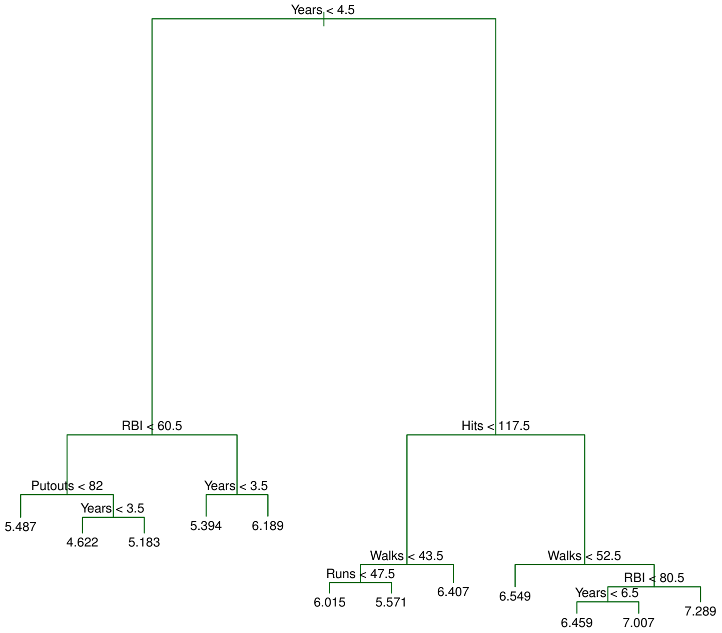
Summary: tree algorithm

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. For each $k = 1, \ldots, K$:
   - Repeat Steps 1 and 2 on the $\frac{K-1}{K}$th fraction of the training data, excluding the kth fold.
   - Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α. Average the results, and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

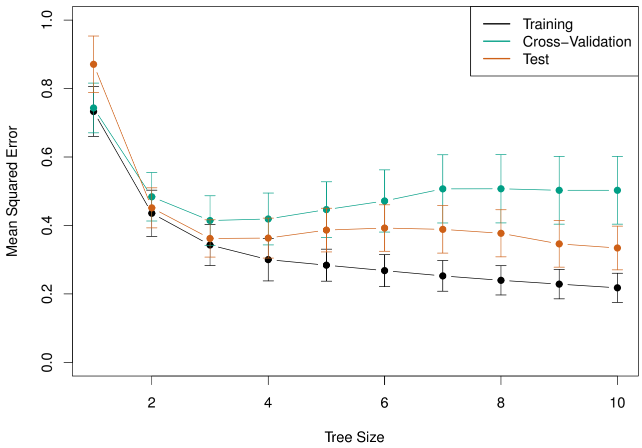
Baseball example continued

- Recall the baseball example from before. How did we decide on that tree?
- First, we randomly divided the data set in half, yielding 132 observations in the training set and 131 observations in the test set.
- We then built a large regression tree on the training data and varied α in in order to create subtrees with different numbers of terminal nodes.
- Finally, we performed six-fold cross-validation in order to estimate the cross-validated MSE of the trees as a function of α.

Baseball example continued



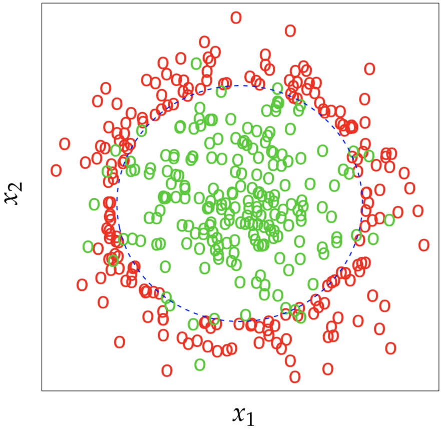
Baseball example continued



Classification Trees

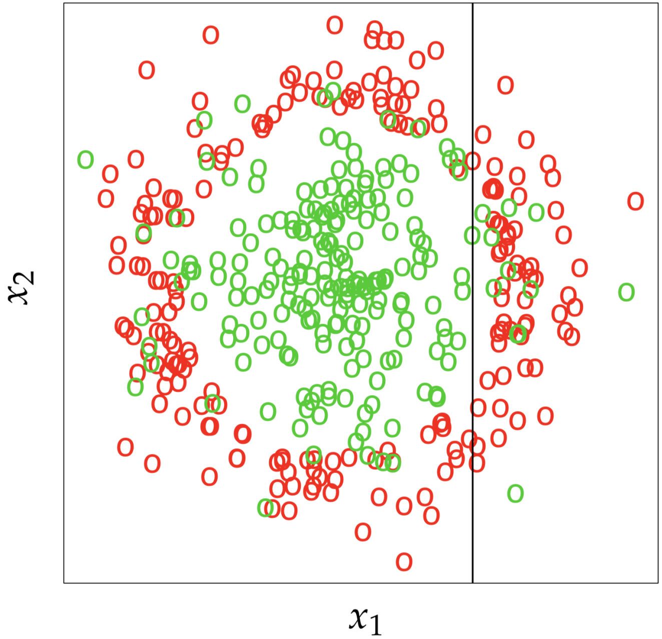
- Very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one.
- For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs.

Visualizing Classification Trees: Donut Data

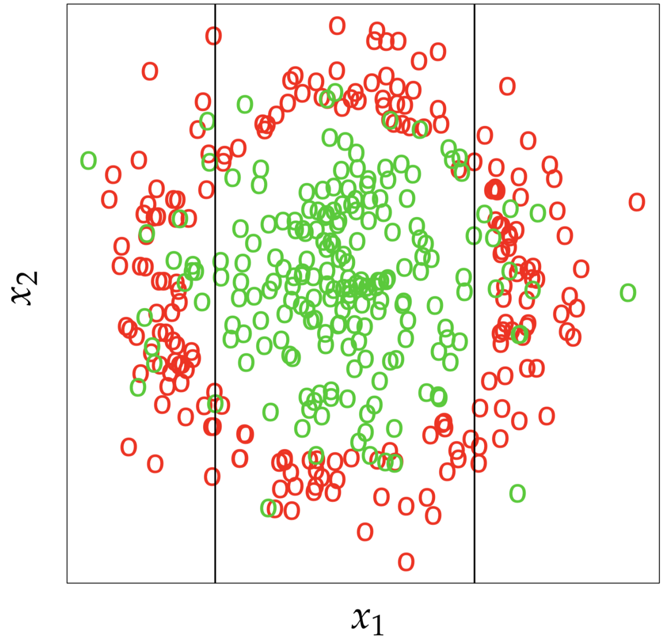


- Green class: two independent standard normal inputs X1, X2
- Red class: conditional on $X_1^2 + X_2^2 \geq 4.6$

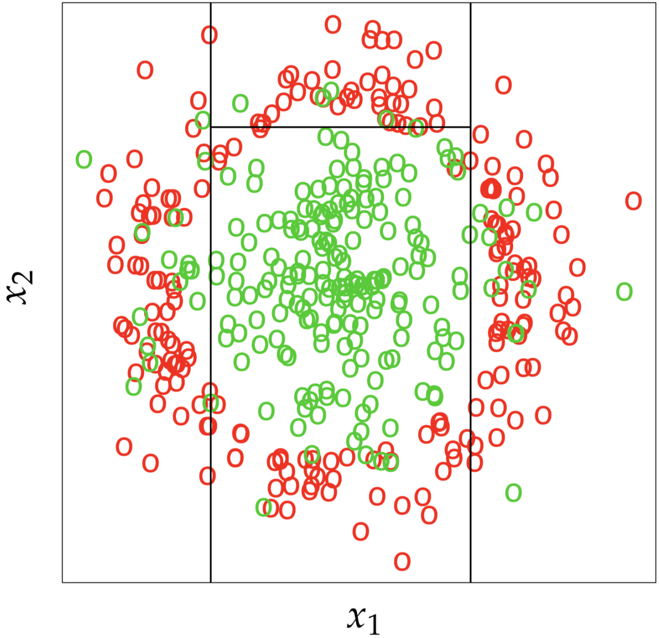
Classification tree: split 1



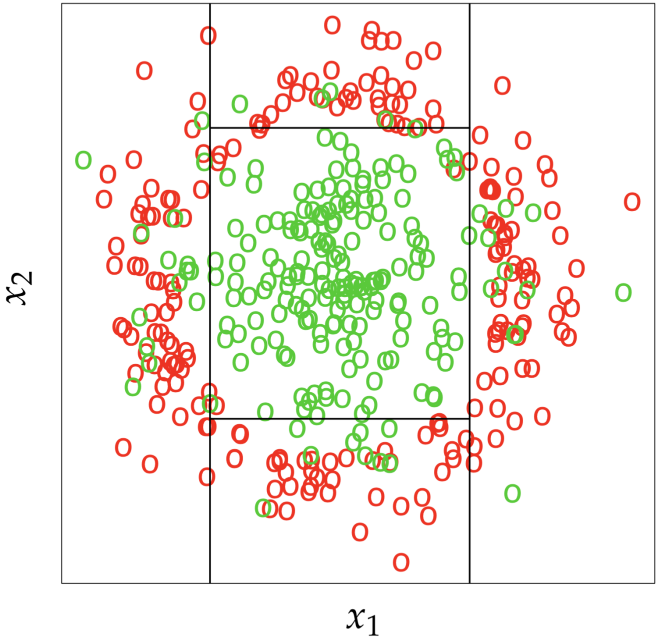
Classification tree: split 2



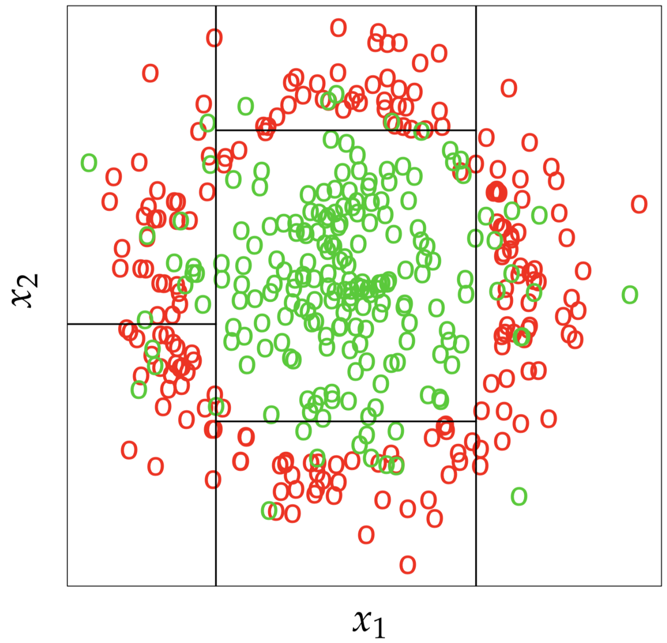
Classification tree: split 3



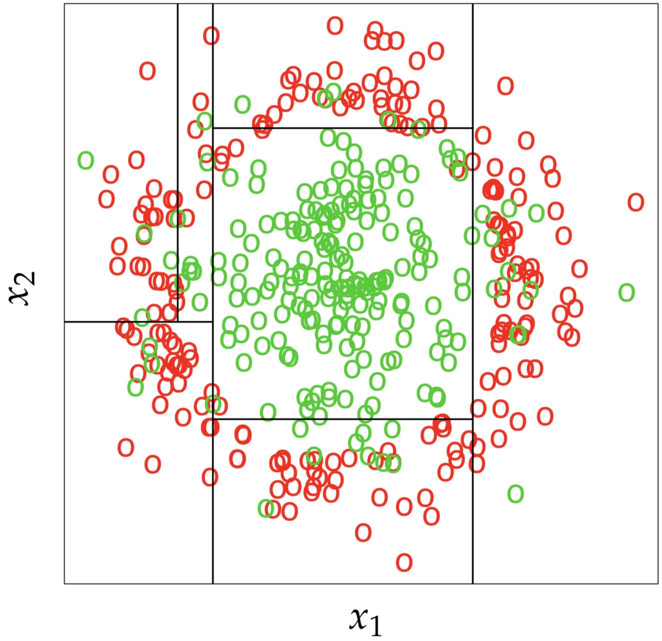
Classification tree: split 4



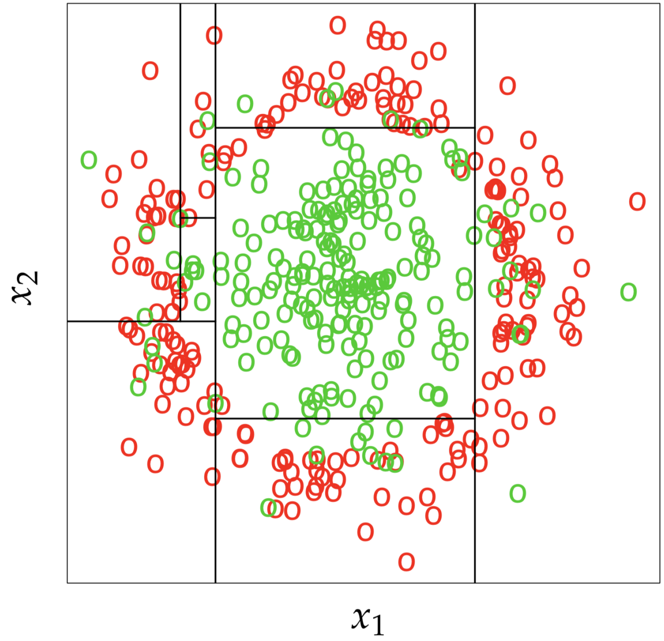
Classification tree: split 5



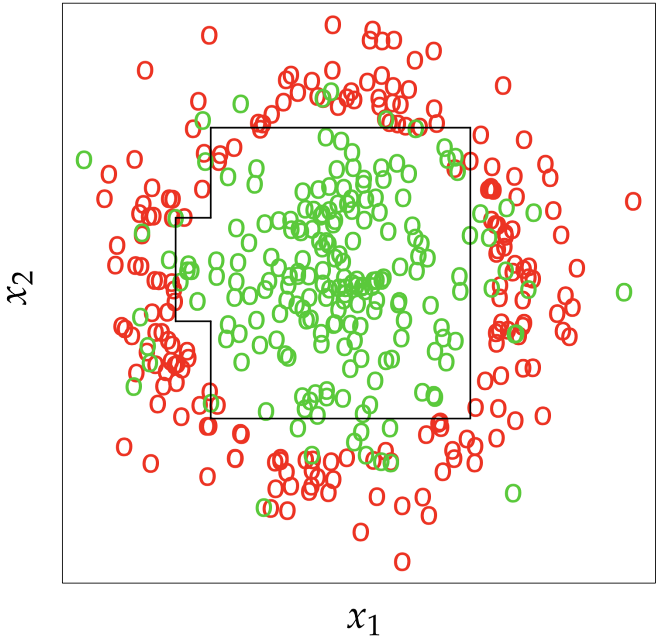
Classification tree: split 6



Classification tree: split 7



Classification tree: final decision boundary



Details of classification trees

- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- In the classification setting, RSS cannot be used as a criterion for making the binary splits
- A natural alternative to RSS is the **classification error rate**. this is simply the fraction of the training observations in that region that do not belong to the most common class: $E = 1 - \max_k(\hat{p}_{mk})$. Here $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class.
- However classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.

Gini index and Deviance

- The **Gini index** is defined by

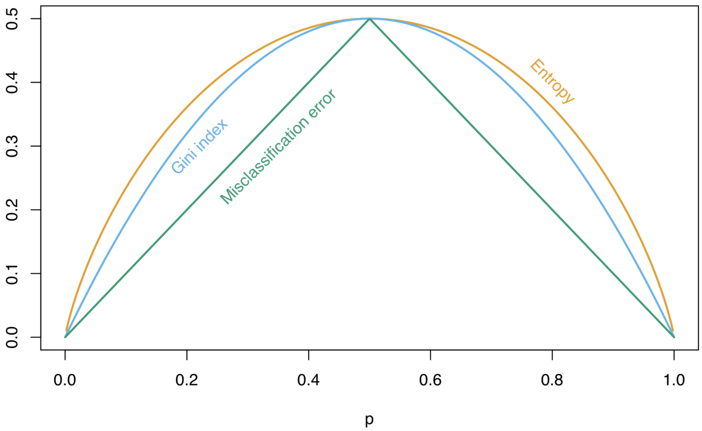
$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

  a measure of total variance across the K classes. Gini takes on a small value if all of the $\hat{p}_{mk}$'s are close to 0 or 1
- Gini represents the variance of a binomial distribution; the **diagonal of the multinomial variance covariance matrix** for > 2 classes
- For this reason the Gini index is referred to as a measure of **node purity** — a small value indicates that a node contains predominantly observations from a single class.
- Alternatively, **cross-entropy** or **deviance**, given by

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} log\, \hat{p}_{mk}$$

- It turns out that the Gini index and the cross-entropy are very similar numerically.

Node impurity measures

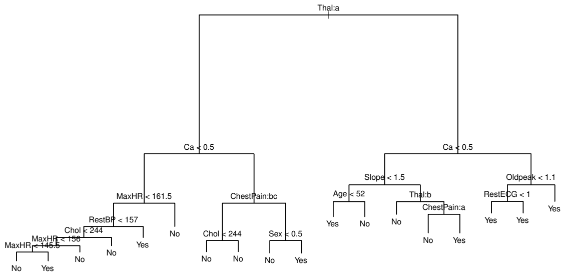


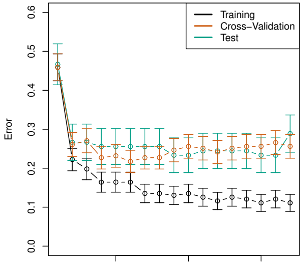
Node impurity measures for two-class classification, as a function of the proportion p in class 2. Cross-entropy has been scaled to pass through (0.5, 0.5).

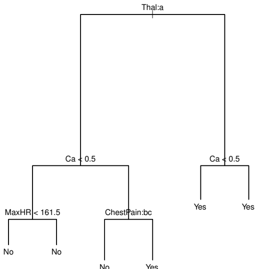
Example: heart data

- These data contain a binary outcome `HD` for 303 patients who presented with chest pain.
- An outcome value of `Yes` indicates the presence of heart disease based on an angiographic test, while `No` means no heart disease.
- There are 13 predictors including `Age`, `Sex`, `Chol` (a cholesterol measurement), and other heart and lung function measurements.
- Cross-validation yields a tree with six terminal nodes. See next figure.

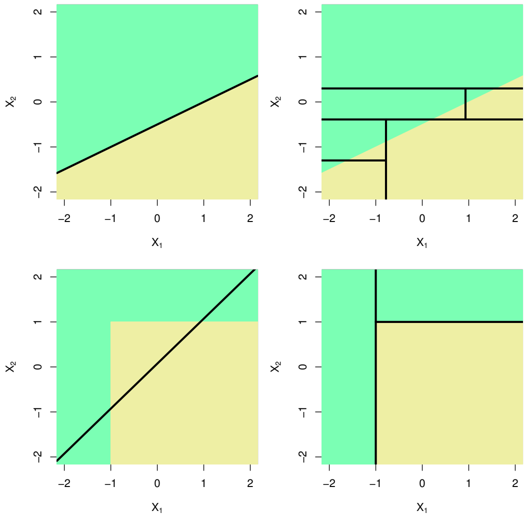
Example: heart data







Trees Versus Linear Models



Top Row: True linear boundary; Bottom row: true non-linear boundary.
Left column: linear model; Right column: tree-based model

Advantages of Trees

- Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- Trees can easily handle qualitative predictors without the need to create dummy variables.

Disadvantages of Trees

- Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches seen in this book.
- **High variance** in trees means a small change in the data can result in a very different series of splits; error in the top split is propagated down to all of the splits below it.
- **Bagging** and **Random Forests** reduce this variance by averaging many trees
- **Lack of smoothness** of the prediction surface. In classification with 0/1 loss, this doesn't hurt much, but can affect performance in the regression setting.
- By aggregating many decision trees, the predictive performance of trees can be substantially improved.