Lecture 6: Ensemble methods, Clustering

Part I: Pruning, Bagging, Boosting
Part II: Clustering

Prof. Alexandra Chouldechova
95-791: Data Mining

February 22, 2017

Agenda for Part I

- **Pruning trees**
- **Ensemble methods**
  - **Bagging**
  - **Random Forests**
  - **Boosting**

Reminder: Recursive binary partitioning

- At each step, you pick a new split by finding the input X_j and split point $\tilde{x}_j$ that best partitions the data
- In prediction, you choose splits to minimize the RSS
- In classification, choose splits to maximize node purity (minimize Gini index)

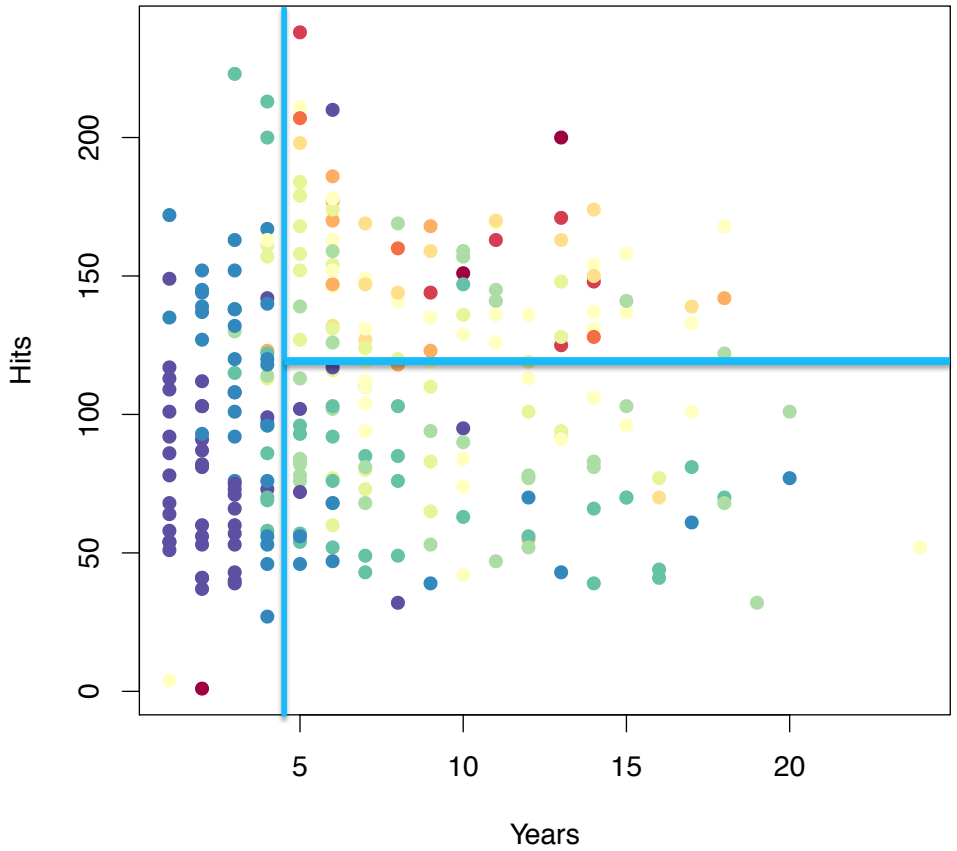
$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

where $\hat{p}_{mk}$ is the proportion of *training observations* in the mth region that are from the kth class

- G is small if all the $\hat{p}_{mk}$ are close to 0 or 1

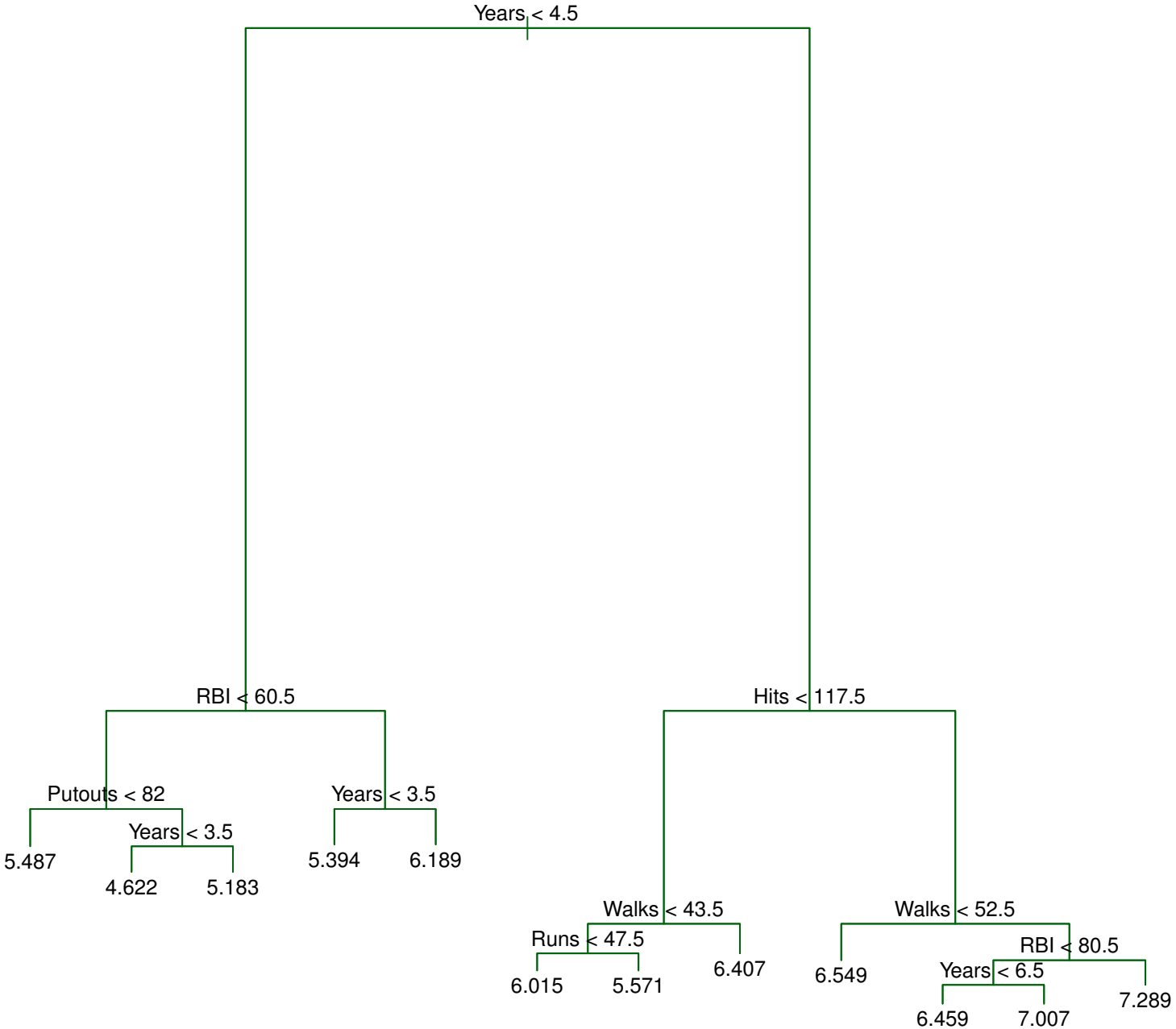
Tree pruning

Why did we stop here? Why not keep partitioning?



Low salary (blue, Green)
High salary (orange, red)

We could just keep going...



Tree pruning

- If we just keep going, we're going to overfit the training data, and get poor test performance
- We could stop as soon as we can't find a split to reduce RSS or Gini index by at least some pre-specified amount
- But this strategy is short-sighted: A seemingly worthless split early on might be followed by a really good split later
- Solution: Grow a very large tree T_0, and then prune it back

Cost complexity pruning

- Here's the regression tree version of cost complexity pruning aka weakest link pruning

- For each α, find the subtree $T \subset T_0$ that minimizes

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

  where $|T|$ is the number of terminal nodes (leaves) in tree T, and R_m is the rectangle corresponding ot the mth terminal node. $\hat{y}_{R_m}$ is just the mean of the training observations in R_m

- This is familiar. It has the form:

$$RSS(T) + \alpha |T|$$

  model error + a penalty on model complexity

Cost complexity pruning

For each α, find the subtree $T \subset T_0$ that minimizes

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

- How do we pick α?
- Use Cross-validation

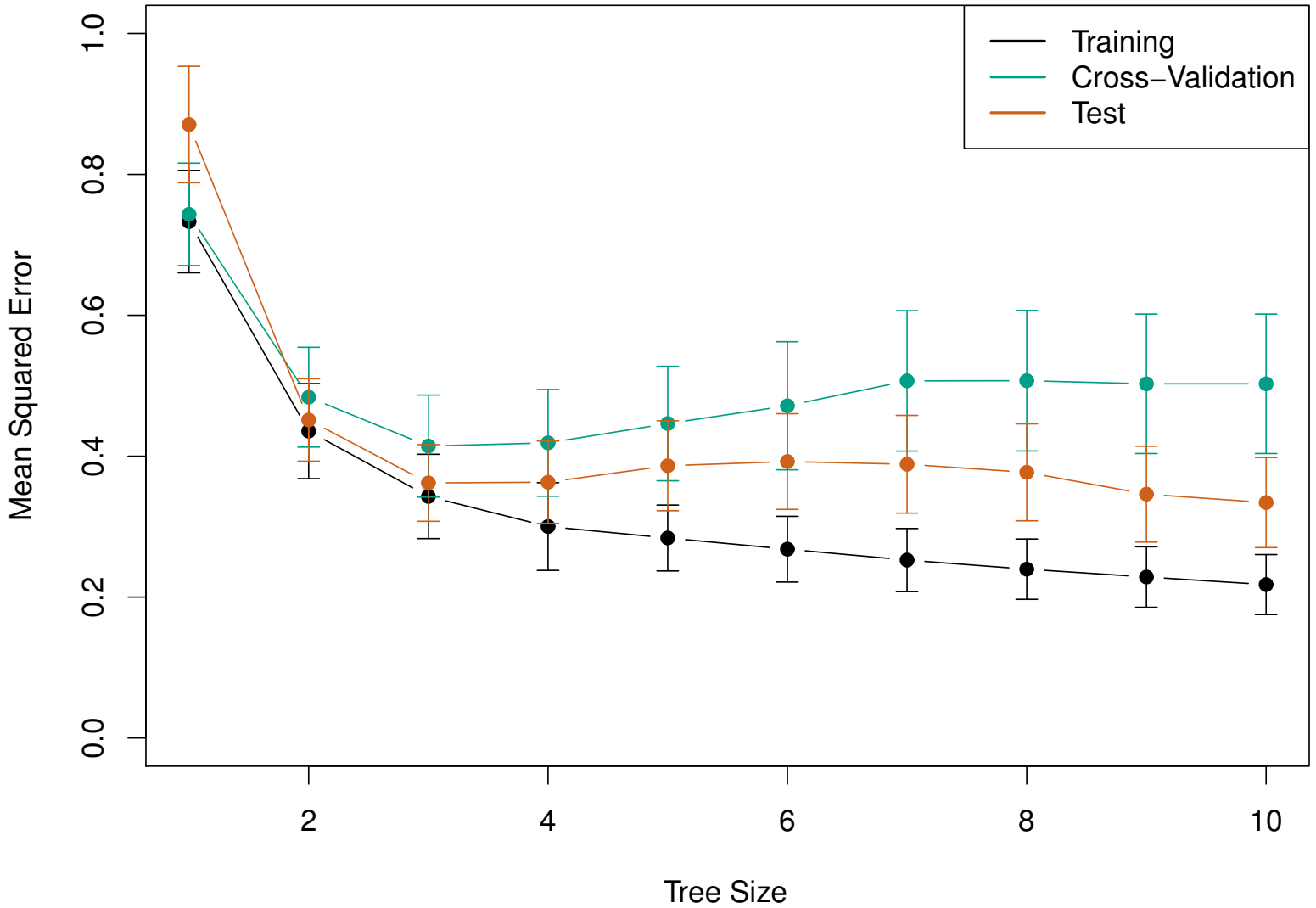
Pruning details

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 $k = 1, \ldots, K$:
   - 3.1 Repeat Steps 1 and 2 on the $\frac{K-1}{K}$th fraction of the training data, excluding the kth fold.
   - 3.2 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.

[source: ISL Chapter 8 slides]

Tree pruning



Looks like the small 3-leaf tree has the lowest CV error.

Good things about trees

- Trees are the most easily interpretable method we've talked about in this class
  - You can explain a decision tree to even your least technical colleague
- Arguably, trees more closely *mirror human decision-making*
- Trees are easy to display graphically.
  - You can print off your tree and easily obtain predictions by hand
- Trees can handle qualitative predictors and ordered qualitative predictors without needing to create dummy variables
- Trees handle missing values very nicely, without throwing out observations
  - When a value is missing, they split on a surrogate variable: E.g., if a user's *years of job experience* is missing, they'll split on an optimally chosen correlated variable like *age*.

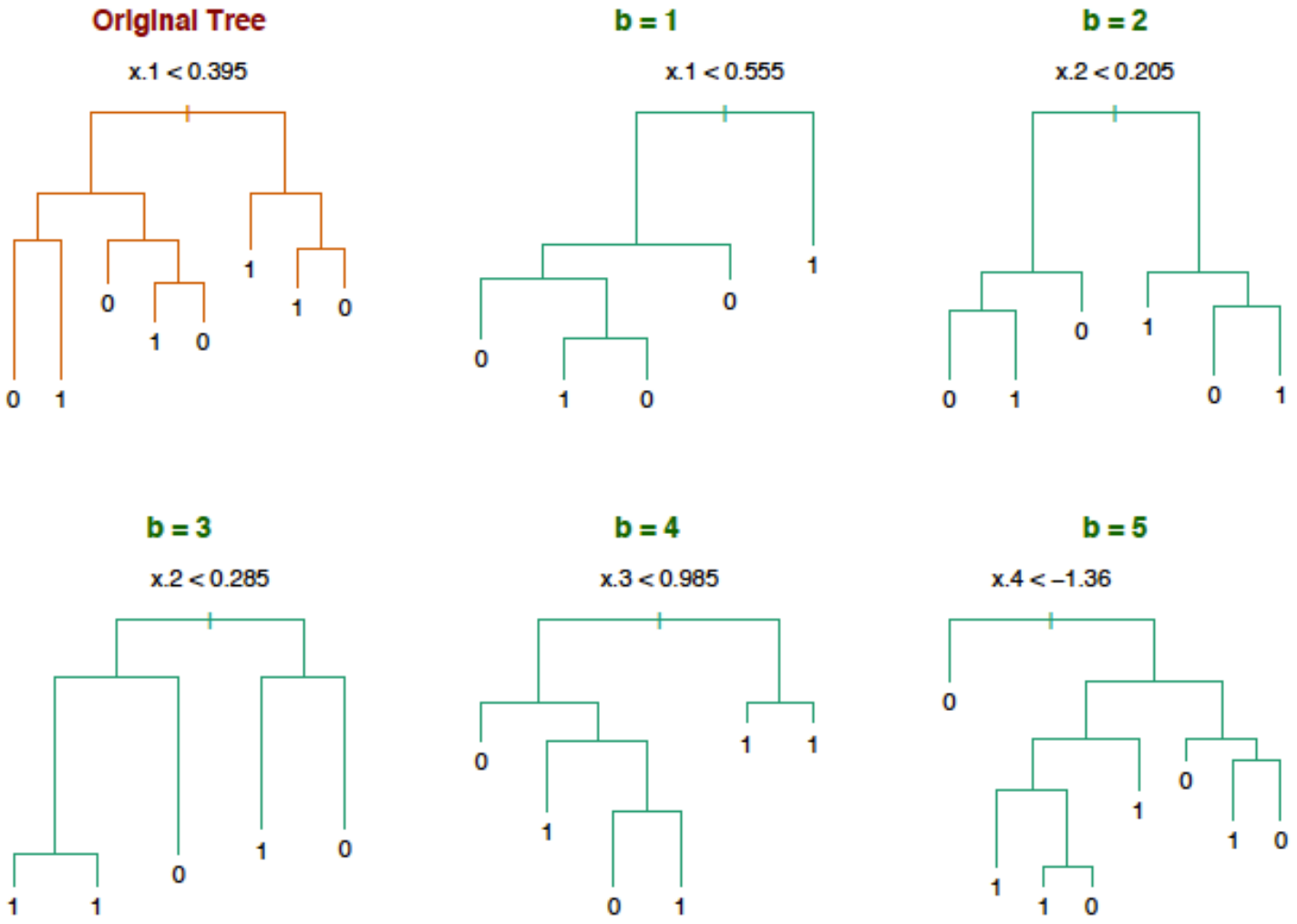
A summary of our methods so far

Method	Interpretable	Flexible	Makes assumptions?
Logistic regression	Yes	Extensible	Yes
k-NN	No	Highly	No
LDA/QDA	Sometimes	No	Yes
Trees	Extremely	Somewhat	No

- Decision trees are perhaps the most Interpretable method we've seen so far
- Trees don't assume any particular relationship between the response Y and the inputs X_j, and large trees are quite flexible
- So what's the catch?
- Turns out, Trees tend to be rather poor predictors/classifiers!
- We can fix this, if we're willing to give up Interpretability

Why are Decision trees poor predictors?

- Decision trees tend to have high variance. A small change in the training data can produce big changes in the estimated Tree.



How are we going to fix this?

- Let's think back to Cross-Validation, and why it gives much better results than the Validation Set Approach

- The Validation Set Approach tends to overestimate the error, but it also gives highly variable estimates
  - If you pick a different random split, you can get *wildly* different estimates of Test error

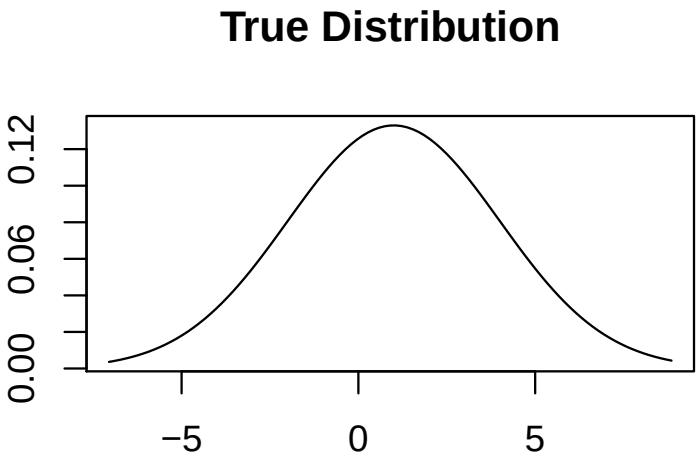
- The K-fold Cross-validation produces much more stable error estimates by averaging over K separate estimates of error (one from each fold).

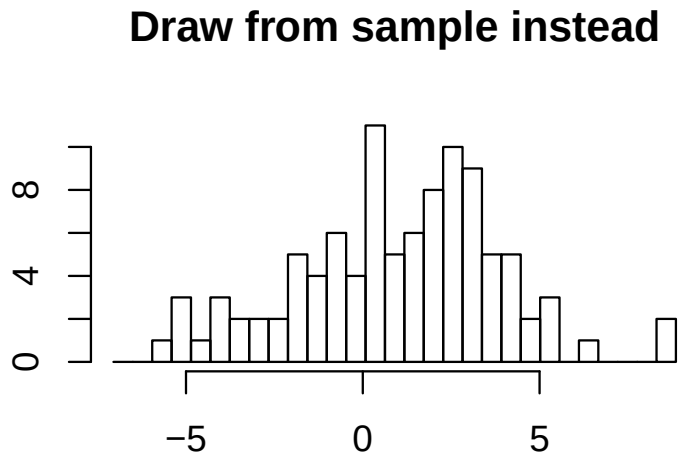
- The idea of Bagging (Bootstrap AGGregatING) has a similar motivation: To decrease the variance of a high-variance predictor, we can average across a bunch of estimates

The Bootstrap

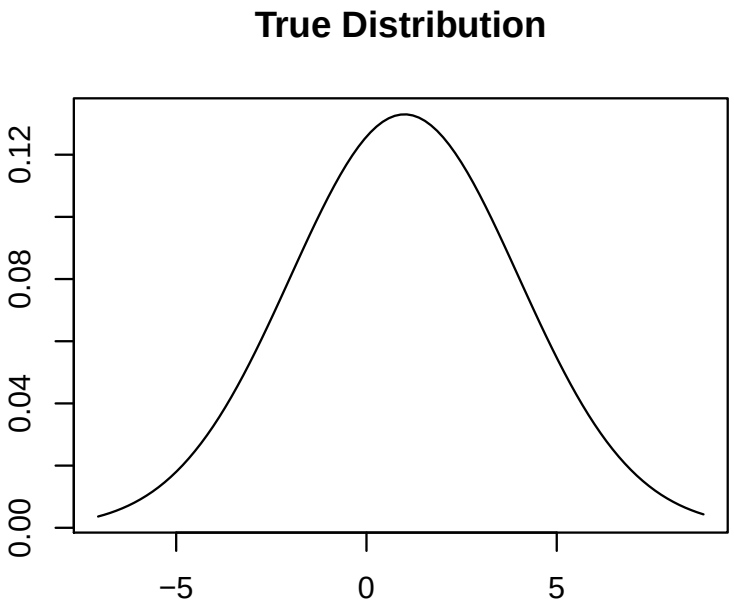
- The Bootstrap[1] is a fundamental resampling tool in statistics.
- Basic idea: We're going to create resampled data sets of size n by sampling from our observed data with replacement
- More formally this idea says: We're going to use the empirical distribution of our data to estimate the true unknown data-generating distribution

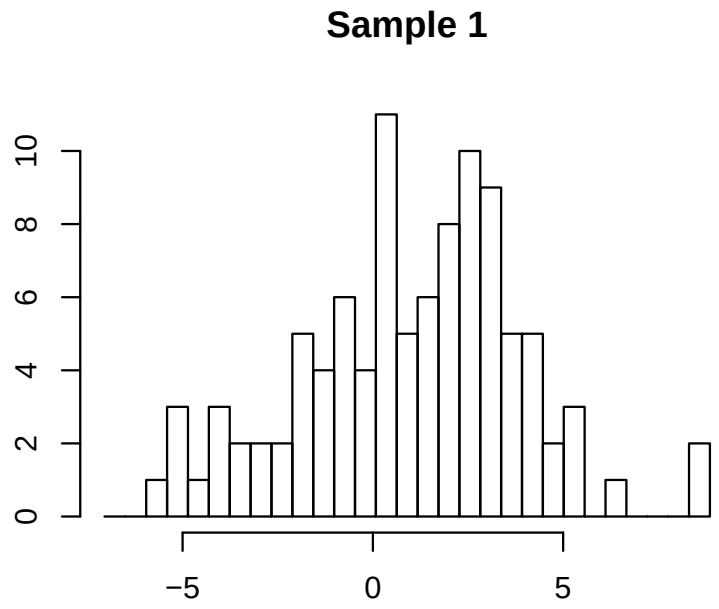
[1] Efron (1979), "Bootstrap Methods: Another Look at the Jackknife"

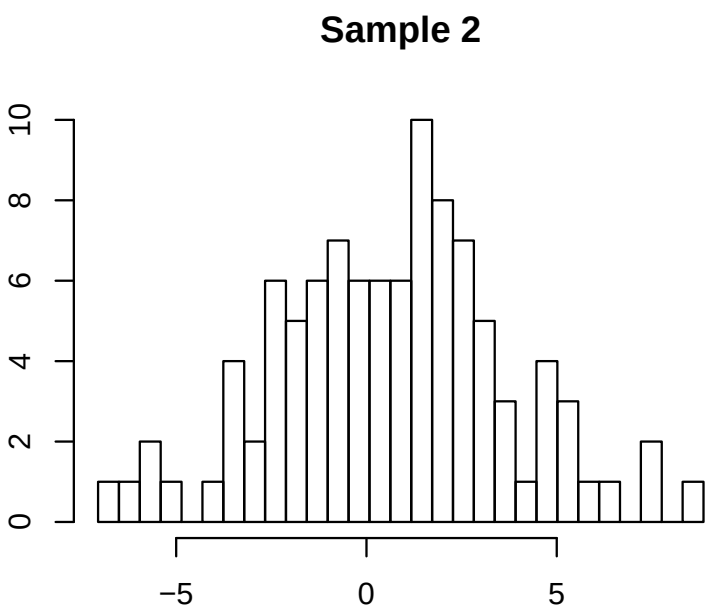


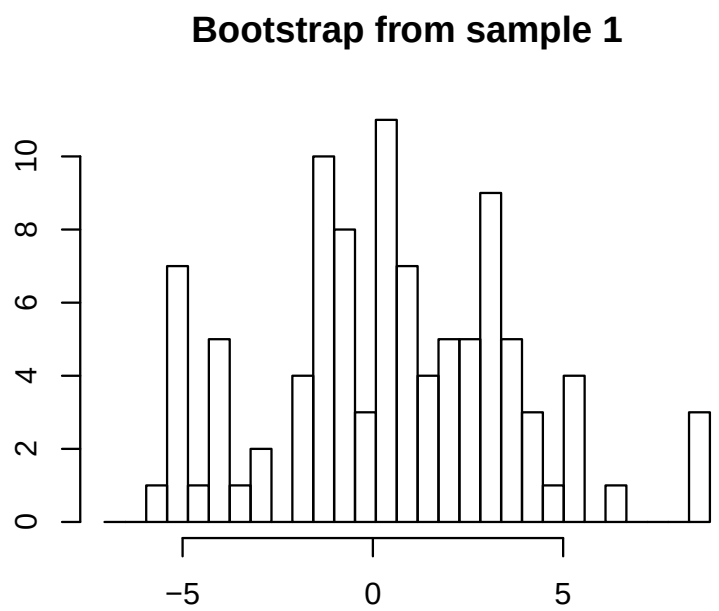


- We'd love to be able to generate more data from the true distribution. This would solve all of our problems.
- But we can't do that. We only get to see a sample of size n
- So we'll approximate sampling from the true distribution by re-sampling from our observed data instead.
  - A bootstrap sample of size n is a data set (x_i^*, y_i^*) $i = 1, \ldots, n$ where each (x_i^*, y_i^*) are sampled uniformly at random with replacement from our observed data $(x_1, y_1), \ldots, (x_n, y_n)$
  - We're (re-)sampling rows of our data, with replacement.


True Distribution
0.00
0.04
0.08
0.12
−5
0
5


Sample 1
0
2
4
6
8
10
−5
0
5


Sample 2
0
2
4
6
8
10
−5
0
5


Bootstrap from sample 1
0
2
4
6
8
10
−5
0
5

What shows up?

- Not all of the training points will appear in each sample

- Each bootstrap sample contains roughly 63.2% of the observed data points
  - The points that randomly get left out points feel like a validation set…we'll return to this later

- If we bootstrap sample B times, we get B data sets of size n, and we can estimate whatever we want on each dataset

Bagging: Classification trees

- Given a training data (x_i, y_i), $i = 1, \ldots n$, bagging[2] averages the predictions from classification trees over a collection of bootstrap samples.
- Here we'll describe how to apply Bagging to Classification Trees
  1. For $b = 1, \ldots, B$, get a bootstrap sample of size n from the training data: (x_i^{*b}, y_i^{*b}), $i = 1, \ldots n$
  2. Fit a classification tree $\hat{f}^{\text{tree},b}$ on each sample
  3. Classify a new point x_0 by taking the plurality vote across all B bootstrapped trees:

$$\hat{y}_0^{\text{bag}} = \underset{k=1,\ldots,K}{\text{argmax}} \sum_{b=1}^{B} I\left(\hat{f}^{\text{tree},b(x_0)} = k\right)$$

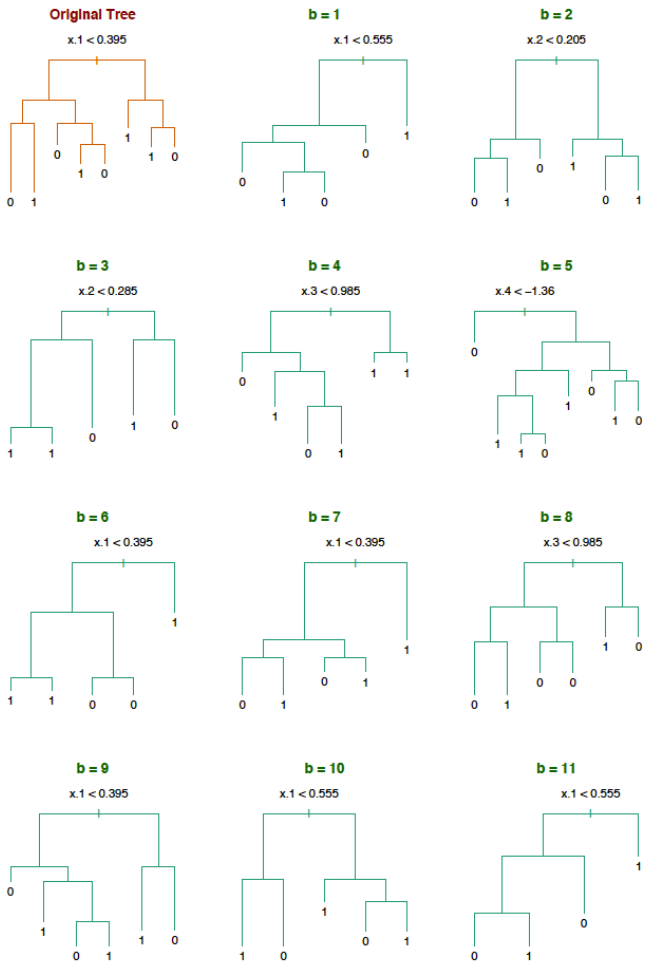
- Step (3) amounts to letting each of the B trees *vote*, and then choosing whichever class has the most votes
- Typically, in Step (2) the trees are grown very large, with no pruning.

  Why are we less worried about tuning each tree?

[2] Breiman (1996), "Bagging Predictors"

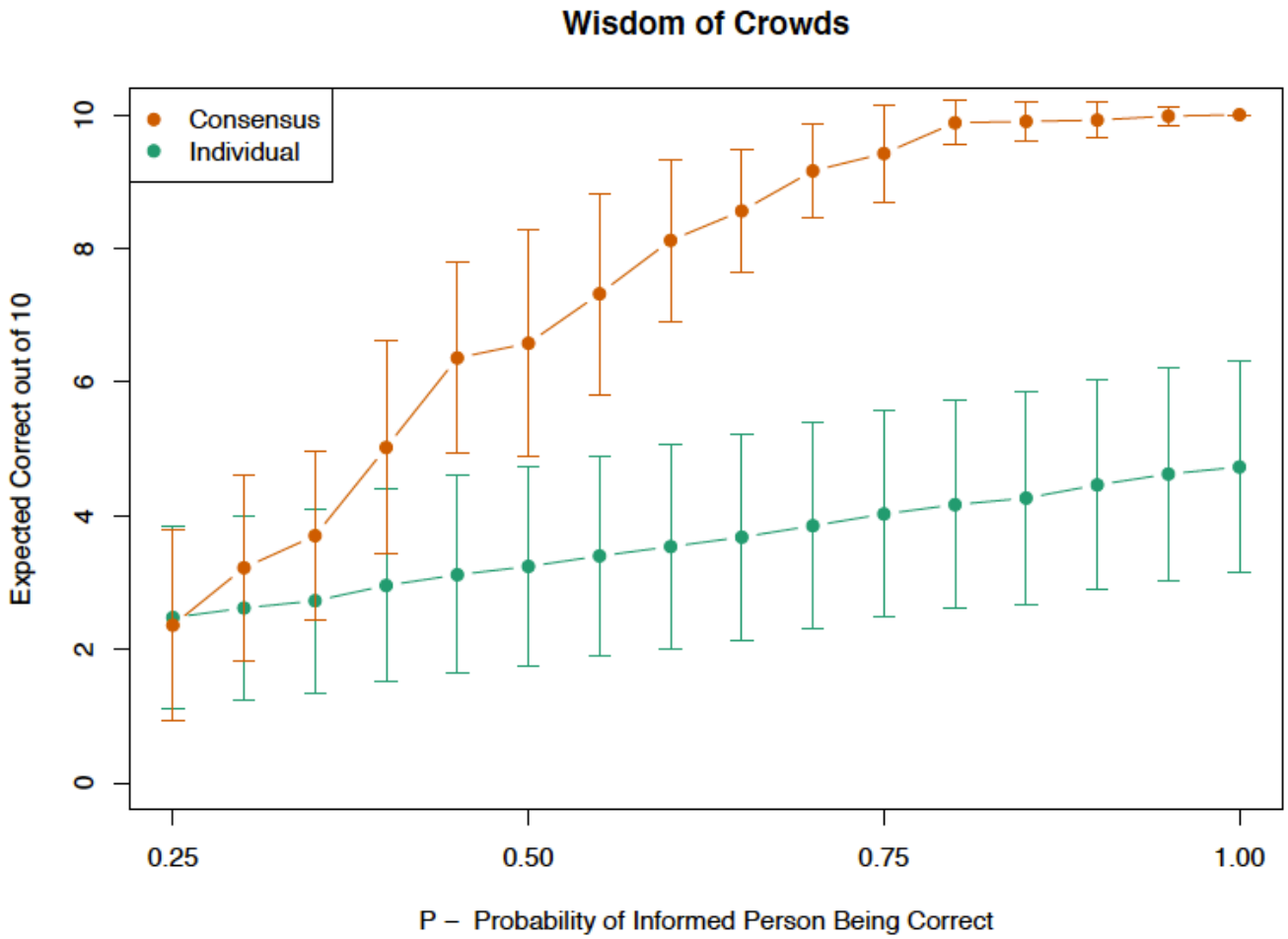
Example: bagging

Example (from ESL 8.7.1): $n = 30$ training data points, $p = 5$ features, and $K = 2$ classes. No pruning used in growing trees:



How could this possibly work?

- You may have heard of the Wisdom of crowds phenomenon
- It's a concept popularized outside of statistics to describe the idea that the collection of knowledge of a group of independent people can exceed the knowledge of any one person individually.
- Interesting example (from ESL page 287): Academy award predictions
  - 50 people are asked to predict academy award winners for 10 categories
  - Each category has 4 nominees
  - For each category, just 15 of the 50 voters are at all informed (the remaining 35 voters are guessing randomly)
  - The 15 informed voters have probability $P \geq 0.25$ of correctly guessing the winner



There are 10 award categories and 4 nominees in each. For each of the 10 categories, there are 15 (of 50) voters who are *informed*. Their probability of guessing correctly is $P \geq 0.25$. Everyone else guesses randomly.

Example: Breiman's bagging

Example from the original Breiman paper on bagging: comparing the misclassification error of the CART tree (pruning performed by cross-validation) and of the bagging classifier (with $B = 50$):

Data Set	$\bar{e}_S$	$\bar{e}_B$	**Decrease**
waveform	29.1	19.3	34%
heart	4.9	2.8	43%
breast cancer	5.9	3.7	37%
ionosphere	11.2	7.9	29%
diabetes	25.3	23.9	6%
glass	30.4	23.6	22%
soybean	8.6	6.8	21%

Voting probabilities are not estimated class probabilities

- Suppose that we wanted *probability estimates* $\hat{p}_k(x)$ out of our bagging procedure.
- What if we tried using:

$$\hat{p}_k^{\text{vote}}(x) = \frac{1}{B}\sum_{b=1}^{B}\left(\hat{f}^{\text{tree},b}(x) = k\right)$$

  This is the proportion of bootstrapped trees that voted for class k.
- This can be a bad idea
- Suppose we have two classes, and the true probability that $y_0 = 1$ when $X = x_0$ is 0.75.
- Suppose each of the bagged trees estimates the probability function quite well, resulting in each tree producing the correct classification of x_0 to class 1: $\hat{f}^{\text{tree},b}(x_0) = 1$
- Then $\hat{p}_1^{\text{vote}}(x) = 1$... that's wrong
- What if we used each tree's estimated probabilities instead?

Alternative form: Probability Bagging

- Instead of just looking at the class predicted by each tree, look at the *predicted class probabilities* $\hat{p}_k^{\text{tree},b}(x)$
- Define the bagging estimate of class probabilities:

$$\hat{p}_k^{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{p}_k^{\text{tree},b}(x) \quad k = 1, \ldots K$$

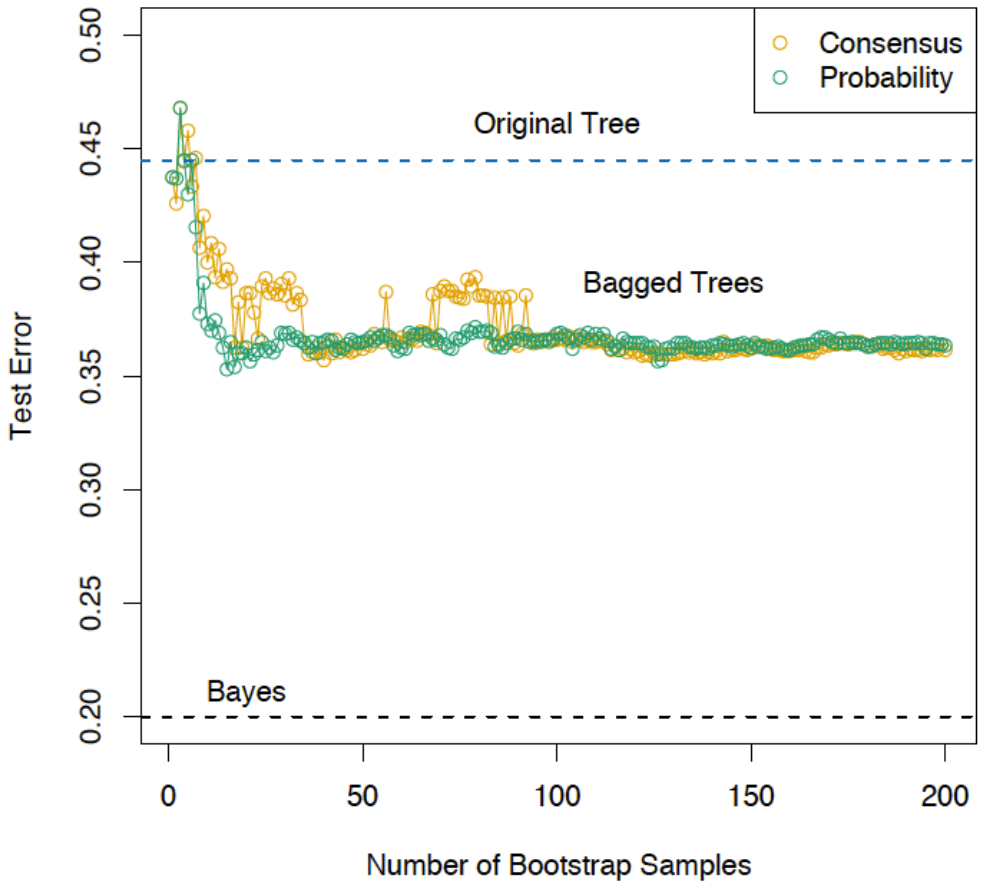
- We can use $\hat{p}_k^{\text{bag}}(x)$ itself as an *alternative* to plurality voting of the trees.
- Given an input vector x_0, we can classify it according to

$$\hat{y}_0^{\text{bag}} = \underset{k=1,\ldots K}{\text{argmax}}\ \hat{p}_k^{\text{bag}}(x)$$

- This form of bagging is preferred if we want to estimate class probabilities, and it may improve overall classification accuracy

Comparison of the two bagging approaches

The probability form of bagging produces misclassification errors shown in green. The Consensus version is what we first introduced. It's not as well behaved.



The Test error eventually stops decreasing past a certain value of B because we hit the limit in the variance reduction bagging can provide

Out-of-Bag (OOB) Error Estimation

- Recall, each bootstrap sample contains roughly 2/3 ($\approx 63.2\%$) of the of the training observations

- The remaining observations not used to fit a given bagged tree are called the out-of-bag (OOB) observations

- Another way of thinking about it: Each observation is OOB for roughly $B/3$ of the trees. We can treat observation i as a test point each time it is OOB.

- To form the OOB estimate of test error:
  - Predict the response for the ith observation using each of the trees for which i was OOB. This gives us roughly $B/3$ predictions for each observation.
  - Aggregate predictions into a single prediction for observation i and calculate the error of this aggregated prediction
  - Average all of the errors

Random Forests

- Random forests provide an improvement over bagged trees by incorporating a small tweak that decorrelates the individual trees
  - This further reduces variance when we average the trees

- We still build each tree on a bootstrapped training sample

- But now, each time a split in a tree is considered, the tree may only split on a predictor from a randomly selected subset of m predictors

- A fresh selection of m randomly selected predictors is presented at each split... not for each tree, but for each split of each tree

- $m \approx \sqrt{p}$ turns out to be a good choice
  - E.g., if we have 100 predictors, each split will be allowed to choose from among 10 randomly selected predictors

Bagging vs. Random Forests

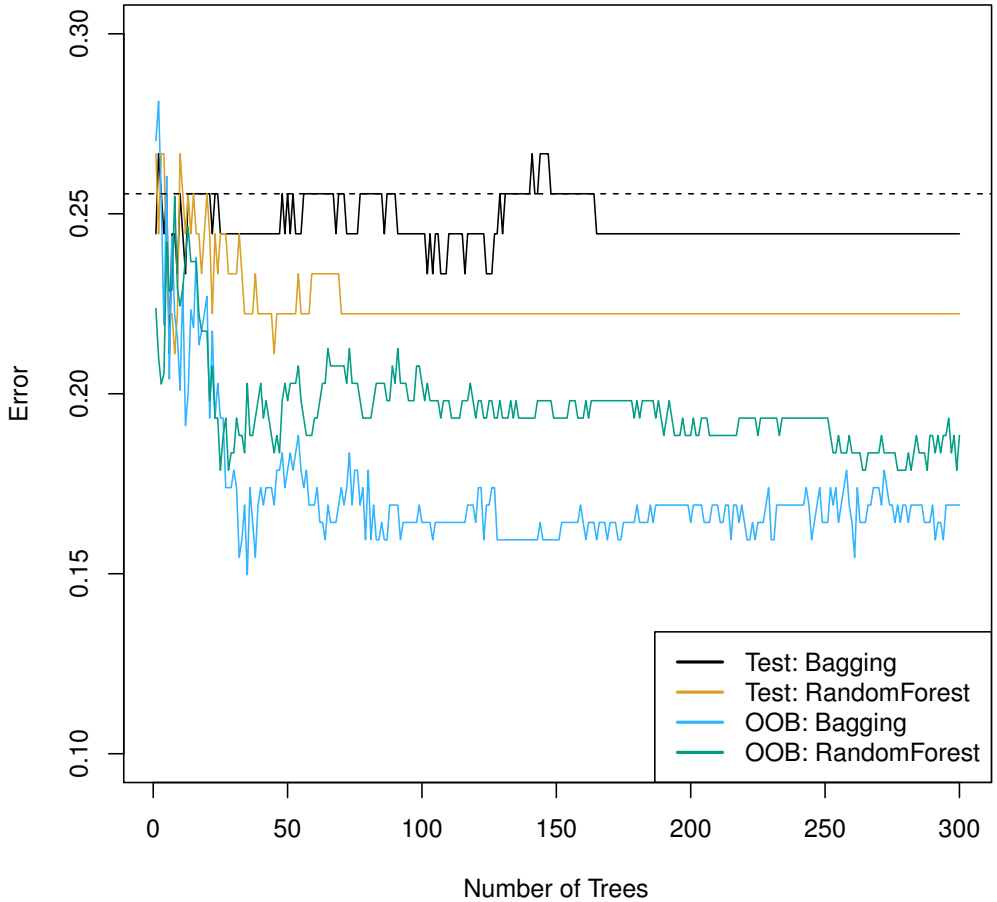


Figure 8.8 from ISL. Various fits to the `Heart` data

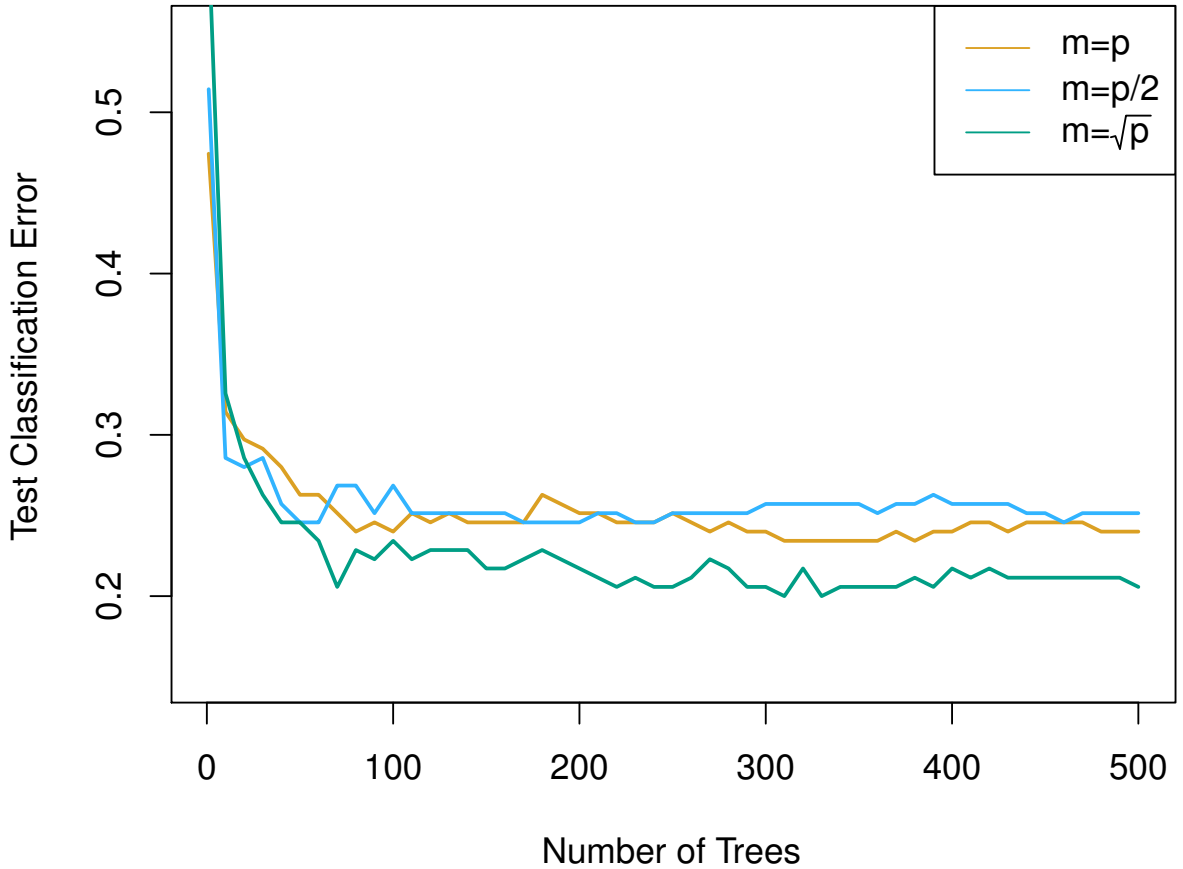
Dashed: Error from a single Classification tree

Random forest fit with $m = 4 \approx \sqrt{13} = \sqrt{p}$

A big data example: Gene expression data

- $p = 4{,}718$ genetic measurements from just 349 patients
- Each patient has a qualitative label. $K = 15$ possible labels
  - Either normal, or one of 14 types of cancer
- Split data into training and testing, fit Random forest to training set for 3 different choices of number of splitting variables, m.
- First, filter down to the 500 genes that have the highest overall variance in the training set

Test error: Gene expression data



- Curves show Test misclassification rates for a 15-class problem with $p = 500$ predictors and under 200 observations used for training
- x-axis gives number of trees (number of bootstrap samples used)
- $m = p$ corresponds to bagging.
- A single classification tree has an error rate of 45.7%.

Summary: Random forests

- Random forests have two tuning parameters:
  - $m =$ the number of predictors considered at each split
  - $B =$ the number of trees (number of bootstrapped samples)

- Increasing B helps decrease the overall variance of our estimator

- $m \approx \sqrt{p}$ is a popular choice

- Cross-validation can be used across a grid of (m, B) values to find the choice that gives the lowest CV estimate of test error

- Out-of-bag (OOB) error rates are commonly reported
  - Each observation is OOB for around $B/3$ of the trees
  - Can get a test error estimate for each observation each time it is OOB
  - Average over all the OOB errors to estimate overall test error

- RF's are parallelizable: You can distribute the computations across multiple processors and build all the trees *in parallel*

Random forests vs. Trees

- We liked trees because the model fit was very easy to understand
  - Large trees wind up hard to interpret, but small trees are highly interpretable

- With random forests, we're averaging over a bunch of bagged trees, and each tree is built by considering a small random subset of predictor variables at each split.

- This leads to a model that's *essentially* uninterpretable

- The Good: Random forests are very flexible and have a *somewhat justifiable* reputation for not overfitting
  - Clearly RF's can overfit: If we have 1 tree and consider all the variables at each split, $m = p$, we just have a single tree
  - If $m \ll p$ and the number of trees is large, RF's tend not to overfit
  - You should still use CV to get error estimates and for model tuning! Don't simply rely on the reputation RF's have for not overfitting.

Boosting

- We talked about Bagging in the context of bagged trees, but we can bag any predictor or classifier we want
- Boosting is yet another general approach that can be applied to any based learning method
- Here we'll briefly discuss Boosting decision trees
- Boosting is another way of taking a base learner (a model) and building up a more complex ensemble
- In bagging, we bootstrap multiple versions of the training data, fit a model to each sample, and then combine all of the estimates
  - The base learners used in bagging tend to have high variance, low bias
- Boosting builds up the ensemble sequentially: E.g., To boost trees, we grow small trees, one at a time, at each step trying to improve the model fit in places we've done poorly so far

Boosting algorithm: Regression trees

1. Set $\hat{f}(x) = 0$ and *residuals* $r_i = y_i$ for all i in the training set
2. For $b = 1, 2, \ldots, B$, repeat:
   1. Fit tree $\hat{f}^b$ with d splits to the training data (X, r)[3]
   2. Update $\hat{f}$ by adding *shrunken* version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$

   3. Update the residuals

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

- λ is called the shrinkage parameter. Typically, $\lambda \ll 1$

[3] We're treating the residual vector r as our outcome at each step.

What's boosting trying to do?

- Boosting works best if d (the size of each tree) is small
- Given the current model, we fit a decision tree to the *residuals* from the model
- This new tree helps us perform just a little bit better in places where the current model wasn't doing well
- Unlike bagging, where each tree is large and tries to model the entire (bootstrapped) data well, each tree in boosting tries to incrementally improve the existing model
- Think of boosting as learning slowly: We use small trees, try to make incremental improvements, and further slow down the learning process by incorporating the *shrinkage parameter* λ

Boosting for classification

- The basic idea is the same: Use weak base learners, update incrementally, shrink at each step
- Details are more complicated…too complicated to present here
- The **R** package `gbm` (gradient boosted models) handles both prediction (regression) and classification problems
- If you're interested in the details, see Chapter 10 of Elements of Statistical Learning

Gene expression example: RF vs Boosted trees

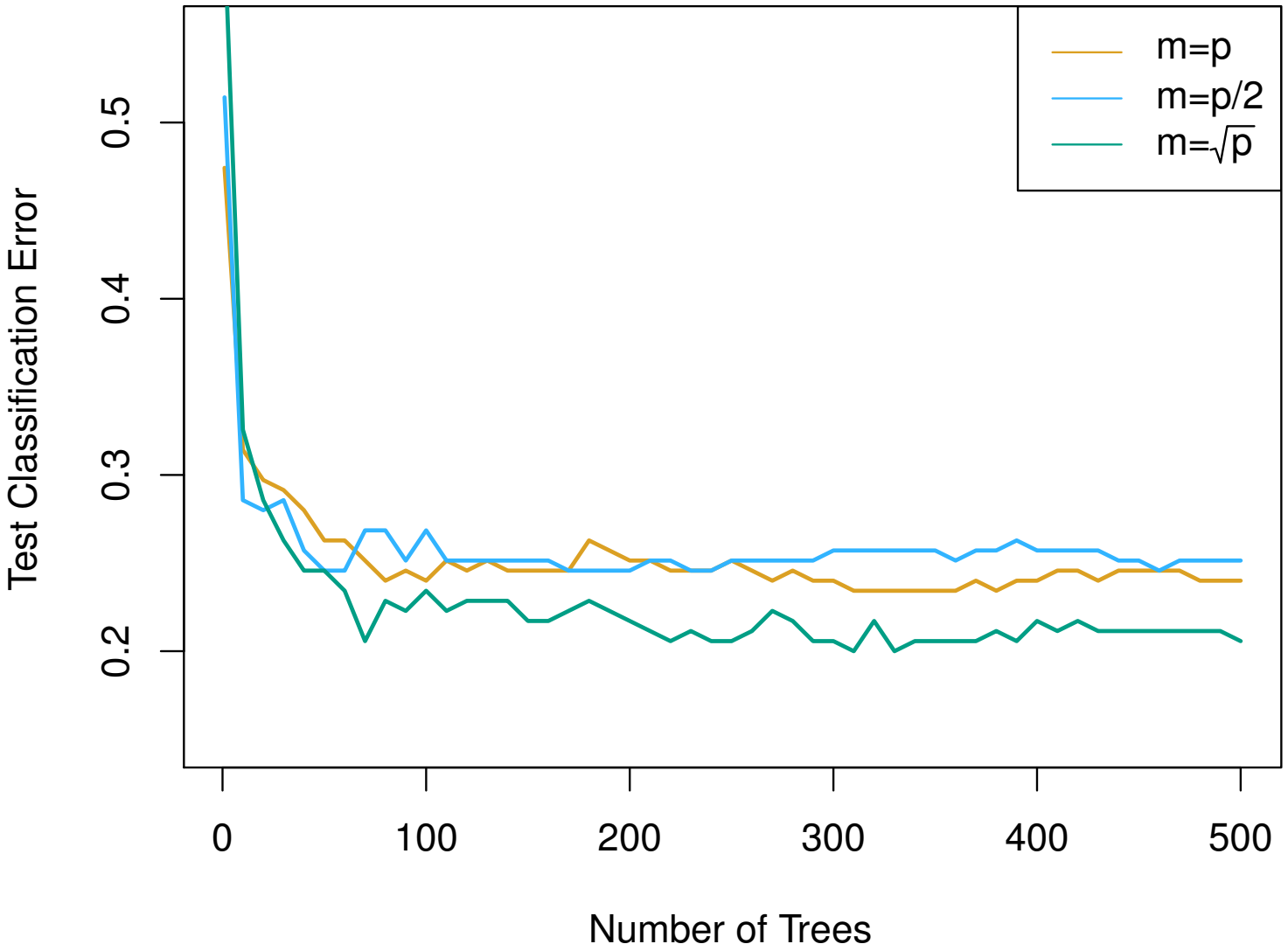


Fig 8.10 Random forests with different choices of m ($K = 15$ classes)

Gene expression example: RF vs Boosted trees

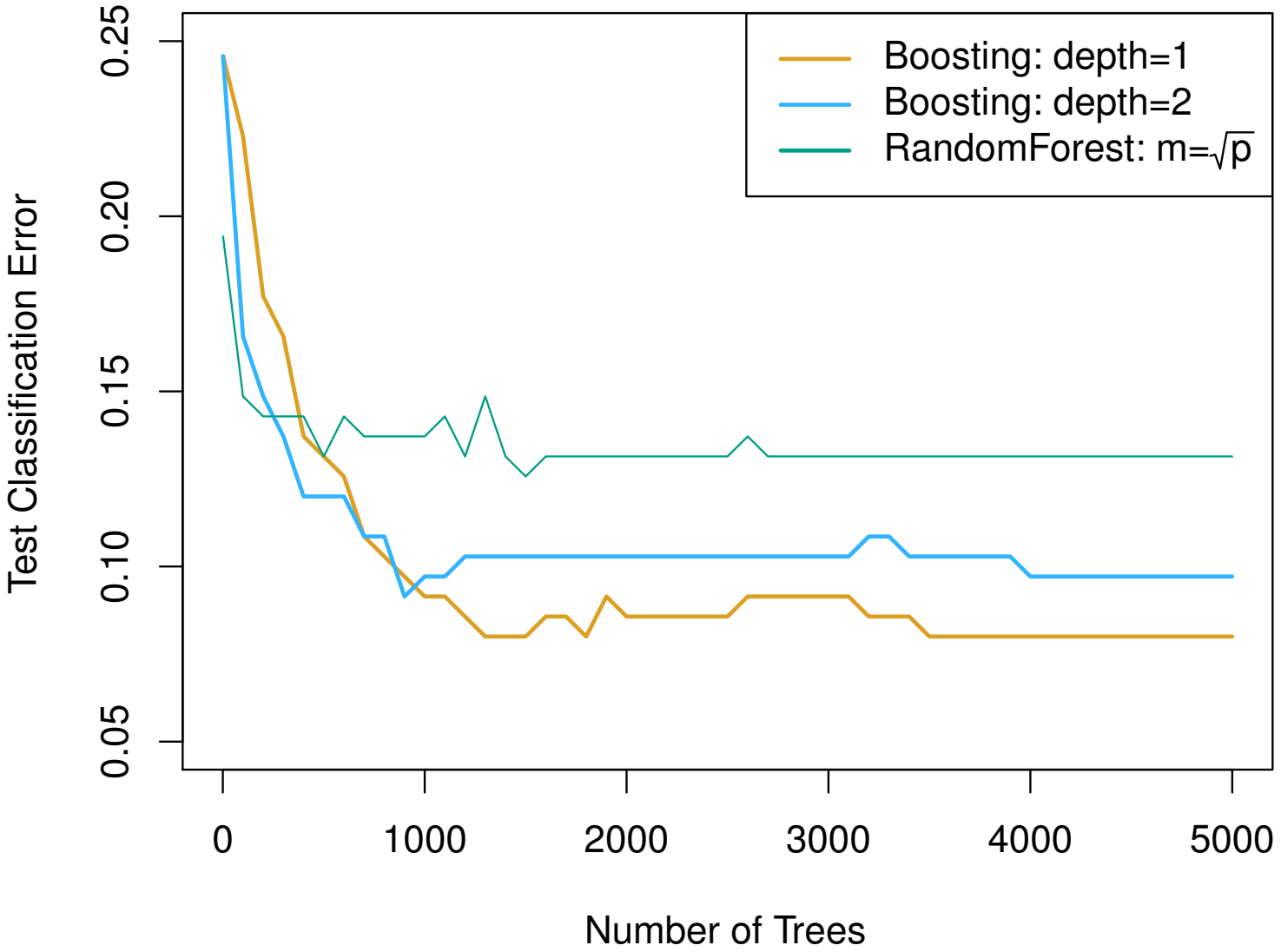
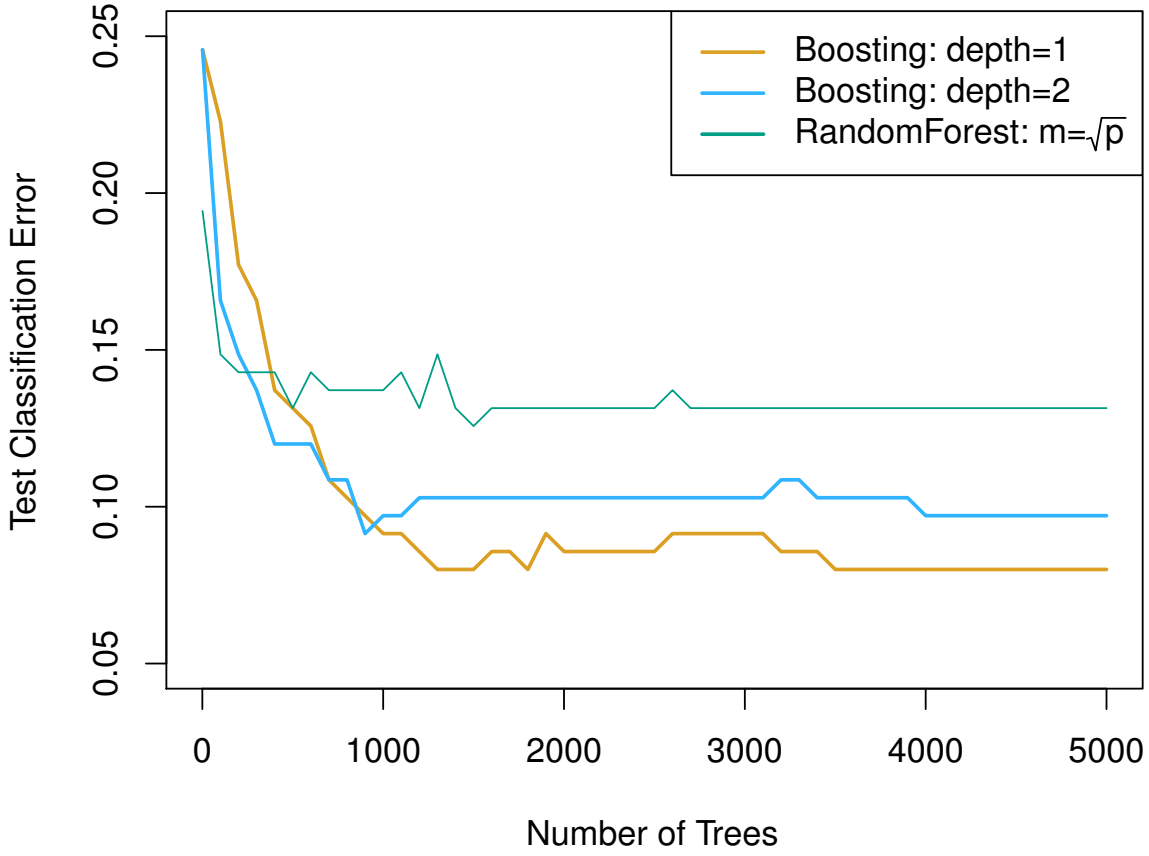


Fig 8.11 Random forests vs. Boosted trees ($K = 2$ classes)



- $K = 2$ class problem: *cancer* vs *non-cancer*
- Boosting with Depth-1 trees outperforms Depth-2 trees, and both outperform random forests…but standard errors are actually around 0.02, so differences aren't really statistically significant
- Boosting uses $\lambda = 0.01$

Tuning parameters for boosting

- Number of trees, B: Boosting can overfit if B is too large, though overfitting happens slowly. Use cross-validation to select

- shrinkage parameter, λ: $0 < \lambda \ll 1$. This is sometimes called the *learning rate*. Common choices are $\lambda = 0.01$ or $\lambda = 0.001$. Small λ requires very large B to achieve good performance.

- Number of splits, d: $d = 1$, called *stumps*, often works well. This amounts to an additive model.
  - Often refer to d as the interaction depth: d splits can involve at most d variables

Email spam data (seen on Homework 4)

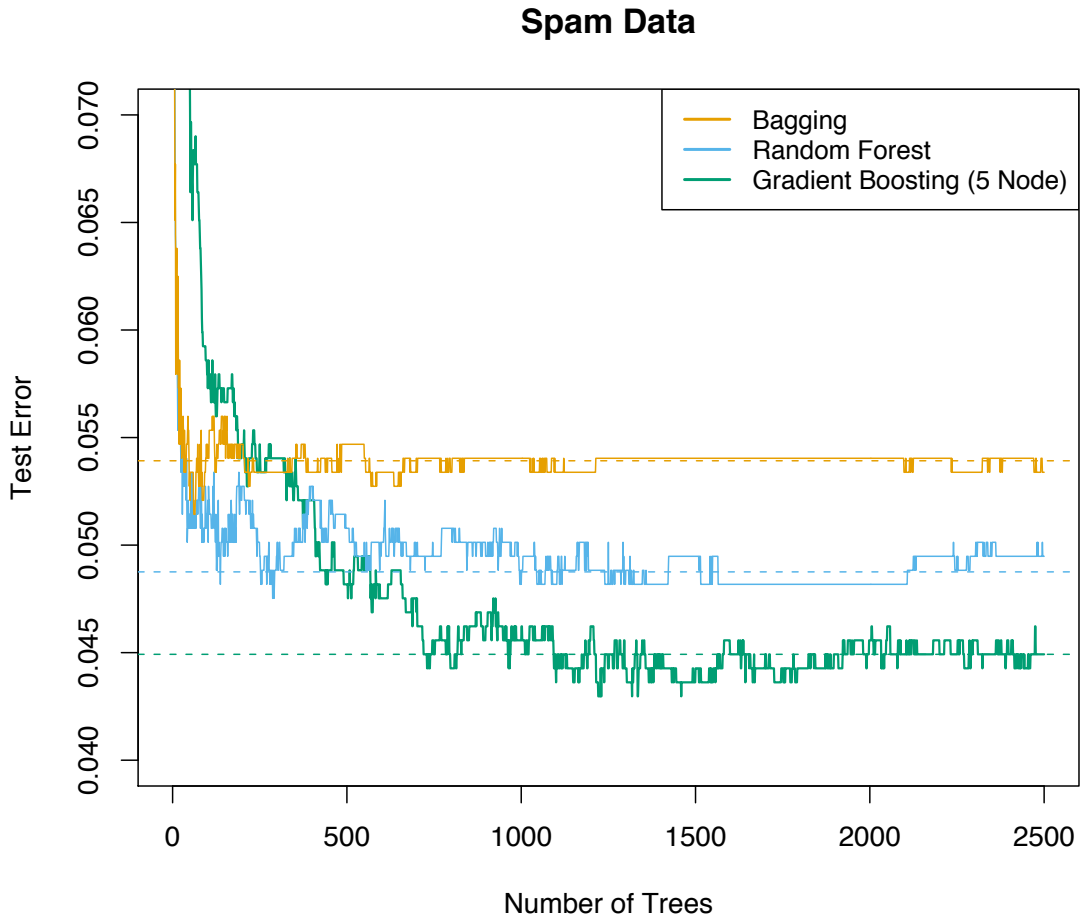


FIGURE 15.1. *Bagging, random forest, and gradient boosting, applied to the spam data. For boosting,* 5*-node trees were used, and the number of trees were chosen by* 10*-fold cross-validation (*2500 *trees). Each "step" in the figure corresponds to a change in a single misclassification (in a test set of* 1536*).*

Bagging, Boosting, Interactions

- Ensemble methods feel like *black boxes*: they make predictions by combining the results of hundreds of separate models
- Such models are able to capture complex interactions between predictors, which additive models are unable to do
- E.g., Suppose that your most profitable customers are young women and older men. A linear model would say:

$$\texttt{profit} \approx \beta_0 + \beta_1 I(\texttt{female}) + \beta_2 \texttt{age}$$

- This doesn't capture the interaction between `age` and `gender`
- Trees (and ensembles of trees) do a great job of capturing interactions
- Indeed, a tree with d splits can capture up to d-way interactions

Variable Importance

- While RF's and Boosted trees aren't interpretable in any meaningful sense, we can still extract some insight from them

- For instance, we can use variable importance plots to help answer the question: Which inputs have the biggest effect on model fit?

- There are two popular ways of measuring variable importance.

- Approach 1: For regression (resp. classification) record the total amount that the RSS (resp. Gini index) is decreased due to splits over a given predictor. Average this over all B trees.
  - A large value indicates an important predictor

- Approach 2: Randomly permute the values of the jth predictor, and measure how much this reduces the performance of your model (e.g., how much it increases MSE or accuracy)
  - A large drop in performance indicates an important predictor

- The `varImpPlot()` function in **R** calculates these quantities for you

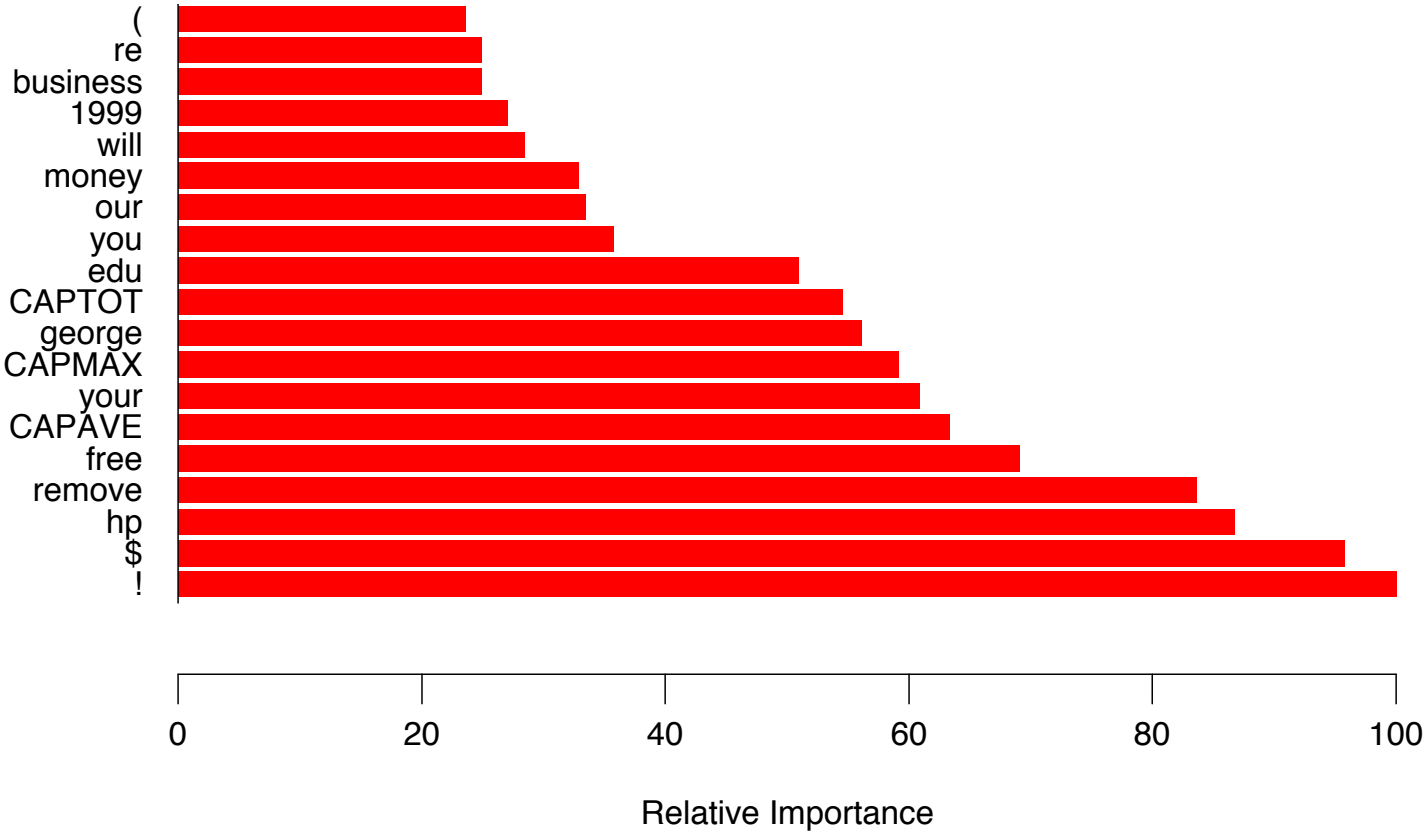


FIGURE 10.6. *Predictor variable importance spectrum for the* `spam` *data. The variable names are written on the vertical axis.*

- This helps us to see which variables are the most important…but it doesn't tell us how they affect the model. E.g., the frequency of ! is very important, but are emails with lots of !'s more likely to be spam or not spam?

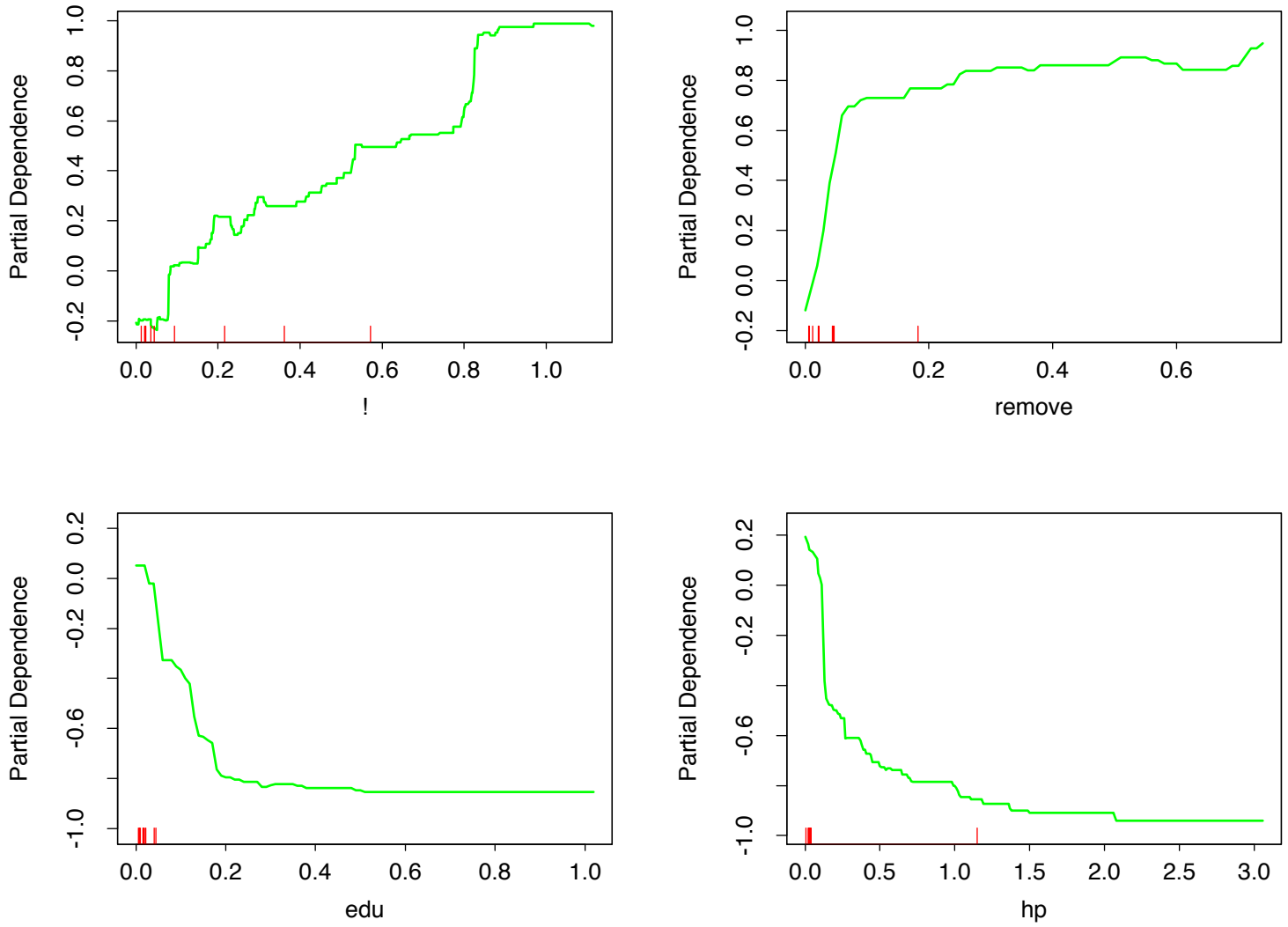


FIGURE 10.7. *Partial dependence of log-odds of* `spam` *on four important predictors. The red ticks at the base of the plots are deciles of the input variable.*

- You can get partial dependence plots by using the `partialPlot` command

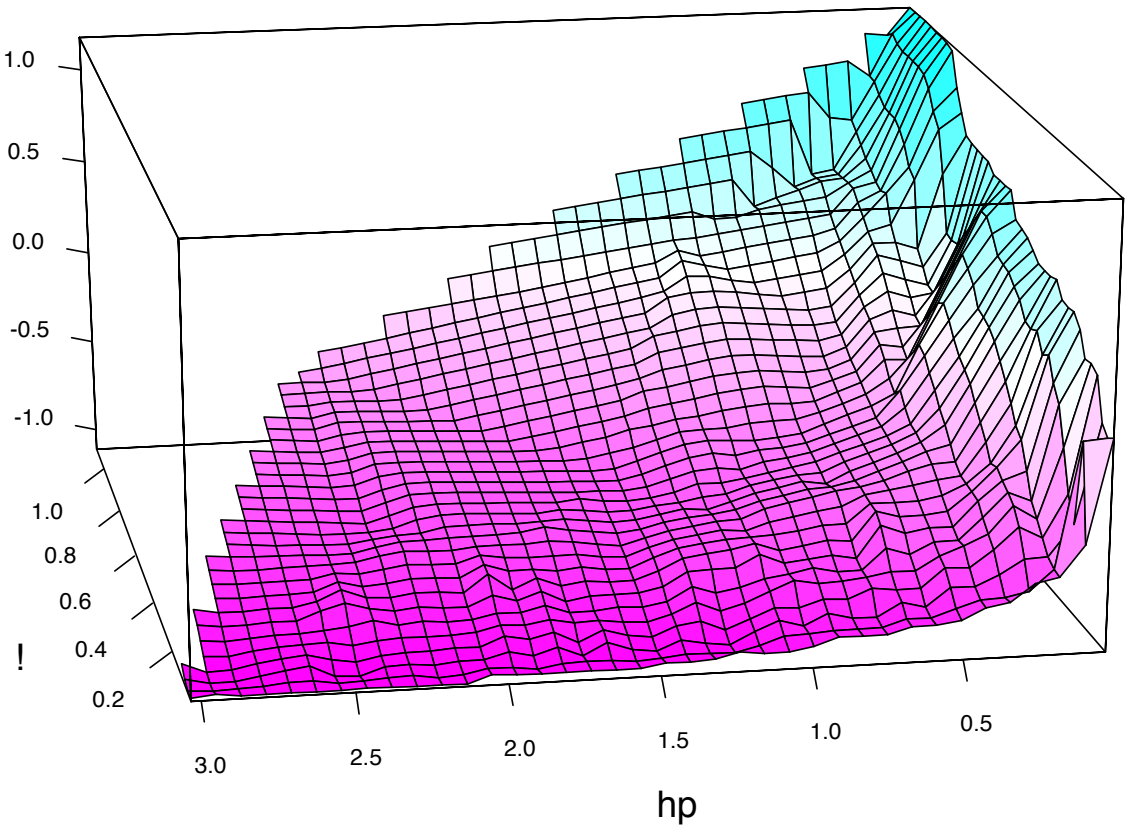


FIGURE 10.8. *Partial dependence of the log-odds of* `spam` *vs.* `email` *as a function of joint frequencies of* `hp` *and the character* `!`.

End of Part I

10 minute break

Bootstrap for SE estimation

- We introduced the Bootstrap for the purpose of bagging
- There's a more common use of bootstrapping: standard error estimation for complicated parameter estimates
- Commonly used to estimate the standard error of a coefficient, or to build confidence intervals
- Can be used to estimate uncertainty for very complex parameters, and in very complex sampling settings
  - We know how to do these things for Normal data, or when the Central limit theorem holds
  - Bootstrapping provides a way of estimating standard errors and building CI's even when the data generating distribution is non-Normal and the CLT cannot be expected to hold

A Toy Example: Asset Allocation

- Let X and Y denote the (log) returns of two financial assets
- We want to invest α of our money in asset X and $1 - \alpha$ of our money in asset Y
- We want to minimize the *risk* (variance) of our investment returns:

$$\mathrm{Var}(\alpha X + (1 - \alpha)Y)$$

- We're given 100 observations of daily returns $(x_1, y_1), \ldots, (x_{100}, y_{100})$
- In addition to estimating the best allocation (getting an estimate $\hat{\alpha}$), we also want to know the standard error of $\hat{\alpha}$
- If the SE of $\hat{\alpha}$ is large, this would mean that our investment strategy may be quite far from optimal

A Toy Example: Asset Allocation

- With some work, one can show that $\mathrm{Var}(\alpha X + (1-\alpha)Y)$ is minimized by

$$\alpha_{opt} = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}}$$

  where $\sigma_X^2 = \mathrm{Var}(X)$, $\sigma_Y^2 = \mathrm{Var}(Y)$, $\sigma_{XY} = \mathrm{Cov}(X, Y)$

- We can use the data to calculate the *sample* variances of X and Y, along with a sample covariance.

- Thus we can estimate the optimal allocation strategy with

$$\hat{\alpha} = \frac{\hat{\sigma}_Y^2 - \hat{\sigma}_{XY}}{\hat{\sigma}_X^2 + \hat{\sigma}_Y^2 - 2\hat{\sigma}_{XY}}$$

- Now the tricky part: What is the standard error of $\hat{\alpha}$?

A Toy Example: Asset Allocation

Here's our estimate of the optimal asset allocation:

$$\hat{\alpha} = \frac{\hat{\sigma}_Y^2 - \hat{\sigma}_{XY}}{\hat{\sigma}_X^2 + \hat{\sigma}_Y^2 - 2\hat{\sigma}_{XY}}$$

- Suppose that we knew the data generating process for (X, Y) *exactly*
- We could then:
  1. Simulate a bunch of *new* data sets of 100 observations (say, do this 1000 times)
  2. Calculate *new* estimates $\hat{\alpha}_1, \hat{\alpha}_2, \ldots, \hat{\alpha}_{1000}$
  3. Estimate the standard error of $\hat{\alpha}$ by calculating the standard deviation of the estimates $\{\hat{\alpha}_r\}_{r=1}^{1000}$ from our simulated data:

$$\hat{\mathrm{SE}}(\hat{\alpha}) = \sqrt{\frac{1}{1000 - 1} \sum_{r=1}^{1000} (\hat{\alpha}_r - \bar{\alpha})}$$

where $\bar{\alpha} = \frac{1}{1000} \sum_{r=1}^{1000} \hat{\alpha}_r$

A Toy Example: Bootstrap Solution

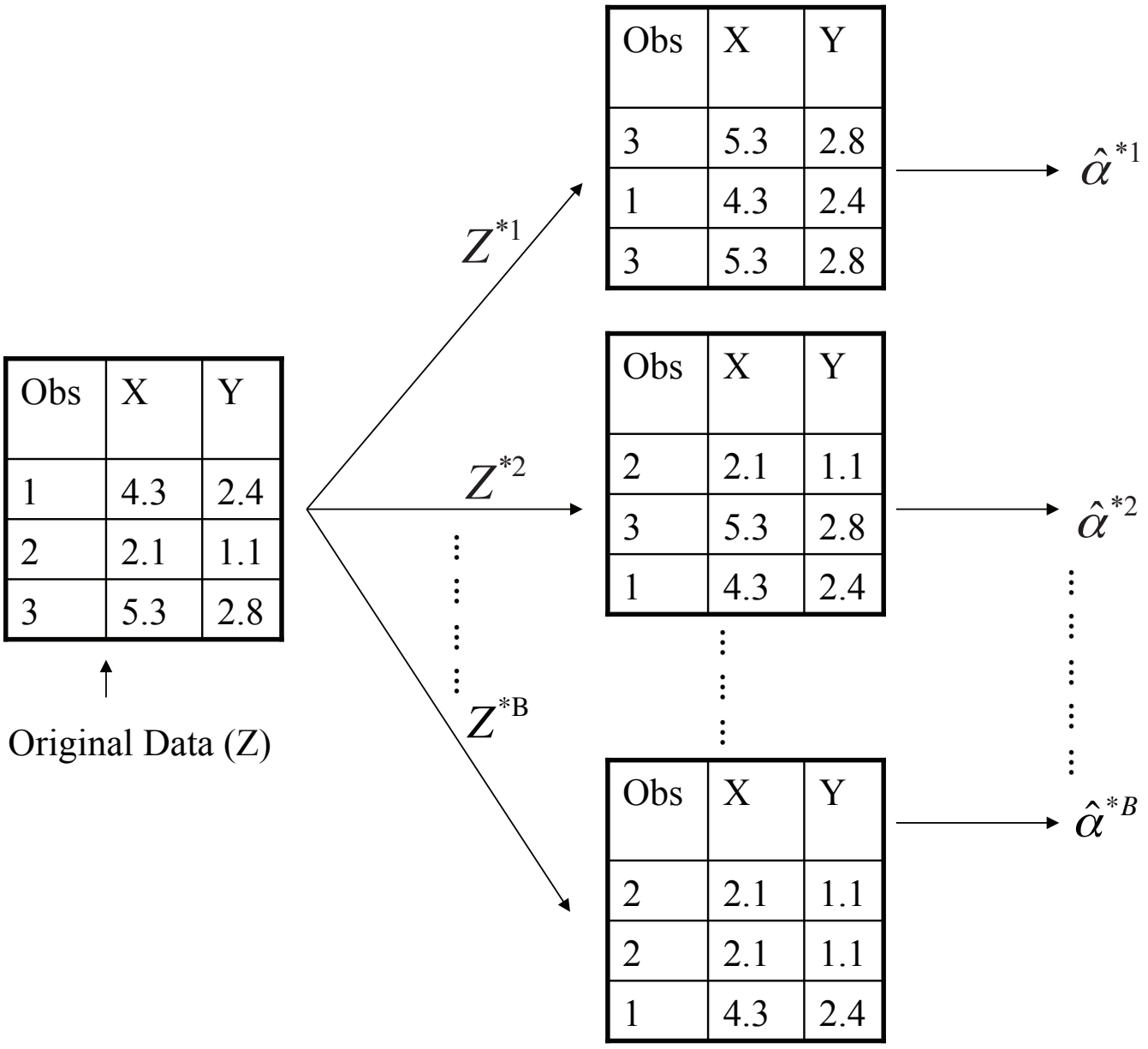
- Great! There's just one major problem…we do not know the distribution of X and Y exactly, so we can't simulate new batches of data

- Bootstrap approach: Let's try generating new data sets by resampling from the data itself…*Sounds crazy, right?*
  1. Get B new data sets $Z^{*1}, \ldots, Z^{*B}$, each by sampling 100 observations with replacement from our observed data (do this, say, $B = 1000$ times)
  2. Calculate new estimates $\hat{\alpha}^{*1}, \hat{\alpha}^{*2}, \ldots, \hat{\alpha}^{*B}$
  3. Estimate the standard error of $\hat{\alpha}$ by calculating the standard deviation of the estimates from our simulated data:

$$\hat{\mathrm{SE}}_B(\hat{\alpha}) = \sqrt{\frac{1}{B-1}\sum_{r=1}^{B}(\hat{\alpha}^{*r} - \bar{\alpha}^*)}$$

where $\bar{\alpha}^* = \frac{1}{B}\sum_{r=1}^{B}\hat{\alpha}^{*r}$

A Bootstrap Picture



How well did we do?

- When we know the data generating process (see p.188 of ISL), simulating 1000 data sets and calculating the standard errors of the corresponding $\hat{\alpha}$ estimates gives

$$\hat{\mathrm{SE}}(\hat{\alpha}) = 0.083$$

- Starting with a single data set of $n = 100$ observations and running the bootstrap procedure to resample $B = 1000$ data sets gives

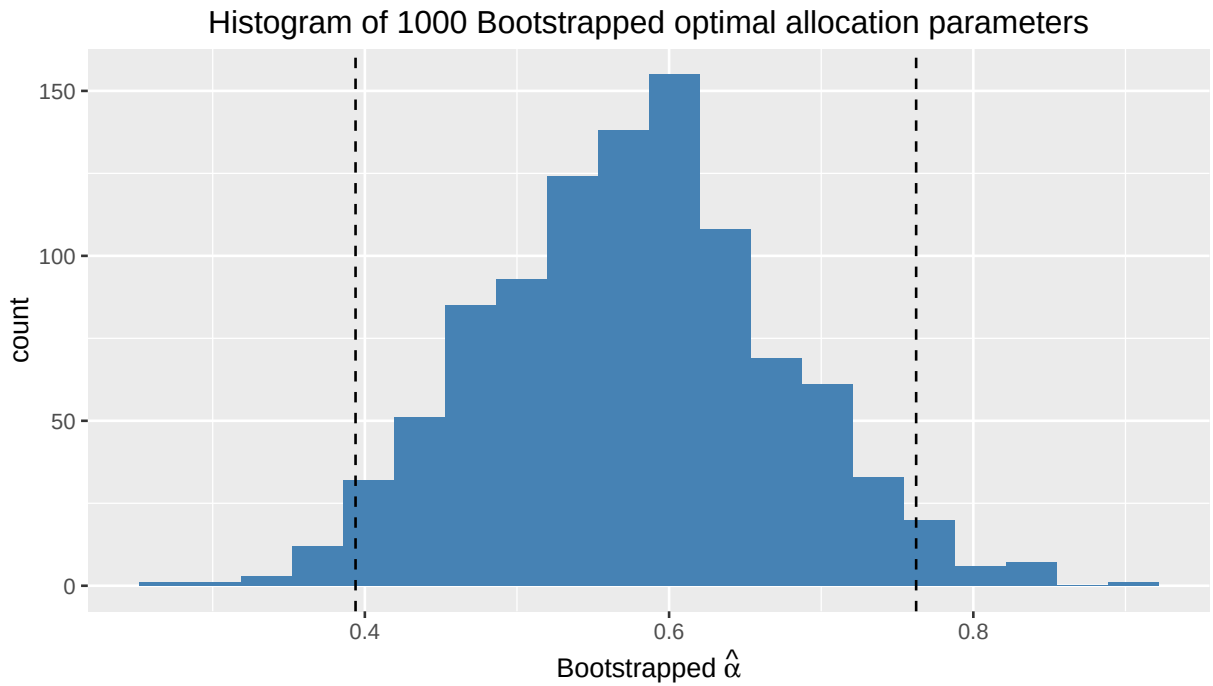
$$\hat{\mathrm{SE}}_B(\hat{\alpha}) = 0.087$$

- Amazing!

- Say we get $\hat{\alpha} = 0.576$. The estimated SE is non-negligible, so we know that there's still *a fair bit of uncertainty* in what allocation to choose. But the SE is small enough that choosing an allocation close to $\hat{\alpha} = 0.576$ seems like a reasonable thing to do.

Bootstrap Confidence Intervals

- The bootstrap procedure gives us B estimates $\hat{\alpha}^{*1}, \hat{\alpha}^{*2}, \ldots, \hat{\alpha}^{*B}$



- To form a $(1-\gamma)\cdot 100\%$ CI for α, we can use the $\gamma/2$ and $1-\gamma/2$ percentiles of the bootstrapped estimates
- In this simulation, we would get a 95% CI of $[0.39, 0.76]$

Agenda for Part II

- **What is Unsupervised learning?**
- **K-means clustering**
- **Hierarchical clustering**
- **Association rule mining**

What is Unsupervised Learning?

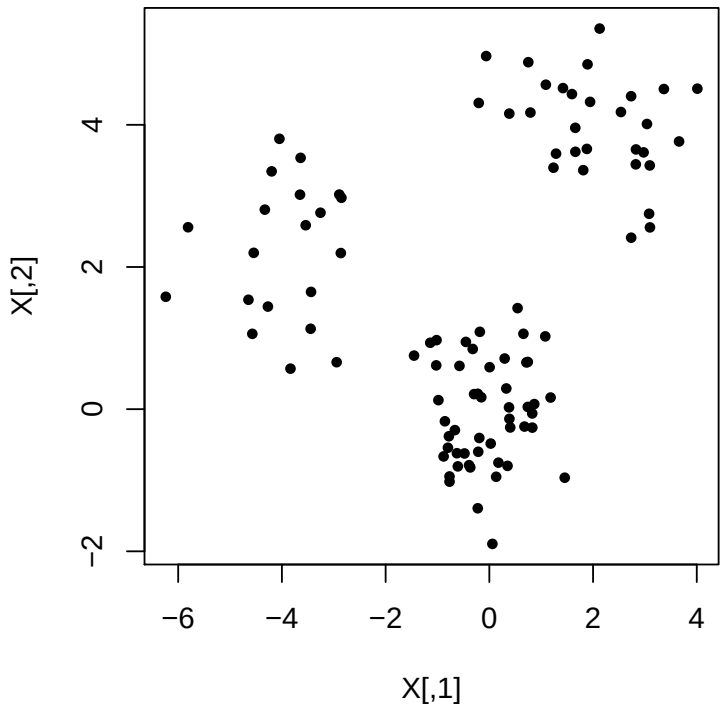
- **Unsupervised learning**, also called Descriptive analytics, describes a family of methods for uncovering latent structure in data

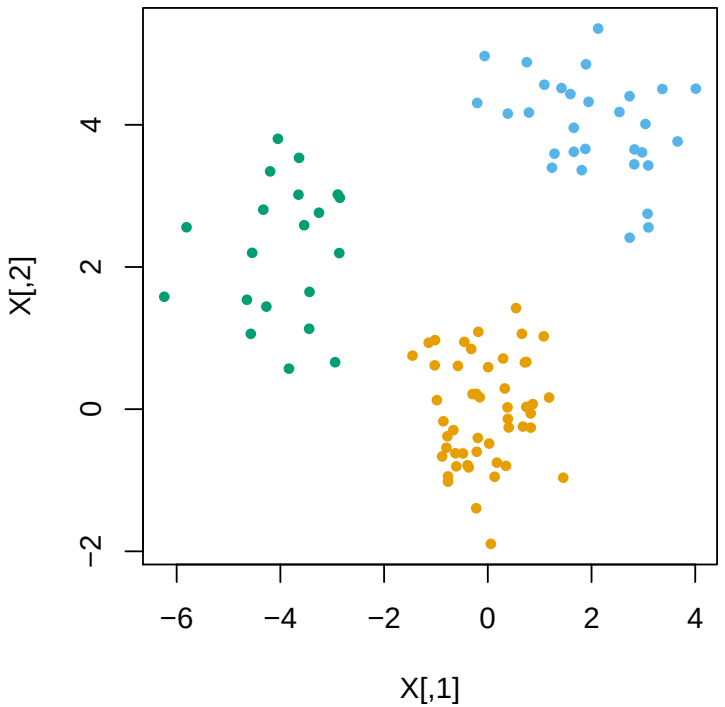
- In Supervised learning aka Predictive analytics, our data consisted of observations (x_i, y_i), $x_i \in \mathbb{R}^p$, $i = 1, \ldots, n$
  - Such data is called *labelled*, and the y_i are thought of as the *labels* for the data

- In Unsupervised learning, we just look at data x_i, $i = 1, \ldots, n$.
  - This is called *unlabelled* data
  - Even if we have labels y_i, we may still wish to temporarily ignore the y_i and conduct unsupervised learning on the inputs x_i

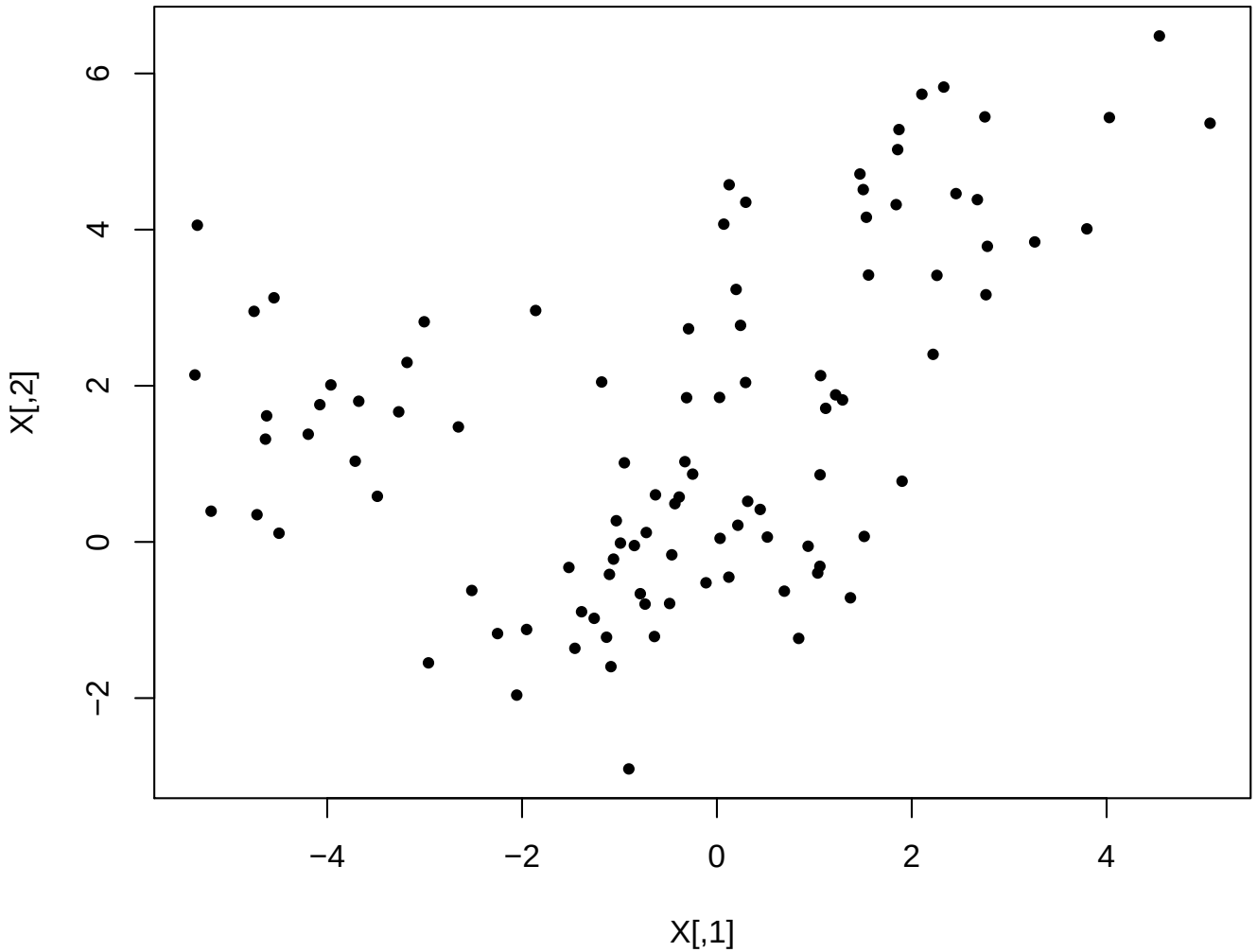
Examples of clustering tasks

- Identify similar groups of online shoppers based on their browsing and purchasing history
- Identify similar groups of music listeners or movie viewers based on their ratings or recent listening/viewing patterns
- Cluster input variables based on their correlations to remove redundant predictors from consideration
- Cluster hospital patients based on their medical histories
- Determine how to place sensors, broadcasting towers, law enforcement, or emergency-care centers to guarantee that desired *coverage criteria* are met

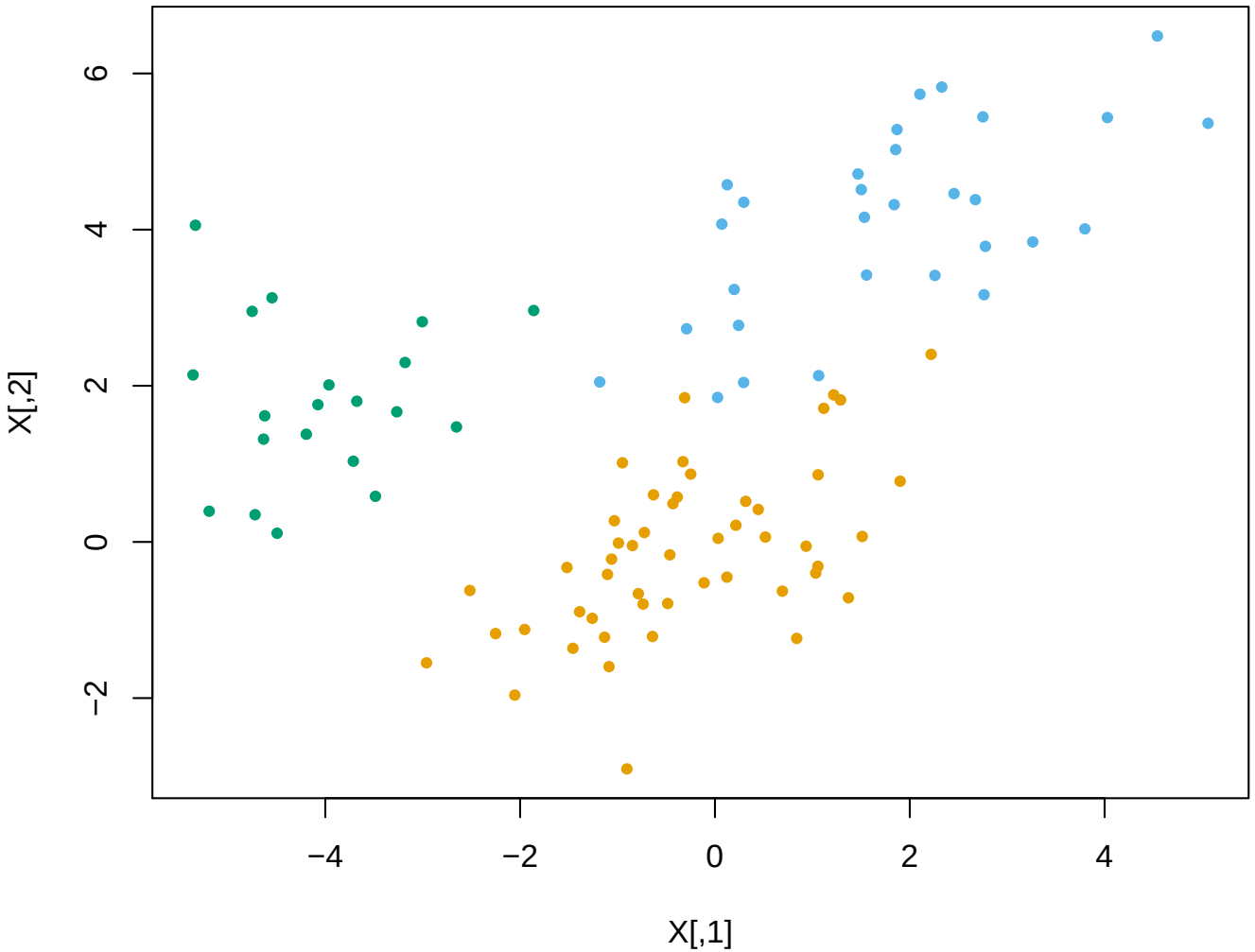




- Left: Data
- Right: One possible way to cluster the data



Here's a less clear example. How should we partition it?



Here's one reasonable clustering.

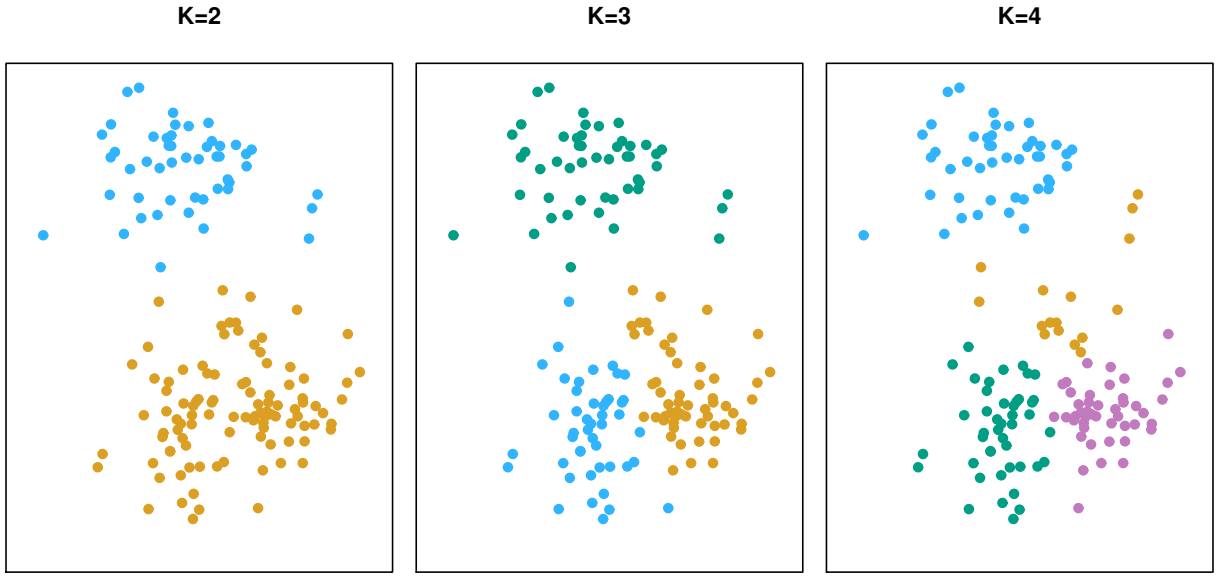


Figure 10.5 from ISL

- A clustering is a *partition* $\{C_1, \ldots, C_K\}$, where each C_k denotes a subset of the observations.
- Each observation belongs to one and only one of the clusters
- To denote that the ith observation is in the kth cluster, we write $i \in C_k$

Method: K-mean clustering

- Main idea: A good clustering is one for which the within-cluster variation is as small as possible.

- The within-cluster variation for cluster C_k is some measure of the amount by which the observations within each class differ from one another

- We'll denote it by $WCV(C_k)$

- Goal: Find $C_1, \ldots, C_K$ that minimize

$$\sum_{k=1}^{K} \mathrm{WCV}(C_k)$$

- This says: Partition the observations into K clusters such that the WCV summed up over all K clusters is as small as possible

How to define within-cluster variation?

- Goal: Find $C_1, \ldots, C_K$ that minimize

$$\sum_{k=1}^{K} \text{WCV}(C_k)$$

- Typically, we use Euclidean distance:

$$\text{WCV}(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

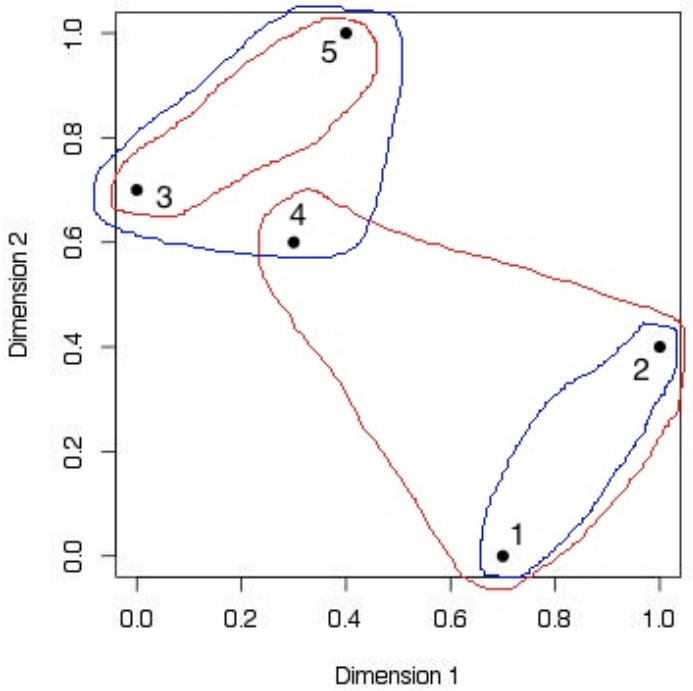
  where $|C_k|$ denotes the number of observations in cluster k

- To be clear: We're treating K as fixed ahead of time. We are *not* optimizing K as part of this objective.

Simple example

Here $n = 5$ and $K = 2$,
The full *distance matrix* for all 5 observations is shown below.

	1	2	3	4	5
1	0	0.25	0.98	0.52	1.09
2	0.25	0	1.09	0.53	0.72
3	0.98	1.09	0	0.10	0.25
4	0.52	0.53	0.10	0	0.17
5	1.09	0.72	0.25	0.17	0



- Red clustering: $\sum \text{WCV}_k = (0.25 + 0.53 + 0.52)/3 + 0.25/2 = 0.56$
- Blue clustering: $\sum \text{WCV}_k = 0.25/2 + (0.10 + 0.17 + 0.25)/3 = 0.30$
- It's easy to see that the Blue clustering minimizes the within-cluster variation among all possible partitions of the data into $K = 2$ clusters

How do we minimize WCV?

$$\sum_{k=1}^{K} \text{WCV}(C_k) = \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

$$= \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \|x_i - x_{i'}\|_2^2$$

- It's *computationally infeasible* to actually minimize this criterion
- We essentially have to try all possible partitions of n points into K sets.
- When $n = 10$, $K = 4$, there are $34{,}105$ possible partitions
- When $n = 25$, $K = 4$, there are 5×10^{13}…
- We're going to have to settle for an approximate solution

K-means algorithm

- It turns out that we can rewrite WCV_k more conveniently:

$$\mathrm{WCV}_k = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \|x_i - x_{i'}\|_2^2 = 2 \sum_{i \in C_k} \|x_i - \bar{x}_k\|^2$$

  where $\bar{x}_k = \frac{1}{|C_k|} \sum_{i \in C_k} x_i$ is just the average of all the points in cluster C_k

So let's try the following:

K-means algorithm

1. Start by randomly partitioning the observations into K clusters
2. Until the clusters stop changing, repeat:
   - 2a. For each cluster, compute the cluster centroid $\bar{x}_k$
   - 2b. Assign each observation to the cluster whose centroid is the closest

K-means demo with $K = 3$

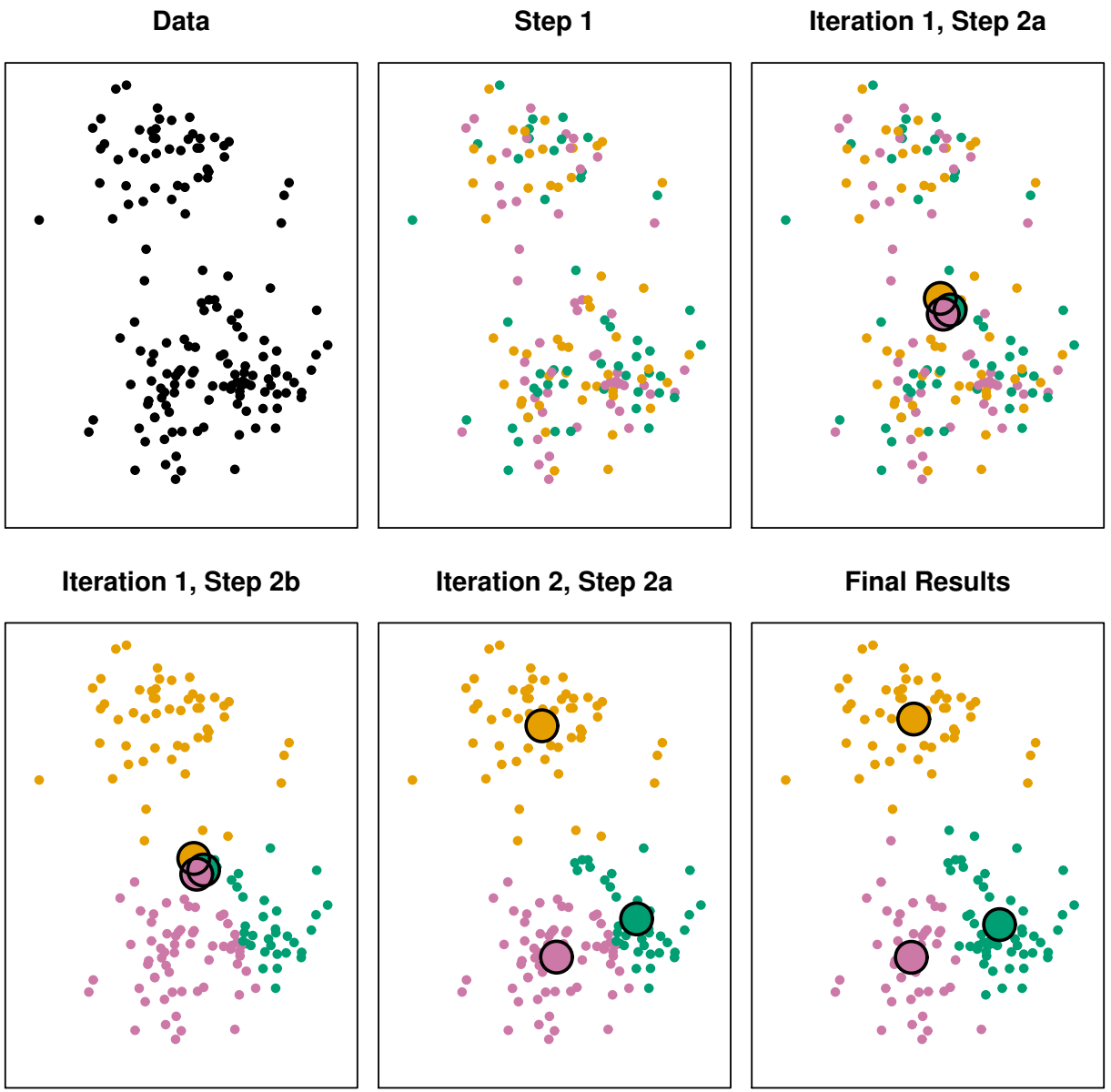


Figure 10.6 from ISL

Do the random starting values matter?

Figure 10.7 from ISL: Final results from 6 different random starting values
The 4 red results all attain the same solution

Summary of K-means

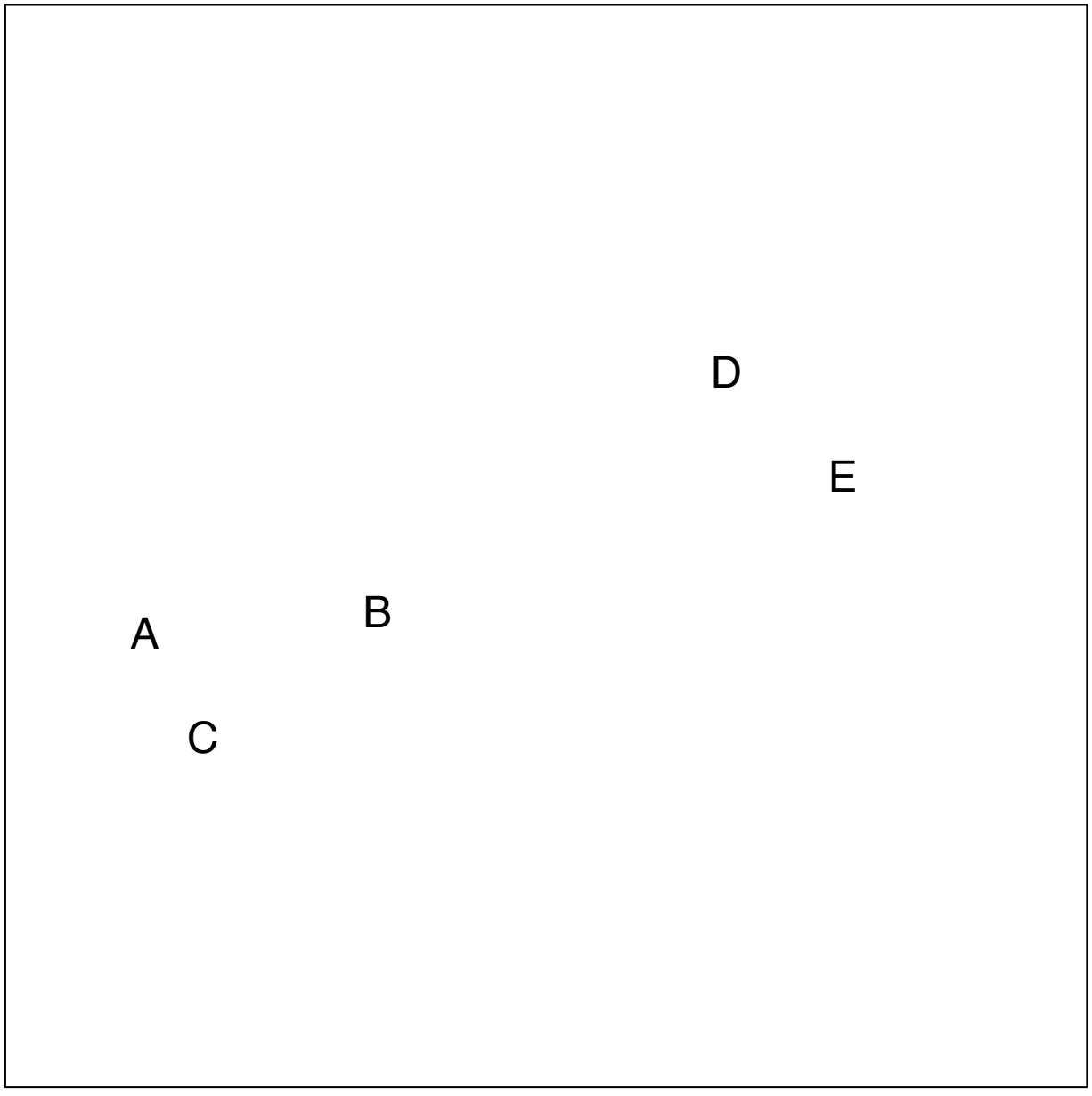
We'd love to minimize

$$\sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \|x_i - x_{i'}\|_2^2$$

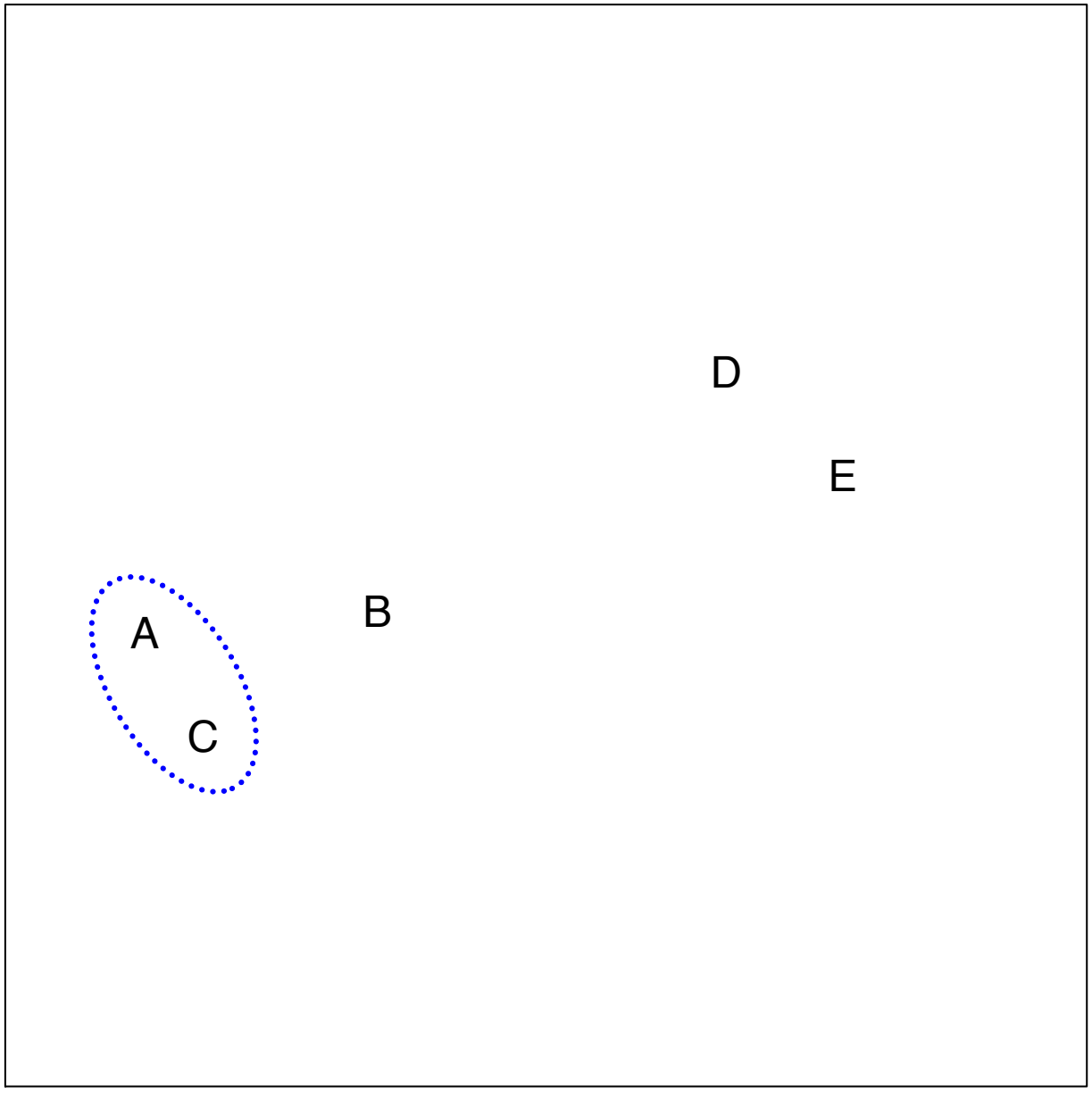
- It's *infeasible* to actually optimize this in practice, but K-means at least gives us a so-called local optimum of this objective
- The result we get depends both on K, and also on the *random initialization* that we wind up with
- It's a good idea to try different random starts and pick the best result among them
- There's a method called K-means++ that improves how the clusters are initialized
- A related method, called K-medoids, clusters based on distances to a *centroid* that is chosen to be one of the points in each cluster

Hierarchical clustering

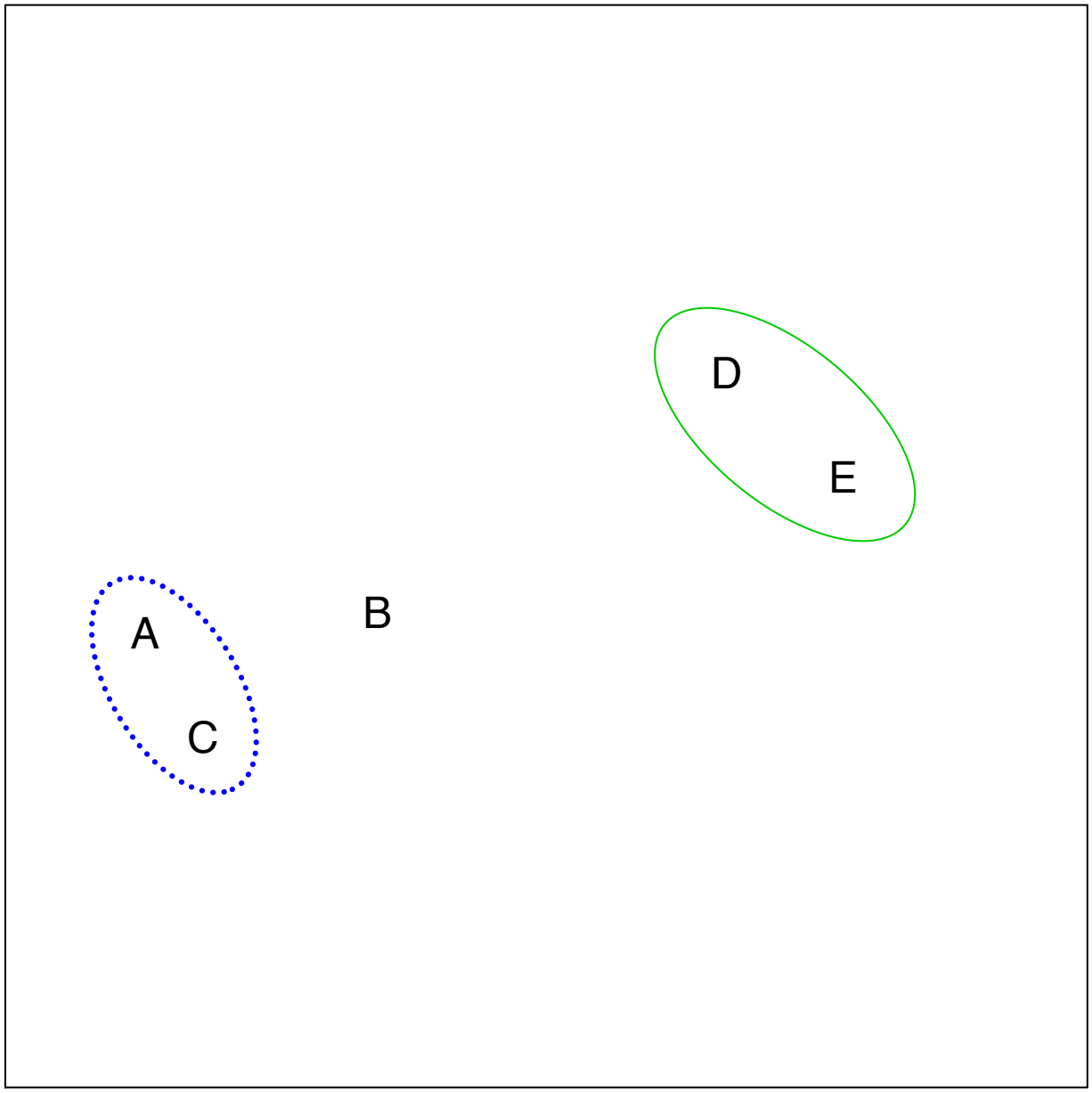
- K-means is an objective-based approach that requires us to pre-specify the number of clusters K
- The answer it gives is somewhat random: it depends on the random initialization we started with
- Hierarchical clustering is an alternative approach that does not require a pre-specified choice of K, and which provides a deterministic answer (no randomness)
- We'll focus on bottom-up or agglomerative hierarchical clustering
- top-down or divisive clustering is also good to know about, but we won't directly cover it here



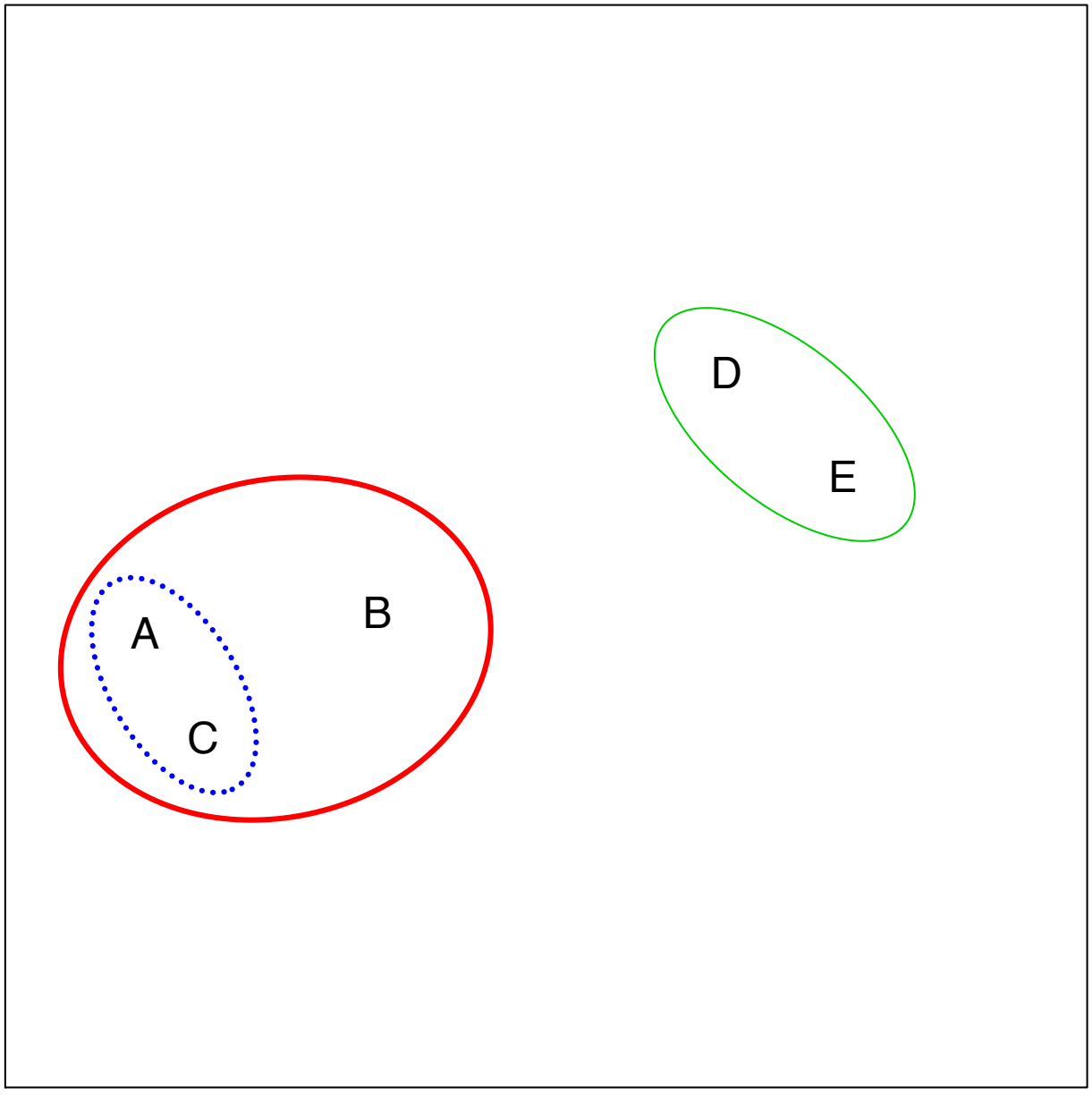
Each point starts as its own cluster



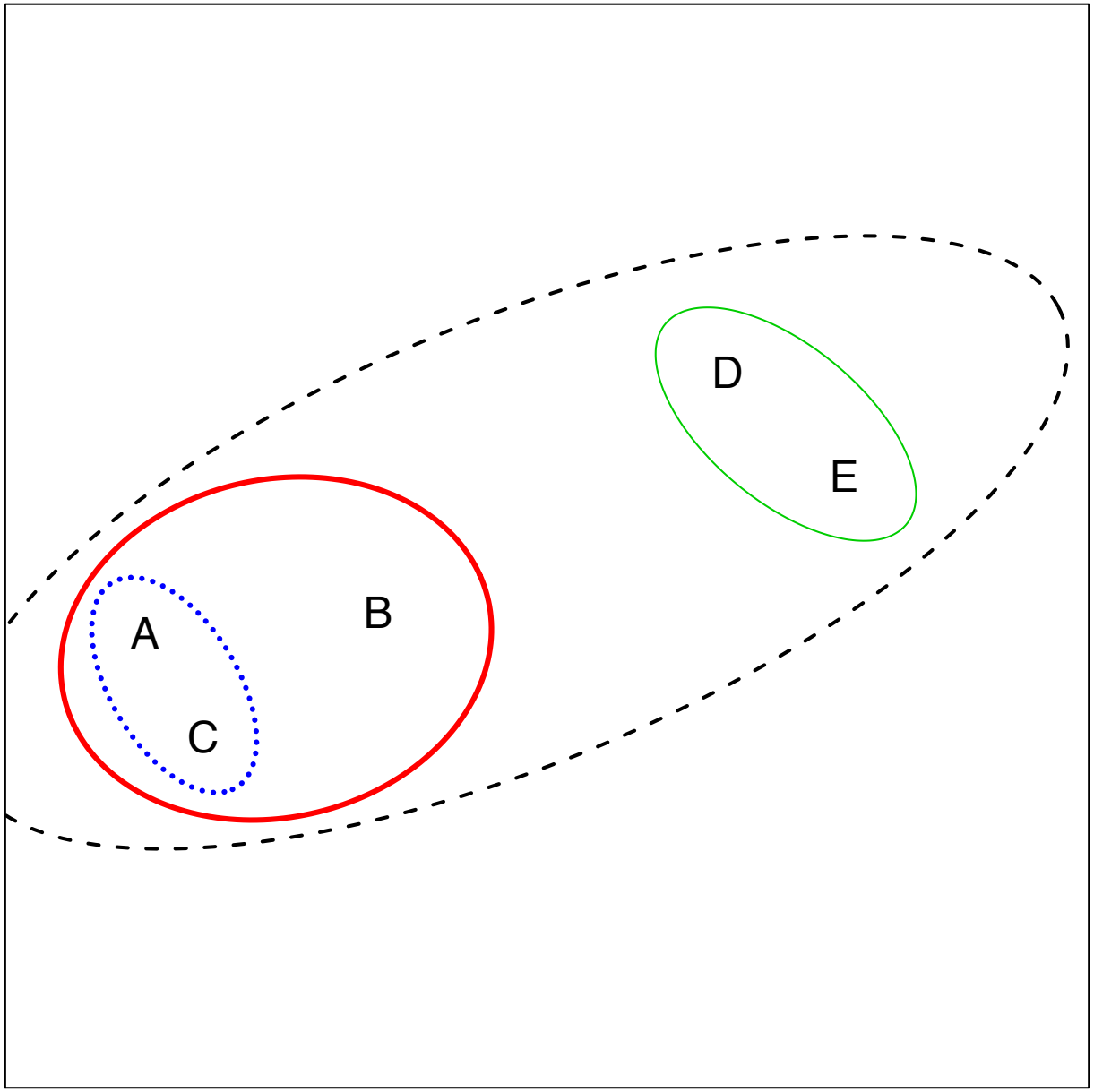
We merge the two clusters (points) that are closet to each other



Then we merge the next two closest clusters



Then the next two closest clusters…

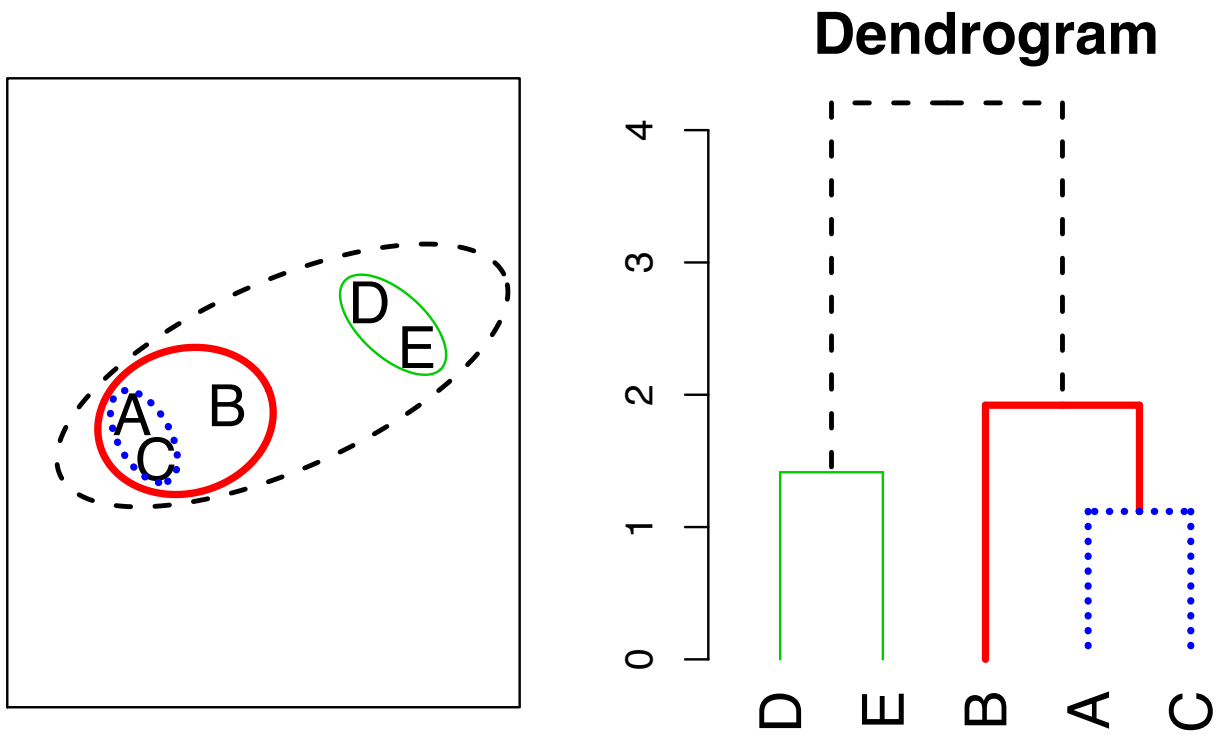


Until at last all of the points are all in a single cluster

Agglomerative Hierarchical Clustering

- Start with each point in its own cluster.
- Identify the two closest clusters. Merge them.
- Repeat until all points are in a single cluster

To visualize the results, we can look at the resulting dendrogram



y-axis on dendrogram is (proportional to) the distance between the clusters that got merged at that step

An example

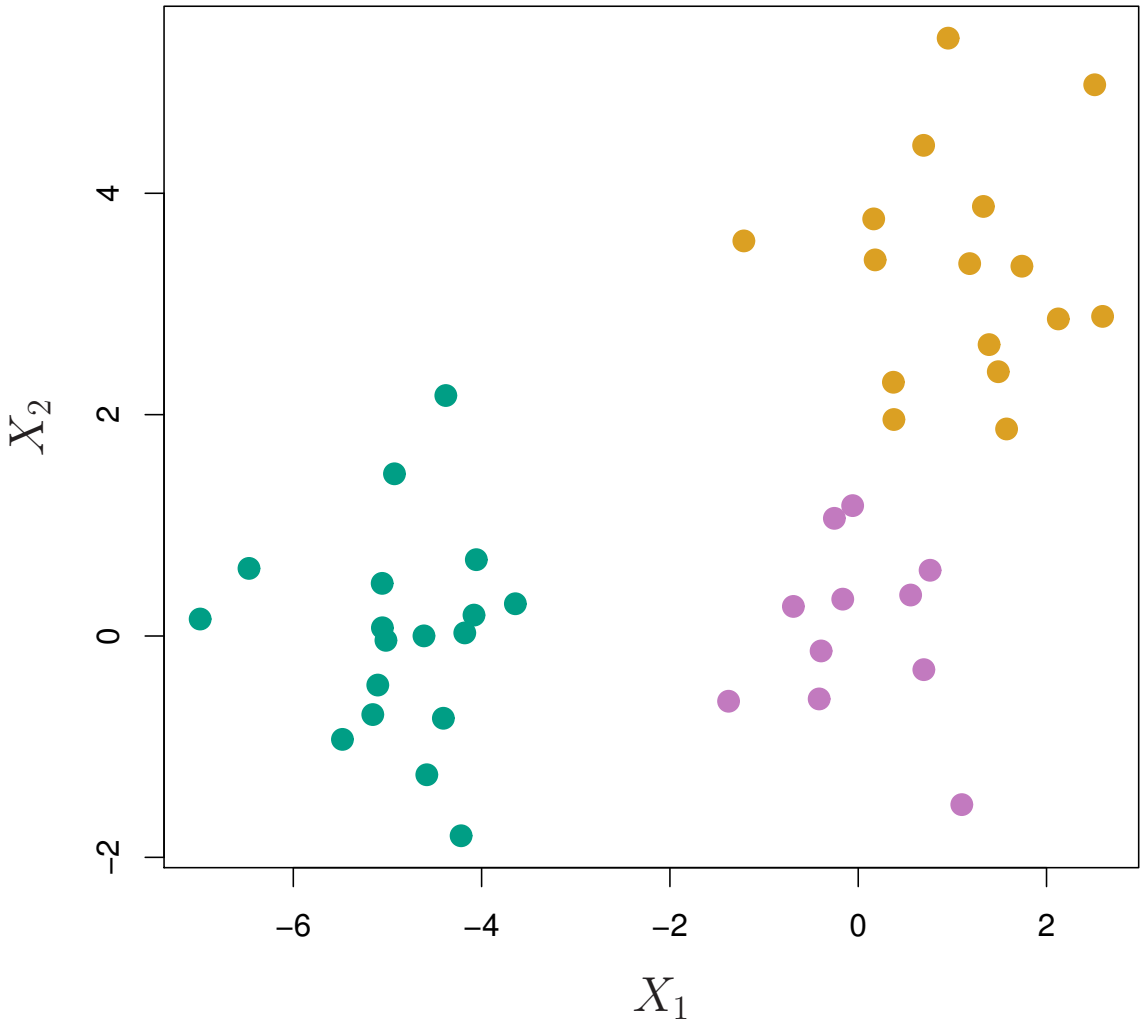


Figure 10.8 from ISL. $n = 45$ points shown.

Cutting dendrograms (example cont'd)

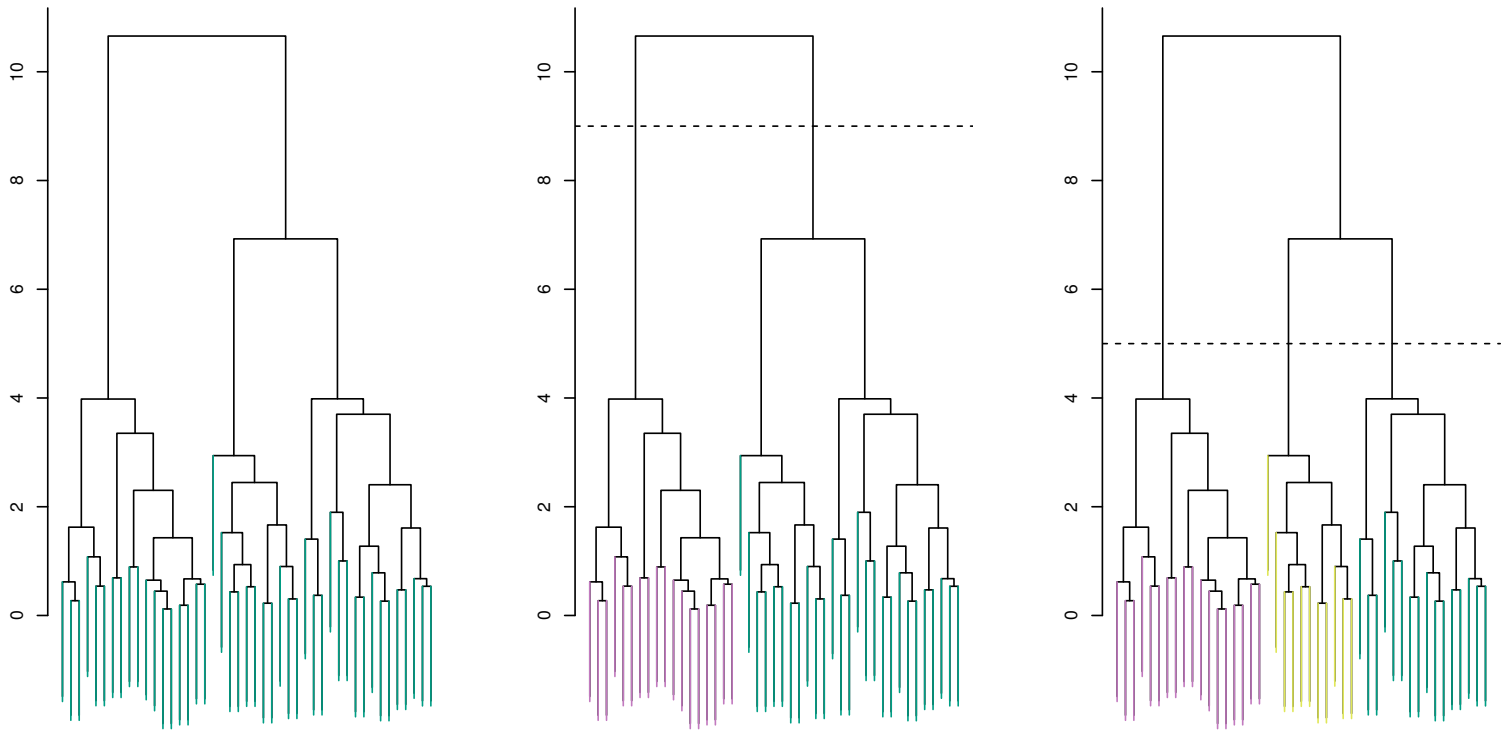


Figure 10.9 from ISL

- Left: Dendrogram obtained from complete linkage[4] clustering
- Center: Dendrogram cut at height 9, resulting in $K = 2$ clusters
- Right: Dendrogram cut at height 5, resulting in $K = 3$ clusters

[4] We'll talk about *linkages* in a moment

Interpreting dendrograms

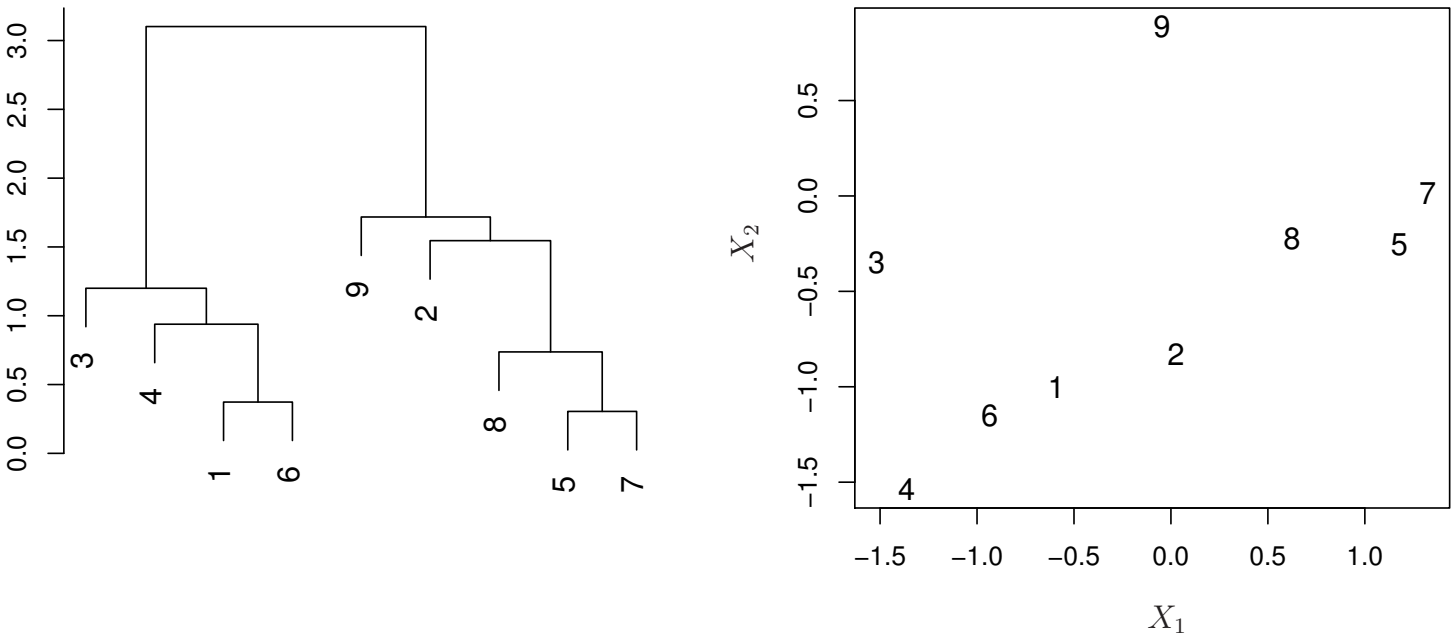


Figure 10.10 from ISL

- Observations 5 and 7 are similar to each other, as are observations 1 and 6
- Observation 9 is no more similar to observation 2 than it is to observations 8, 5 and 7
  - This is because observations $\{2, 8, 5, 7\}$ all fuse with 9 at height ~ 1.8

Linkages

- Let $d_{ij} = d(x_i, x_j)$ denote the dissimilarity[5] (distance) between observation x_i and x_j
- At our first step, each cluster is a single point, so we start by merging the two observations that have the *lowest dissimilarity*
- But after that…we need to think about distances not between points, but between sets (clusters)
- The dissimilarity between two clusters is called the linkage
- i.e., Given two sets of points, G and H, a linkage is a dissimilarity measure $d(G, H)$ telling us how different the points in these sets are
- Let's look at some examples

[5] We'll talk more about dissimilarities in a moment

Common linkage types

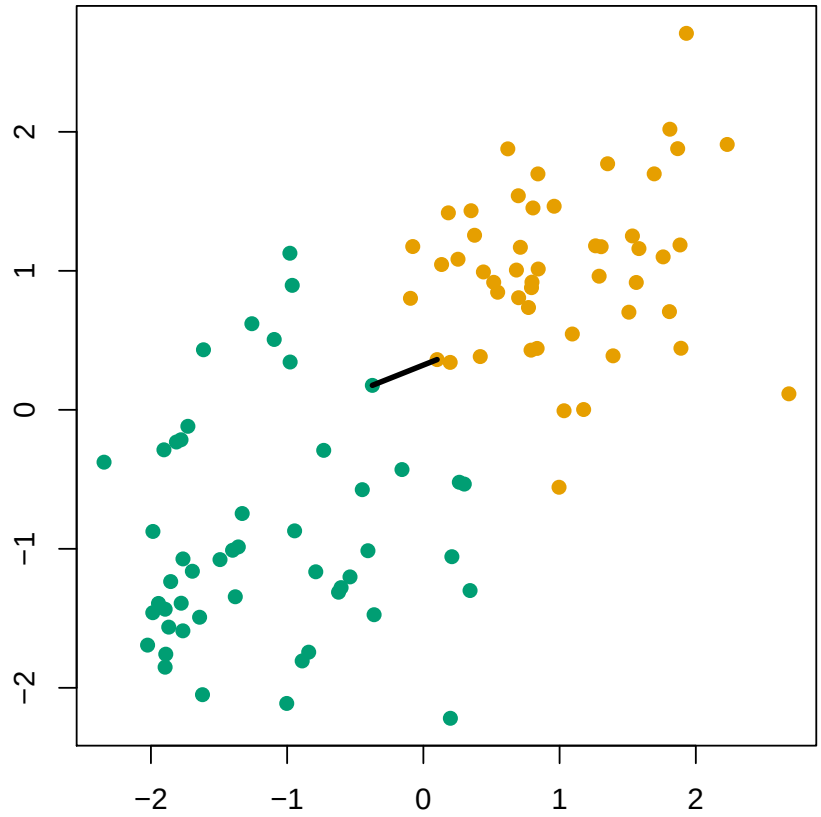
Linkage	*Description*
Complete	Maximal inter-cluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the *largest* of these dissimilarities.
Single	Minimal inter-cluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the *smallest* of these dissimilarities.
Average	Mean inter-cluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the *average* of these dissimilarities.
Centroid	Dissimilarity between the centroid for cluster A (a mean vector of length p) and the centroid for cluster B. Centroid linkage can result in undesirable *inversions*.

Single linkage

In single linkage (i.e., nearest-neighbor linkage), the dissimilarity between G, H is the smallest dissimilarity between two points in different groups:

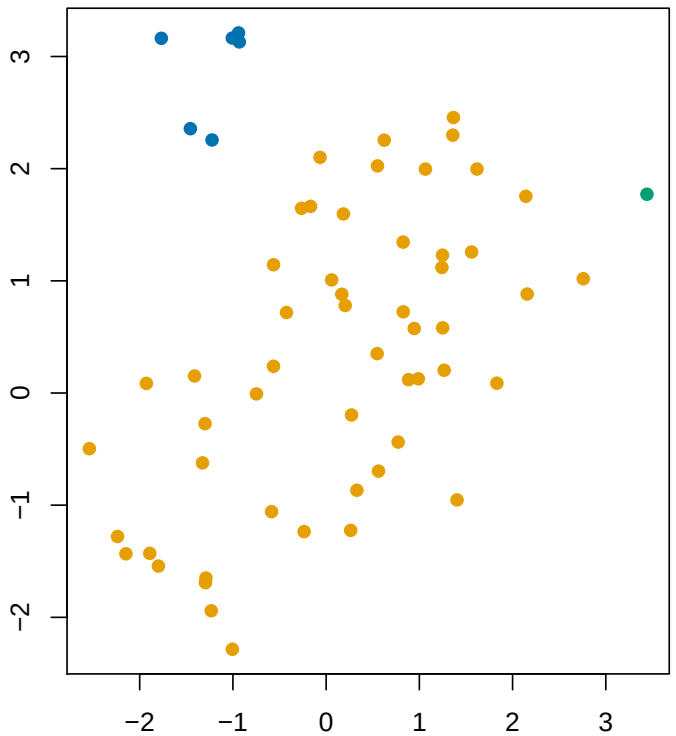
$$d_{\text{single}}(G, H) = \min_{i \in G,\, j \in H} d(x_i, x_j)$$

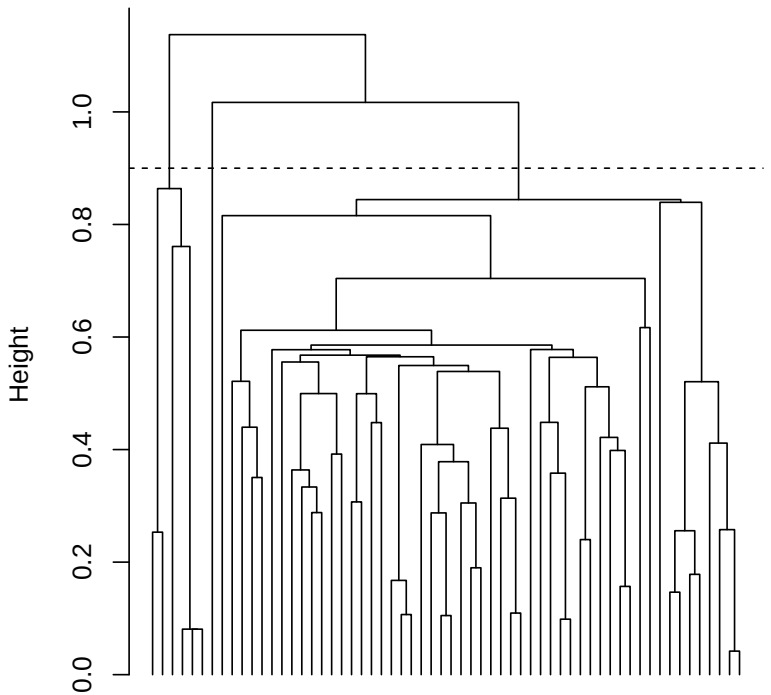
Example (dissimilarities d_{ij} are distances, groups are marked by colors): single linkage score $d_{\text{single}}(G, H)$ is the distance of the closest pair



Single linkage example

Here $n = 60$, $x_i \in \mathbb{R}^2$, $d_{ij} = \|x_i - x_j\|_2$. Cutting the tree at $h = 0.9$ gives the clustering assignments marked by colors





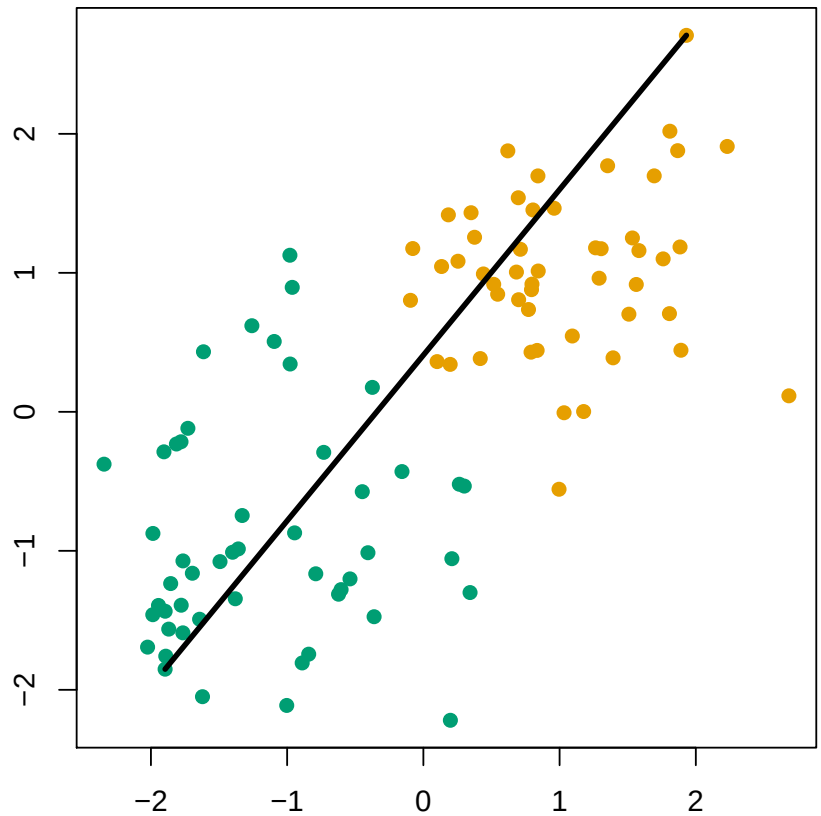
Cut interpretation: for each point x_i, there is another point x_j in its cluster such that $d(x_i, x_j) \leq 0.9$

Complete linkage

In complete linkage (i.e., furthest-neighbor linkage), dissimilarity between G, H is the largest dissimilarity between two points in different groups:

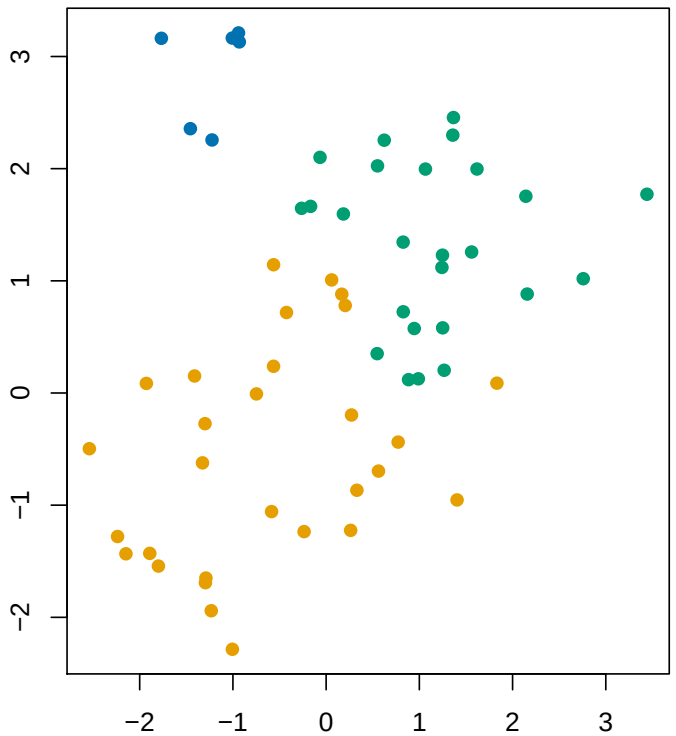
$$d_{\text{complete}}(G, H) = \max_{i \in G,\, j \in H} d(x_i, x_j)$$

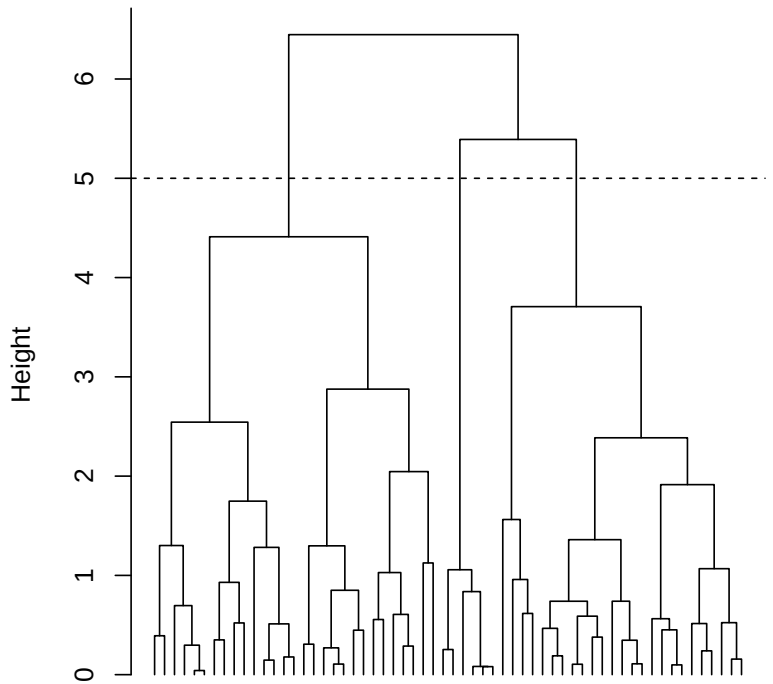
Example (dissimilarities d_{ij} are distances, groups are marked by colors): complete linkage score $d_{\text{complete}}(G, H)$ is the distance of the furthest pair



Complete linkage example

Same data as before. Cutting the tree at $h = 5$ gives the clustering assignments marked by colors





Cut interpretation: for each point x_i, every other point x_j in its cluster satisfies $d(x_i, x_j) \leq 5$

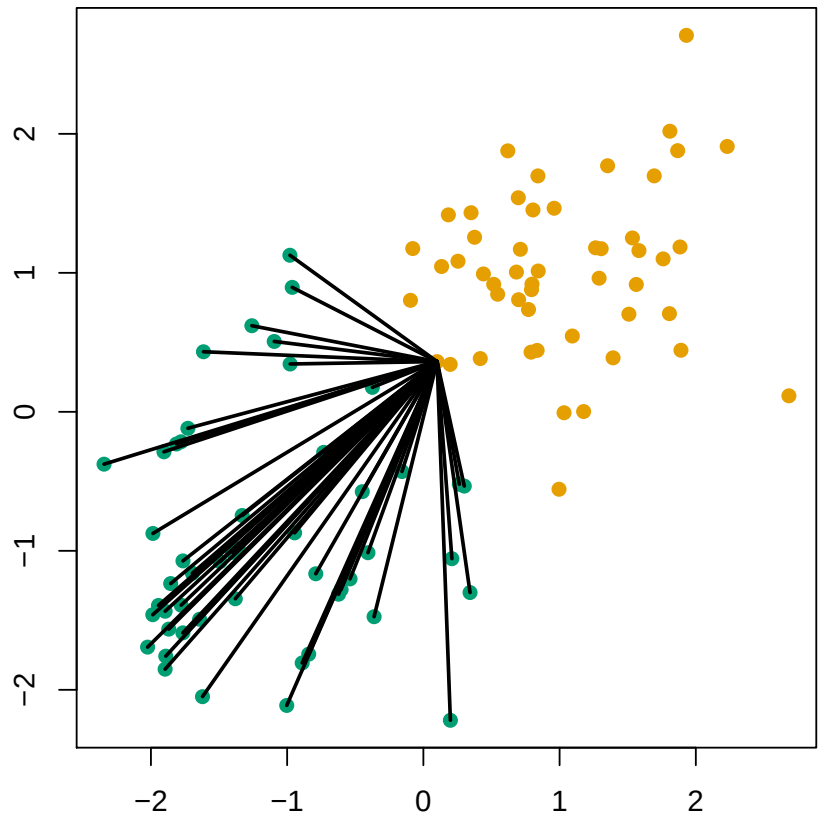
Average linkage

In average linkage, the dissimilarity between G, H is the average dissimilarity over all points in opposite groups:

$$d_{\text{average}}(G, H) = \frac{1}{|G| \cdot |H|} \sum_{i \in G,\, j \in H} d(x_i, x_j)$$

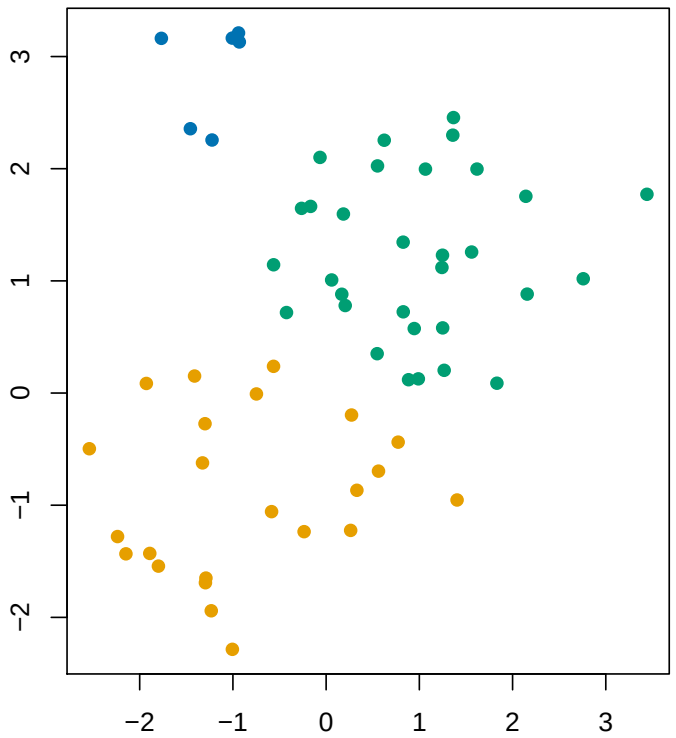
Example (dissimilarities d_{ij} are distances, groups are marked by colors): average linkage score $d_{\text{average}}(G, H)$ is the average distance across all pairs

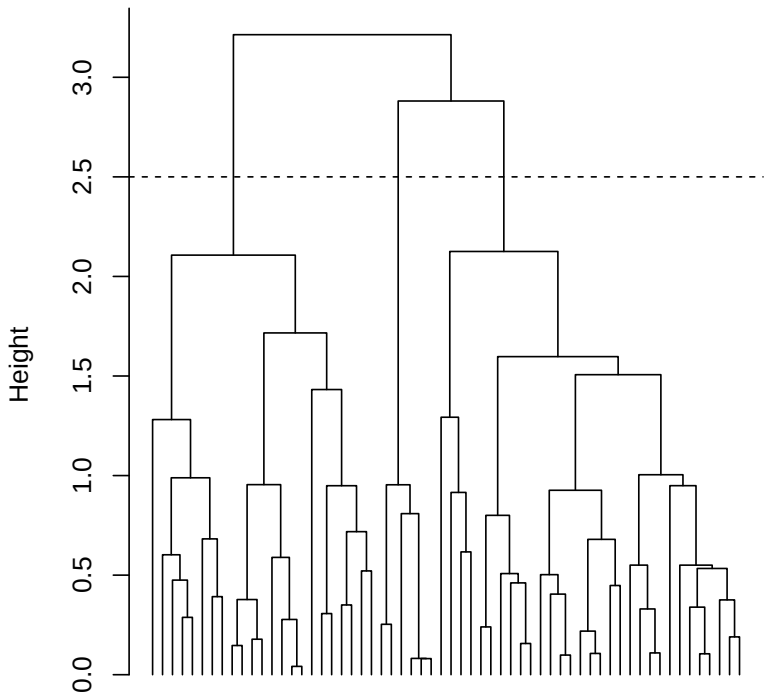
(Plot here only shows distances between the green points and one orange point)



Average linkage example

Same data as before. Cutting the tree at $h = 2.5$ gives clustering assignments marked by the colors





Cut interpretation: there really isn't a good one! ☹

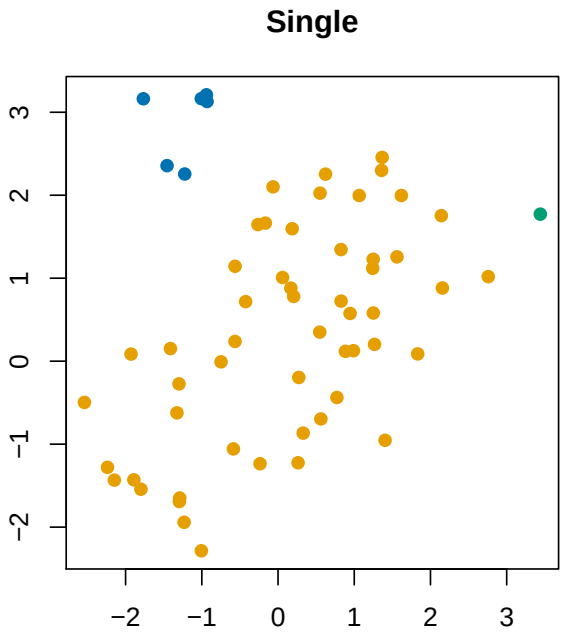
Shortcomings of Single and Complete linkage

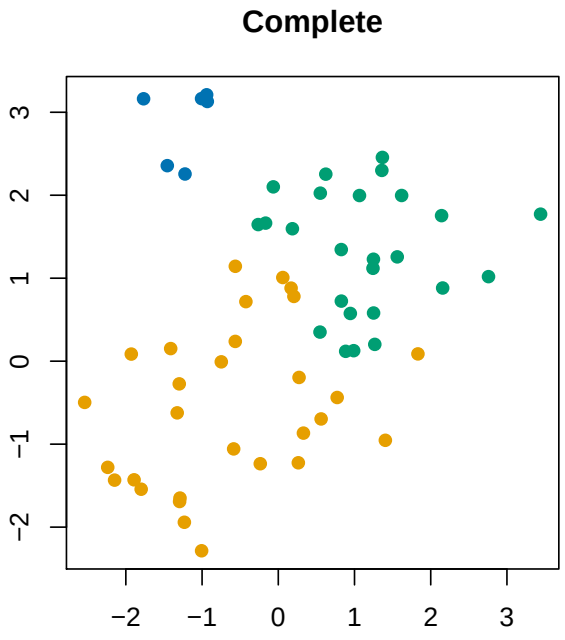
Single and complete linkage have some practical problems:

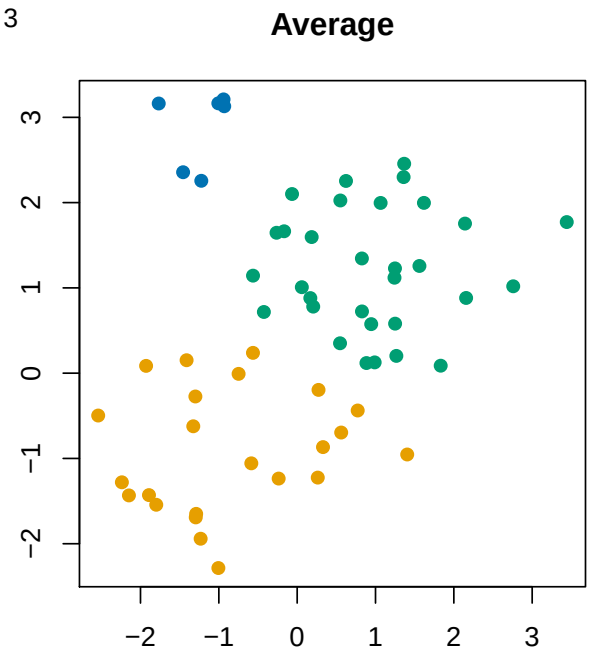
- Single linkage suffers from chaining.
  - In order to merge two groups, only need one pair of points to be close, irrespective of all others. Therefore *clusters can be too spread out*, and not compact enough.
- Complete linkage avoids chaining, but suffers from crowding.
  - Because its score is based on the worst-case dissimilarity between pairs, *a point can be closer to points in other clusters than to points in its own cluster*. Clusters are compact, but not far enough apart.

Average linkage tries to strike a balance. It uses average pairwise dissimilarity, so clusters tend to be relatively compact and relatively far apart

Example of chaining and crowding





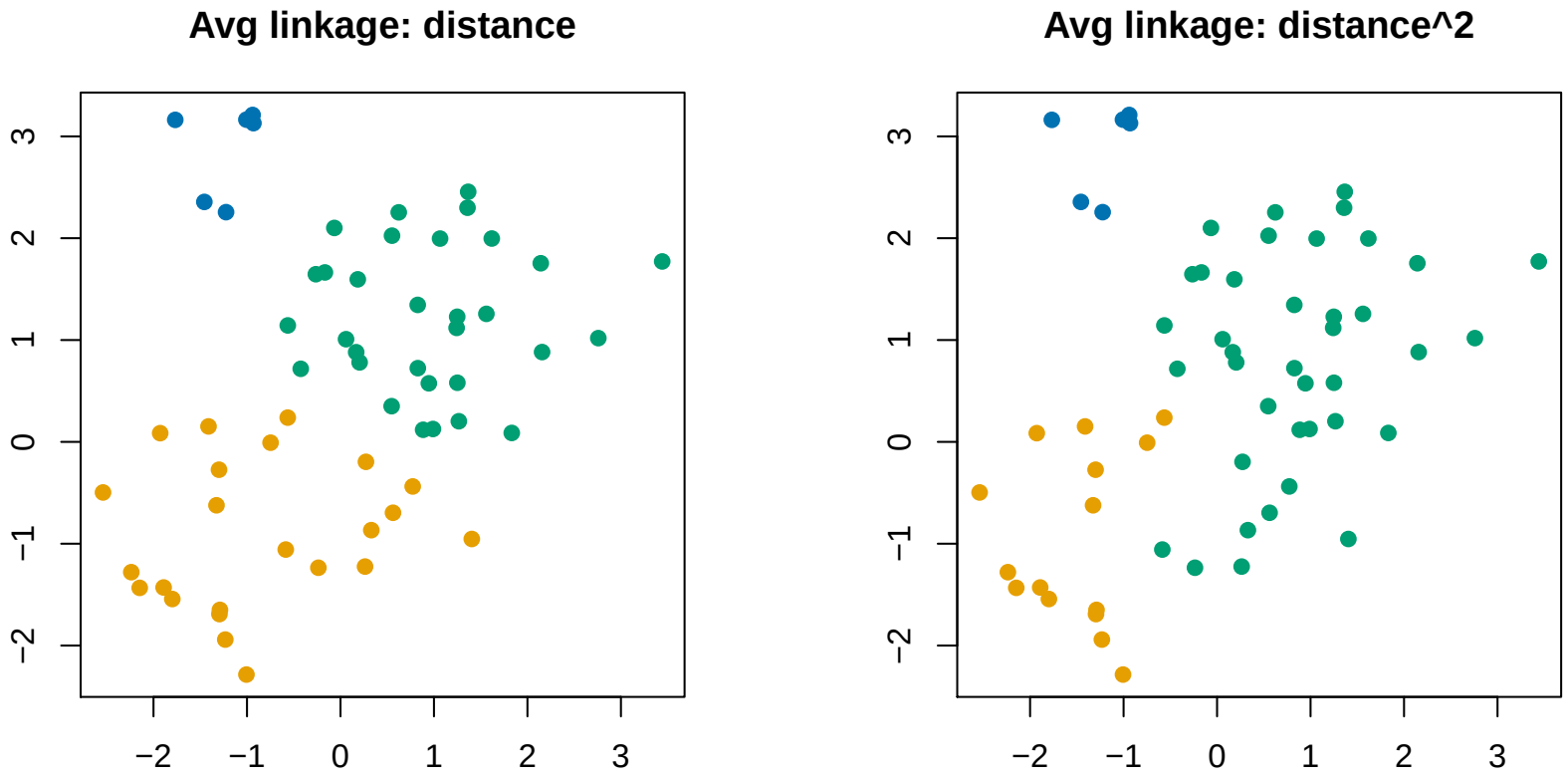


Shortcomings of average linkage

Average linkage has its own problems:

- Unlike single and complete linkage, average linkage *doesn't give us a nice interpretation* when we cut the dendrogram
- Results of average linkage clustering can change if we simply apply a monotone increasing transformation to our dissimilarity measure, our results can change
  - E.g., $d \to d^2$ or $d \to \frac{e^d}{1+e^d}$
  - This can be a big problem if we're not sure precisely what dissimilarity measure we want to use
  - Single and Complete linkage do not have this problem

Average linkage monotone dissimilarity transformation



The left panel uses $d(x_i, x_j) = \|x_i - x_j\|_2$ (Euclidean distance), while the right panel uses $\|x_i - x_j\|_2^2$. The left and right panels would be same as one another if we used single or complete linkage. For average linkage, we see that the results can be different.

[source: http://imagicdigital.com/, Mark & Danya Henninger]

Suppose we wanted to place cell towers in a way that ensures that no building is more than 3000ft away from a cell tower. What linkage should we use to cluster buildings, and where should we cut the dendrogram, to solve this problem?

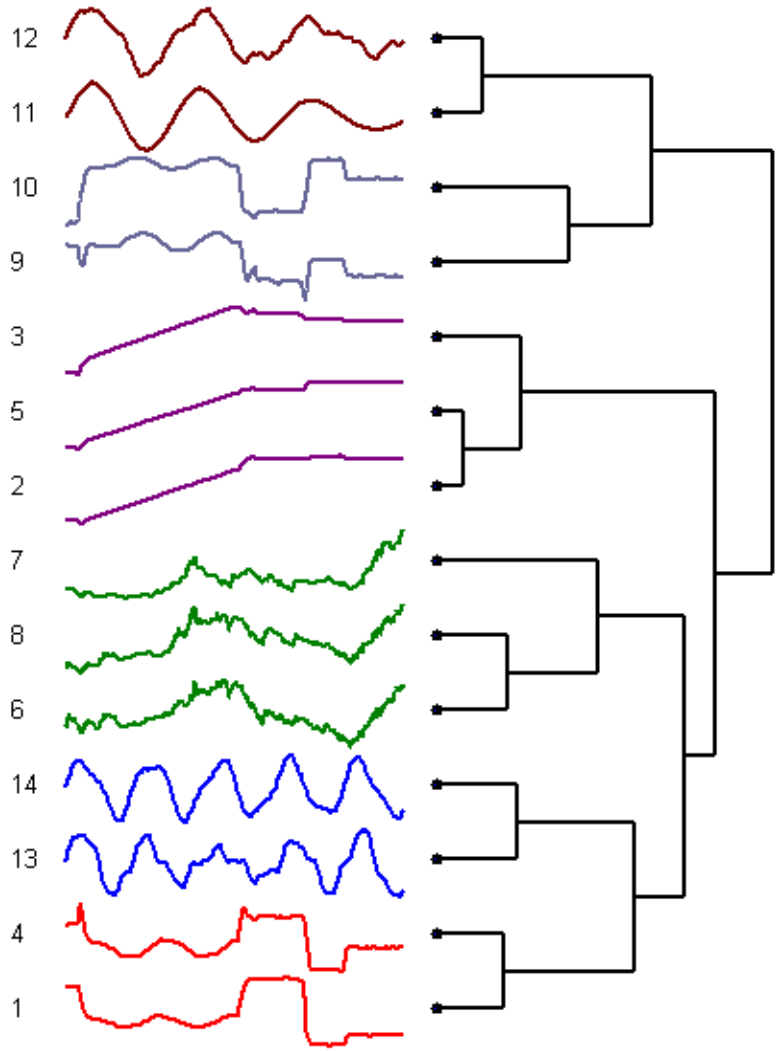
Dissimilarity measures

- The choice of linkage can greatly affect the structure and quality of the resulting clusters
- The choice of dissimilarity (equivalently, similarity) measure is *arguably* even more important
- To come up with a similarity measure, you may need to think carefully and use your intuition about what it means for two observations to be similar. E.g.,
  - What does it mean for two people to have similar *purchasing behaviour*?
  - What does it mean for two people to have similar music listening habits?
- You can apply hierarchical clustering to any similarity measure $s(x_i, x_j)$ you come up with. The difficult part is coming up with a good similarity measure in the first place.

Example: Clustering time series

Here's an example of using hierarchical clustering to cluster time series.

You can quantify the similarity between two time series by calculating the correlation between them. There are different kinds of correlations out there.



[source: A Scalable Method for Time Series Clustering, Wang et al]

Association rules (Market Basket Analysis)

- Association rule learning has both a supervised and unsupervised learning flavour
- We didn't discuss the supervised version when we were talking about regression and classification, but you should know that it exists.
  - Look up: Apriori algorithm (Agarwal, Srikant, 1994)
  - In **R**: `apriori` from the `arules` package
- Basic idea: Suppose you're consulting for a department store, and your client wants to better understand patterns in their customers' purchases
- patterns or rules look something like:

$$\{\texttt{suit}, \texttt{belt}\} \Rightarrow \{\texttt{dress shoes}\}$$

$$\underbrace{\{\texttt{bath towels}\}}_{\text{LHS}} \Rightarrow \underbrace{\{\texttt{bed sheets}\}}_{\text{RHS}}$$

  - In words: People who buy a new `suit` and `belt` are more likely to also by `dress shoes`.
  - People who by `bath towels` are more likely to buy `bed sheets`

Basic concepts

- Association rule learning gives us an automated way of identifying these types of patterns
- There are three important concepts in rule learning: support, confidence, and lift
- The support of an *item* or an *item set* is the fraction of transactions that contain that item or item set.
  - We want rules with high support, because these will be applicable to a large number of transactions
  - {`suit`, `belt`, `dress shoes`} likely has sufficiently high support to be interesting
  - {`luggage`, `dehumidifer`, `teapot`} likely has low support
- The confidence of a rule is the probability that a new transaction containing the LHS item(s) {`suit`, `belt`} will also contain the RHS item(s) {`dress shoes`}
- The lift of a rule is

$$\frac{\text{support(LHS, RHS)}}{\text{support(LHS)} \cdot \text{support(RHS)}} = \frac{\mathbb{P}(\{\texttt{suit}, \texttt{belt}, \texttt{dress shoes}\})}{\mathbb{P}(\{\texttt{suit}, \texttt{belt}\})\mathbb{P}(\{\texttt{dress shoes}\})}$$

An example

```
> a_list<-list(
+   c("CrestTP","CrestTB"),
+   c("OralBTB"),
+   c("BarbSC"),
+   c("ColgateTP","BarbSC"),
+   c("OldSpiceSC"),
+   c("CrestTP","CrestTB"),
+   c("AIMTP","GUMTB","OldSpiceSC"),
+   c("ColgateTP","GUMTB"),
+   c("AIMTP","OralBTB"),
+   c("CrestTP","BarbSC"),
```

- A subset of drug store transactions is displayed above
- First transaction: Crest ToothPaste, Crest ToothBrush
- Second transaction: OralB ToothBrush
- etc…

[source: Stephen B. Vardeman, STAT502X at Iowa State University]

```
> rules<-apriori(trans,parameter=list(supp=.02, conf=.5, target="rules"))

parameter specification:
 confidence minval smax arem  aval originalSupport support minlen maxlen target   ext
        0.5    0.1    1 none FALSE            TRUE    0.02      1     10  rules FALSE
```

- This says: Consider only those rules where the item sets have support *at least* 0.02, and confidence *at least* 0.5
- Here's what we wind up with

```
> inspect(head(sort(rules,by="lift"),n=20))
  lhs                rhs           support confidence     lift
1 {GilletteSC} => {ColgateTP}      0.03  1.0000000 20.00000
2 {ColgateTP}  => {GilletteSC}     0.03  0.6000000 20.00000
3 {CrestTB}    => {CrestTP}        0.03  0.7500000 15.00000
4 {CrestTP}    => {CrestTB}        0.03  0.6000000 15.00000
5 {GUMTB}      => {AIMTP}          0.02  0.6666667 13.33333
```

[source: Stephen B. Vardeman, STAT502X at Iowa State University]

Acknowledgements

All of the lectures notes for this class feature content borrowed with or without modification from the following sources:

- 36-462/36-662 Lecture notes (Prof. Tibshirani, Prof. G'Sell, Prof. Shalizi)
- 95-791 Lecture notes (Prof. Dubrawski)
- *An Introduction to Statistical Learning, with applications in R* (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani
- *Applied Predictive Modeling*, (Springer, 2013), Max Kuhn and Kjell Johnson