Summary and Excerpts of 《An Introduction to Statistical Learning》

Theoretical Background

What is the statistical model to be learned?

Suppose we observe a quantitative response variable Y and p different predictors $X_1, X_2 \ldots X_p$. We assume that there is some relationship between Y and $\boldsymbol{X}$ which can be written in a general form

$$Y = f(\boldsymbol{X}) + \varepsilon$$

- where $f(\boldsymbol{X})$ is a fixed but unknown function of $X_1, X_2 \ldots X_p$; and ε is a random error term.
- Also $\boldsymbol{X} \perp \varepsilon$ and $E(\varepsilon) = 0$

Classifying the models into parametric vs. non-parametric

Our goal is to apply a statistical learning method to the training data in order to estimate the unknown function f. Broadly speaking, most methods can be classified into either *parametric* or *non-parametric* method.

- Parametric model involves a two-step model-based approach:
  - First, we make an assumption about the functional form of f. For example, we can assume that f takes a linear form:
  $$f(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$
  - Second, after a model has been selected, we need a procedure that uses the training data to fit or train the model. The goal is to find values for the parameters $\beta_0\ \beta_1\ \ldots\ \beta_p$.

  Pros: The parametric approach reduces the problem of estimating f down to estimating a set of parameters.

  Cons: The model we choose will usually not match the true unknow form of f.
- Non-parametric model does not make explicit assumption about the functional form of f. They seek an estimate of f that gets as close to the data points as possible without being too rough or wiggly.

  Pros: Without explicit assumptions on f, they have the potential to accurately fit a wider range of possible shapes for f.

  Cons: They do not simplify the problem to estimating a set of parameters, so a very large number of observations is typically required in order to obtain an accurate estimate for f.

How can f be useful?

- Prediction:

  We can use the $\hat{f}(\boldsymbol{X})$ and X_{new} to make predictions on Y_{new} for previously unseen cases. The accuracy of $\hat{Y}$ depends on two quantities—*reducible error* and *irreducible error*.

To conceptualize these two quantities, let's consider a simulation setting. If 10 predictors $\boldsymbol{X}$ and an error term ε are specified to simulate the outcome variable Y, then the reducible error is the prediction errors that can be eliminated by including more predictors $\boldsymbol{X}$: Including 8 (out of 10) predictors would make less erroneous prediction than including 4 (out of 10) predictors. But the irreducible error comes from ε. Even if all 10 predictors were included in the model, there is still going to be leftover error that cannot be explained.

See the equations below for a formal breakdown of the prediction error, assuming that both $\boldsymbol{X}$ and $\hat{f}()$ *are fixed*:

Figure 1.

$$E(Y-\hat{Y})^2 = E\left[f(x)+\varepsilon-\hat{f}(x)\right]^2$$

$$= E\left[(f(x)-\hat{f}(x))^2+\varepsilon^2-2\varepsilon(f(x)-\hat{f}(x))\right]$$

$$= E\left[(f(x)-\hat{f}(x))^2\right] + E(\varepsilon^2) - 2E\left[\varepsilon(f(x)-\hat{f}(x))\right]$$

$2E[\varepsilon(f(x)-\hat{f}(x))] = 0$

X and f are considered fixed

$(f(x)-\hat{f}(x))^2$

$V(\varepsilon) = E(\varepsilon^2) - (E(\varepsilon))^2$

$\because E(\varepsilon) = 0$

$\therefore E(\varepsilon^2) = V(\varepsilon)$

bc ε is i.i.d and doesn't depend on X or $f(x)$ or $f(x)-\hat{f}(x)$

$$= (f(x)-\hat{f}(x))^2 + V(\varepsilon)$$

Reducible error

Irreducible error.

$\hat{f}(x)$ should be able to get closer to $f(x)$ if we specify the model correctly and include the right amount of X

- Inference
  We wish to understand the $f(\boldsymbol{X})$ model.

Accessing model accuracy

Different $f(\boldsymbol{X})$ are useful for dealing with different data structure. There are a few key concepts in selecting statistical learning models:

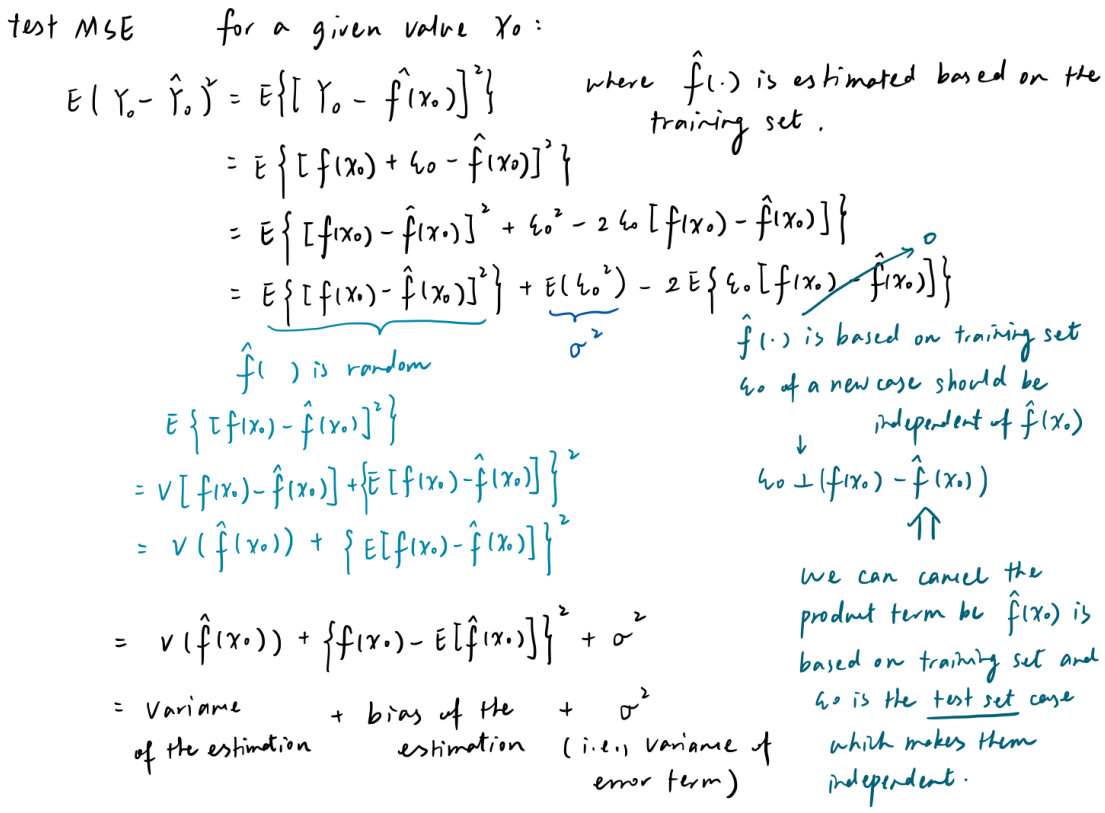
- **Quality of fit**
  In the *regression setting* (i.e., continuous outcome variable), the most commonly-used

measure is the mean squared error (MSE), given by the averaged squared difference between the observed Y value and the estimated $\hat{Y}$ value:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}[y_i - \hat{f}(x_i)]^2$$

It is important to differentiate between training MSE and test MSE.
In the standard statistical modeling procedure, oftentimes only the training MSE is considered. In fact, many statistical methods are specifically designed to estimate coefficients so as to minimize the training MSE, but there is no guarantee that a model that minimize the training MSE generalize well. Therefore, we want to choose the method that gives the lowest test MSE, as opposed to the lowest training MSE.

- **Bias-Variance trade-off**
  Expectation of test MSE can be broken down to the sum of bias and variance, as specified below, where the Expectation E refers to theoretically averaging across different training sets:
  Figure 2.



- *Variance* refers to the amount by which $\hat{f}$ would change if we estimated it using a different training set. Since the training data are used to fit the statistical learning model, different training datasets will result in different $\hat{f}$.

- Bias on the other hand, refers to the difference between the mean of the predictions and the true value. Recall the simulation setting introduced above, $f(\boldsymbol{x_0})$ refers to the value of Y as a function of the 10 predictors without the error terms. In contrast, $\hat{f}(x_0)$ depends on the model we fit, the $\boldsymbol{x_0}$ doesn't even need to be the 10 variables, can be 5 out of the 10 variables, or other virtually irrelevant things that we include.
- As a general rule, as we use more flexible methods, the variance will increase and the bias will decrease. The relative rate of change of these two quantities determines whether the test MSE increase or decrease.

The derivation in Figure 2 presents the expected test MSE of one case in the test set. Overall expected MSE can be computed by averaging over all possible values of x_0 in the test set.

- **The classification setting**

  The most common approach for quantifying the accuracy of our estimate $\hat{f}$ is the *training error rate*

$$\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i)$$

  where $\hat{y}_i$ is the predicted class label for the ith observation using $\hat{f}$.

  The test error rate associated with a set test observation of the form (x_0, y_0) is given by

$$Ave\big(I(y_0 \neq \hat{y}_0)\big)$$

  A good classifier should be one that gives the smallest test error.

  Bayes Classifier

  Hypothetically, if we know the conditional distribution $Y|\boldsymbol{X}$, then a very simple classifier that assigns each observation to its most likely class, given its predictor values, can minimize the test error rate. In other word, assign a test observation with predictor vector x_0 to the class j for which

$$P(Y = j|X = x_0)$$

  is the largest. This is *Bayes Classifier*. This is a theoretical classifier based on the assumption that we know the conditional distribution of Y given $\boldsymbol{X}$. It can be obtained, if, say, we simulate the data ourselves based on the joint distribution of Y and X. But in reality, we obviously do not know about this distribution. Later, we will discuss how to empirically estimate this conditional distribution based on training data.

  The Bayes classifier produces the lowest possible test error rate, called Bayes error rate:

$$1 - E_X(max\ P(Y = j|X))$$

  where the expectation averages the probability over all possible values of X.

KNN

KNN classifier

KNN Classifier is one simple method to estimate the conditional distribution of $Y|\boldsymbol{X}$. Given an integer K, for a test observation x_0, the classification can be made by utilizing the probabilities

$$P(Y = j|X = x_0) = \frac{1}{K}\sum_{i\epsilon S_0} I(y_i = j)$$

where S_0 denotes the closest K observations around the test observation. Then this observation is assigned to the class j that is associated with the largest probability.

- The choice of K has a drastic influence on the KNN classifier obtained. As K grows, the method becomes less flexible and produces a decision boundary that is close to linear. This corresponds to a low-variance-high-bias classifier.
- The choice of distance function that is used to identified the K-nearest neighbors also influences the KNN classifier. Examples of the distance function are Euclidian distance, Manhattan distance, Mahalanobis distance etc.

KNN regression

KNN regression is a version of KNN that deals with continuous outcome variables. Given an integer K, for a test observation x_0, the regression model estimates $f(x_0)$ using the averages of all the training responses in S_0—the K nearest neighbors of the test observation x_0. In other words,

$$\hat{f}(x_0) = \frac{1}{K}\sum_{i\epsilon S_0} y_i$$

The consideration of K in the context of KNN regression is the same as it in the context of KNN classifier.

Comparing KNN with Linear Regression

I have made extensive summary on linear regression and will not repeat it here. Let's directly jump into the comparison between KNN and linear regression.

Linear regression is a parametric model and KNN is a phenotypical non-parametric model. *In what setting will a parametric approach outperforms a non-parametric approach?*

- The answer is straightforward: the parametric approach will outperform the non-parametric approach *if the parametric form that has been selected is close to the true form of f*.

  For example, if the underlying relationship is linear, then KNN cannot outperform an OLS because non-linearity incurs a cost in variance that is not offset by a reduction in bias. However, in low-dimensional setting (few predictors), if K is set sufficiently large, then KNN is only slightly worse than OLS.
- *In higher dimensions, KNN often performs worse* than linear regression even if the underlying relationship is not linear.

The decrease in performance as the dimension increases is a common problem for KNN. This results from the fact that in higher dimensions there is effectively a reduction in sample size. There is no nearby neighbors for a given observation. This is the so-called curse of dimensionality.

Logistic Regression

Classify 0-1 binary outcome variable. I have summarized logistic regression elsewhere and will not repeat this model here.
I only note one point: Logistic regression models have multiple-class extensions, but in practice they tend not to be used that often.

Linear Discriminant Analysis

What is the advantage of LDA comparing to logistic?

- When the classes are well-separated, logistic regression model are unstable, whereas LDA does not suffer from this problem.
- If n is small and the distribution of the predictors X is approximately normal in each of the classes, the LDA is again more stable.
- LDA is popular for dealing with multinomial outcome variables.

Logistic regression directly models $P(Y = k|X = x)$ using the logistic function; that is, modeling the conditional distribution of the response Y given the predictors X.
In contrast, in LDA model, we model the conditional distribution of X given Y: $f(X|Y = k)$; then Bayes' theorem is used to flip the conditional distribution of $X|Y$ to estimate the probabilities of $Y = k|X$.

- Let π_k represents the overall probability that a randomly chosen observation comes from the kth class $\pi_k = P(Y = k)$. This is the ***prior*** probability. If there is no other information, we would make prediction simply based such relative probabilities of the k classes.
- Let $f_k(x) = P(X = x|Y = k)$ denote the density function of X for an observation that comes from the kth class. *This is the model of LDA*.
- Using Bayes' theorem, the probability can be flipped

$$\begin{aligned} & P(Y = k|X = x) \\ &= \frac{P(X = x|Y = k)P(Y = k)}{P(X = x)} \\ &= \frac{P(X = x|Y = k)P(Y = k)}{\sum_{l=1}^{K} P(X = x|Y = l)P(Y = l)} \\ &= \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)} \end{aligned}$$

  now we have a ***posterior*** probability of Y conditioning on the observed X. We achieve the same outcome as the logistic regression, but with a different model specification.

The key of LDA is the assumption/model of $f_k(x) = P(X = x|Y = k)$, and in LDA, normal distribution is assumed.

First, consider first a simple setting where there is only one predictor X. Conditioning on $Y = k$, the distribution of $X|Y = k$ is a univariate normal distribution:

$$f_k(x) = \frac{1}{\sqrt{2\pi{\sigma_k}^2}} \exp\left(-\frac{(x-\mu_k)^2}{2{\sigma_k}^2}\right)$$

where μ_k and ${\sigma_k}^2$ are the mean and variance parameter of the kth class. For simplicity, assuming that ${\sigma_1}^2 = {\sigma_2}^2 = \cdots = {\sigma_k}^2$: that is, there is a shared variance term across all K classes.
Plugging this model into the posterior probability above

$$P(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)} = \frac{\pi_k \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu_k)^2}{2\sigma^2}\right)}{\sum_{l=1}^{K} \pi_l \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu_l)^2}{2\sigma^2}\right)}$$

For a given case x_0, the goal is to assign the case to the class k which has the maximum $P(Y = k|X = x_0)$. Two tricks are involved here:

a. For the k classes, the denominators are the same, so whichever class that maximizes the numerator would has the largest probability.
b. Since log is an increasing function, the class that has the largest P would also have the largest $\log(P)$. Taking log of both sides:

$$\log[P(Y = k|X = x)] = \log\left[\frac{\pi_k \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu_k)^2}{2\sigma^2}\right)}{\sum_{l=1}^{K} \pi_l \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu_l)^2}{2\sigma^2}\right)}\right]$$

To maximize the numerator $\log\left[\pi_k \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu_k)^2}{2\sigma^2}\right)\right]$

$= \log \pi_k + \underbrace{\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)}_{\text{same for all K classes}} - \frac{(x-\mu_k)^2}{2\sigma^2}$

is to maximize $\log \pi_k - \frac{(x-\mu_k)^2}{2\sigma^2}$

$= \log \pi_k - \underbrace{\frac{x^2}{2\sigma^2}}_{\text{same for all K classes}} - \frac{\mu_k^2}{2\sigma^2} + \frac{2x\mu_k}{2\sigma^2}$

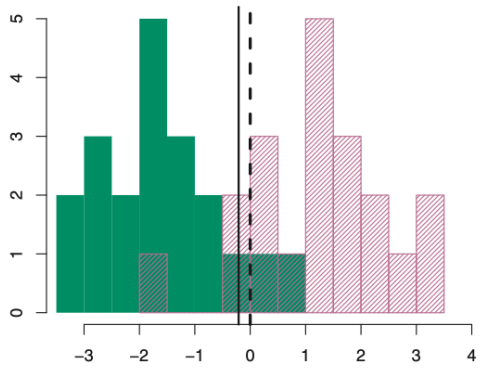
is to maximize $\log \pi_k - \frac{\mu_k^2}{2\sigma^2} + \frac{x\mu_k}{\sigma^2}$

If $K = 2$ and the two classes are balanced in size (i.e., $\pi_1 = \pi_2$), then the maximization problem above can be further reduced to $\frac{-{\mu_k}^2}{2\sigma^2} + \frac{x\mu_k}{\sigma^2}$. The decision

boundary will be given by the x that makes $\frac{-{\mu_1}^2}{2\sigma^2} + \frac{x\mu_1}{\sigma^2} = \frac{-{\mu_2}^2}{2\sigma^2} + \frac{x\mu_2}{\sigma^2}$:

$$x = \frac{\mu_1 + \mu_2}{2}$$

Here is an example showing the decision boundary that separates two normal distributions. The solid line is the decision boundary of LDA and the dashed line is the decision boundary of Bayes classifier based on the simulation set up (can be interpreted as the "truth"):



With the model specified as above, how to estimate in practice?
For the model of the conditional normal distribution of X given Y:

$$f_k(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu_k)^2}{2\sigma^2}\right)$$

μ_k and σ^2 are the parameters, both of which as straightforward to estimate.

μ_k is simply the average of all the training observations from the kth class: $\mu_k = \frac{1}{n_k}\sum_{x_i \in \, class\ k} x_i$

σ^2 can be seen as a weighted average of the sample variances for each of the K classes $\sigma^2 = \frac{1}{n-K}\sum_{k=1}^{K}\sum_{x_i \in \, class\ k}(x_i - \hat{\mu}_k)^2$

In terms of the prior probability of class K, π_k, we can either use external information on the marginal distribution of Y; or we can estimate the probabilities of each of the class from the training observation using

$$\hat{\pi}_k = \frac{n_k}{n}$$

Second, in regular cases we have multiple predictors ($\boldsymbol{X} = (X_1\ X_2\ \ldots\ X_p)$). Conditioning on $Y = k$, the distribution of $\boldsymbol{X}|Y = k$ can be modeled as a multivariate normal distribution $N_p(\boldsymbol{\mu_k}, \Sigma)$, with class-specific mean vector $\boldsymbol{\mu_k}$ and a common covariance matrix Σ.
The multivariate density function is given by

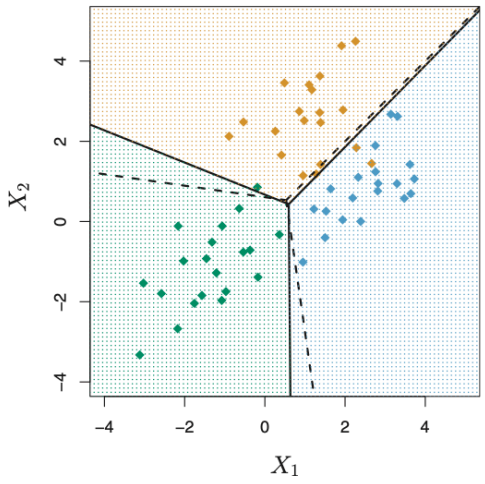
$$f(\boldsymbol{X}|Y) = \frac{1}{(2\pi)^{-\frac{p}{2}}\Sigma^{-\frac{1}{2}}} \exp\left[-\frac{1}{2}(\boldsymbol{X}-\boldsymbol{\mu_k})'\Sigma^{-1}(\boldsymbol{X}-\boldsymbol{\mu_k})\right]$$

Again, we plug in this model to obtain the posterior distribution $P(Y = k|\boldsymbol{X} = \boldsymbol{x}) = \frac{\pi_k f_k(\boldsymbol{x})}{\sum_{l=1}^{K}\pi_l f_l(\boldsymbol{x})}$.

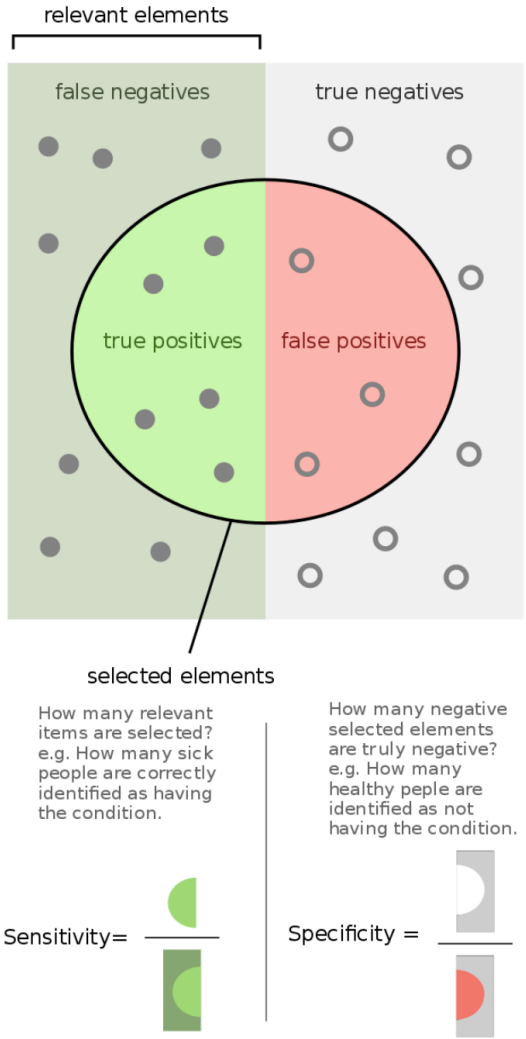
For a given observation $\boldsymbol{x_0}$, when we try to assign it to a class, we look for the maximum $P(Y = k|\boldsymbol{X} = \boldsymbol{x_0})$. The algebraic process is slightly more complicated, but the logic is the same as the derivation above. To maximize $P(Y = k|\boldsymbol{X} = \boldsymbol{x_0})$ is to maximize

$$\boldsymbol{x_0}'\Sigma^{-1}\boldsymbol{\mu_k} - \frac{1}{2}\boldsymbol{\mu_k}'\Sigma^{-1}\boldsymbol{\mu_k} + log\pi_k$$

Here is an example showing the classification of three classes using two predictors X_1 X_2. The decision boundary carves out the area fallen within which the cases will be classified as either green, orange, or blue. The solid line is the LDA decision boundary and the solid line is the Bayes classifier based on the simulation set up (can be interpreted as the "truth"):



Specificity vs. Sensitivity



Assuming that $K = 2$.

The default setting is that if $P(Y = 1|X) \geq 0.5$, then the observation is classified as 1; if $P(Y = 0|X) < 0.5$, then the observation is classified as 0.

However, we do not have to use the 0.5 as the cutoff. By changing the cutoff to other values, we can modify the specificity and sensitivity of the model.

For example, if the cutoff point is 0.2 rather than 0.5; that is, $(Y = 1|X) \geq 0.2$. Then a lot more cases are going to be classified as positive (= the circle in the left figure becoming larger), the sensitivity is increased and specificity decreases.

Quadratic discriminant analysis

The LDA models discussed above are referred to as "linear" model because the term that we wish to maximize to determine class assignment—$[\boldsymbol{x_0'}\Sigma^{-1}\boldsymbol{\mu_k} - \frac{1}{2}\boldsymbol{\mu_k'}\Sigma^{-1}\boldsymbol{\mu_k} + log\pi_k]$ for multiple predictors—is a linear function of $\boldsymbol{x}$. This linearity comes from assuming a common variance or covariance matrix for all K classes. If this assumption is not made and we allow each class to has its own covariance matrix $N_p(\boldsymbol{\mu_k}, \Sigma_k)$, then the model is quadratic discriminant analysis (QDA). The logic of QDA is similar to LDA, the classifier involves estimating $\boldsymbol{\mu_k}, \Sigma_k$ and π_k and use them to calculate the posterior probability of $P(Y = k|\boldsymbol{X})$. To maximize the probability is to maximize the formula below

$$\begin{aligned}\delta_k(x) &= -\frac{1}{2}(x-\mu_k)^T\boldsymbol{\Sigma}_k^{-1}(x-\mu_k) + \log\pi_k \\ &= -\frac{1}{2}x^T\boldsymbol{\Sigma}_k^{-1}x + x^T\boldsymbol{\Sigma}_k^{-1}\mu_k - \frac{1}{2}\mu_k^T\boldsymbol{\Sigma}_k^{-1}\mu_k + \log\pi_k\end{aligned}$$

QDA is a model that estimates more parameters since it allows Σ_k to vary across K classes. Not surprisingly, QDA is a more flexible model and thus the choice between LDA and QDA echoes the discussion on bias-variance trade-off.

Comparison of classification models

- LDA vs. Logistic regression
  Even though the theoretical setups behind these two models are different, the end results of them are actually very similar. Both ended up modeling $P(Y = k|\boldsymbol{X})$; both specifies a linear function of the predictors $\boldsymbol{X}$.
  Consider the simple case when $K = 2$ and there is only one predictor X.
  - For LDA:
    $$\log\left(\frac{p_1(x)}{p_2(x)}\right) = \log\left[\frac{\pi_1 \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(x-\mu_1)^2}{2\sigma^2}\right)}{\pi_2 \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(x-\mu_1)^2}{2\sigma^2}\right)}\right]$$
    $$which\ can\ be\ simplified\ to\ the\ form$$
    $$= c_0 + c_1 x$$
  - For logistic regression, it is the familiar
    $$\log\left(\frac{p_1(x)}{p_2(x)}\right) = \beta_0 + \beta_1 x$$

  That is to say, the only difference between the two approaches is that LDA uses the estimated mean and variance of $X|Y = k$ to estimate c_0 c_1, whereas logistic regression uses maximum likelihood to estimate β_0 β_1.
  These two approaches, however, do not always give the same results in practice. LDA outperforms logistic regression when the normal assumption of X is met. Otherwise, logistic regression outperforms LDA.
- KNN vs. logistic regression & LDA
  Both logistic regression and LDA are parametric approaches, whereas KNN is completely non-parametric. Thus, KNN can outperform logistic regression and LDA when the decision boundary is highly non-linear.
- QDA
  QDA is a compromise between KNN and logistic/LDA. It is more flexible than the two linear approach and can accurately model a wider range of problem. But comparing to KNN, the procedure of QDA can still be summarized as estimating a set of parameters.

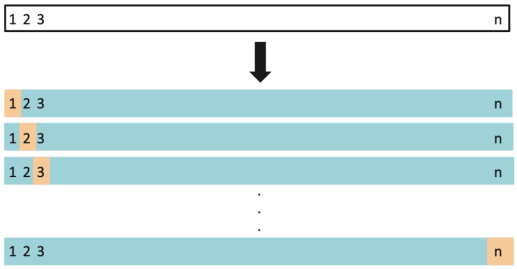
Resampling

Validation set approach

Randomly dividing the available set of observations into two parts, a training set and a validation set. The model is fit on the training set and the fitted model is used to predict the responses for the observations in the validation set.

Leave-one-out cross-validation

A single observation is used for the validation set$(\boldsymbol{x_1}, y_1)$ and the remaining observations $\{(\boldsymbol{x_2}, y_2)$ ... $(\boldsymbol{x_n}, y_n)\}$ make up the training set. The statistical learning model is fit on the $n-1$ training observations and a prediction $\hat{y}_1$ is made for the excluded observation. A MSE can be calculated as $MSE_1 = (y_1 - \hat{y}_1)^2$. MSE_1 is an unbiased estimate for the test error. [What does "unbiased" estimate for the test error mean? What is the "true value" of test error?] But MSE_1 is highly variable because it depends on a single observation. Repeating this process for every case results in n estimates of MSE_i.



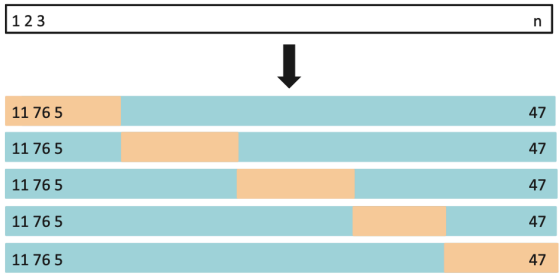
The LOOCV estimate for the test MSE is the average of these n test error estimates

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i$$

Pros: LOOCV has far less bias. There is no randomness in the training/validating set splits.
Cons: If n is large, then LOOCV is potentially expensive to implement.

k-Fold cross-validation

Randomly dividing the set of observations into k groups, or folds, of approximately equal size. The first fold is first treated as a validation set, and the method is fit on the remaining k-1 fold. The mean squared error MSE_1 can then be computed on the observations in the held-out fold. Repeating this process for the k folds result in k estimates of MSE_i.



The k-fold CV estimate is computed by averaging these values

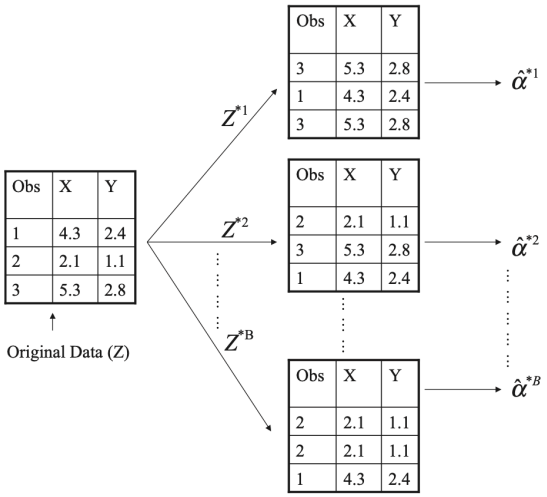
$$CV_{(k)} = \frac{1}{k}\sum_{k=1}^{n} MSE_k$$

Note. When using cross-validation methods to perform parameter selection, the location of the minimum point in the estimated test MSE curve is important, but the actual value of the estimated test MSE is not.

Bias-variance trade-off for k-fold cross-validation

Bootstrap

We treat the original dataset as the population and repeated sample observations from the original dataset. The sampling is performed with replacement, and thus the re-sampled samples can be as large as we wish. Bootstrapping is commonly used to estimate standard error of statistics.



where $Z^{*1}, Z^{*2} \ldots Z^{*B}$ denotes the bootstrapped datasets corresponding to the estimates $\hat{\alpha}^{*1}, \hat{\alpha}^{*2} \ldots \hat{\alpha}^{*B}$. The variability across these estimates give the standard error of these bootstrap estimates using the formula

$$ar(\hat{\alpha}) = \frac{1}{B-1}\sum_{r=1}^{B}(\hat{\alpha}^{*r} - \frac{1}{B}\sum_{r'=1}^{B}\hat{\alpha}^{*r'})^2$$

In the setting of bootstrapping with replacement, even though there are overlaps between samples, the samples are *not* correlated because the sampling of each case is independent. Composition of one sample does not inform the composition of other samples.

Linear model selection – subset selection

If we have p predictors, how to select a model that contains only some of the most informative predictors?

Procedures

- Best subset selection

Algorithm 6.1 *Best subset selection*

1. Let $\mathcal{M}_0$ denote the *null model*, which contains no predictors. This model simply predicts the sample mean for each observation.
2. For $k = 1, 2, \ldots p$:
   (a) Fit all $\binom{p}{k}$ models that contain exactly k predictors.
   (b) Pick the best among these $\binom{p}{k}$ models, and call it $\mathcal{M}_k$. Here *best* is defined as having the smallest RSS, or equivalently largest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

This is a conceptually appealing approach, which ends up with the best subset of predictors that generalize well to the test data. But it is computationally infeasible. The number of models to be fitted increases quickly as p increase.
The computational feasible alternatives are:

- Forward stepwise selection

Algorithm 6.2 *Forward stepwise selection*

1. Let $\mathcal{M}_0$ denote the *null* model, which contains no predictors.
2. For $k = 0, \ldots, p-1$:
   (a) Consider all $p - k$ models that augment the predictors in $\mathcal{M}_k$ with one additional predictor.
   (b) Choose the *best* among these $p - k$ models, and call it $\mathcal{M}_{k+1}$. Here *best* is defined as having smallest RSS or highest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Predictors are included in the model one-at-a-time. Each step is built on top of the previously obtained model. In each step, the procedure include the best predictor among the remaining predictors into the model.
Though forward stepwise tends to work well in practice, it does not guarantee to find the best subset out of all possible subsets of the p predictors.
Forward selection can handle situation when $p > n$. It will just select no more than n predictors from all p predictor candidates.

- Backward stepwise selection

Algorithm 6.3 *Backward stepwise selection*

1. Let $\mathcal{M}_p$ denote the *full* model, which contains all p predictors.
2. For $k = p, p-1, \ldots, 1$:
   (a) Consider all k models that contain all but one of the predictors in $\mathcal{M}_k$, for a total of $k-1$ predictors.
   (b) Choose the *best* among these k models, and call it $\mathcal{M}_{k-1}$. Here *best* is defined as having smallest RSS or highest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Reversing the logic of forward selection and excluding the worst predictor at each step. Backward selection requires $p \leq n$, otherwise it cannot start with fitting a full model.

- Hybrid approach
  Combining forward selection and backward selection. After adding each new variable, the method may also remove any variables that no longer provide an improvement in the model. This approach mimic best subset selection while retaining the computational advantages of forward and backward stepwise selection.

How to choose the optimal model

In the three algorithms above, the third steps all talk about selecting a single best model. What criteria do the selection based on?

The goal of the selection is to select the model that has the smallest test error. But the models are built on training data by minimizing training error, so how to get to the test error? There are two general approach:

1) We can indirectly estimate test error by making an adjustment to the training error to account for the bias due to overfitting. The C_p, AIC, BIC, and Adjusted R^2 all fall under this approach.
   a. $C_p = \frac{1}{n}(RSS + 2d\hat{\sigma}^2)$
      where d refers to the number of predictors.
      C_p add a penalty to the training RSS. If $\hat{\sigma}^2$ is an unbiased estimate of σ^2, then C_p would be an unbiased estimate of test MSE.
   b. $AIC = \frac{1}{n\hat{\sigma}^2}(RSS + 2d\hat{\sigma}^2)$
   c. $BIC = \frac{1}{n\hat{\sigma}^2}(RSS + \ln(n)d\hat{\sigma}^2)$
      BIC places a heavier penalty on models with many variables.
   d. $Adjusted\ R^2 = 1 - \frac{\frac{RSS}{n-d-1}}{\frac{TSS}{n-1}}$ (In contrast, $R^2 = 1 - \frac{RSS}{TSS}$)
      The intuition behind $Adjusted\ R^2$ is that if including additional variables does not improve the model fit substantively (d goes up but RSS stays similar), then the $Adjusted\ R^2$ would decrease. Thus the $Adjusted\ R^2$ is measuring whether the decrease in RSS is worth the extra predictors.

2) We can directly estimate the test error, using the validation methods.
In the past, performing cross-validation was computationally prohibitive and thus the C_p, AIC, BIC, and Adjusted R^2 are preferred. But nowadays, cross validation is an attractive approach to directly get to the test error.

Linear model regularization – shrinkage

OLS minimize least squares

$$RSS = \sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \boldsymbol{X_i}\widehat{\boldsymbol{\beta}})^2$$

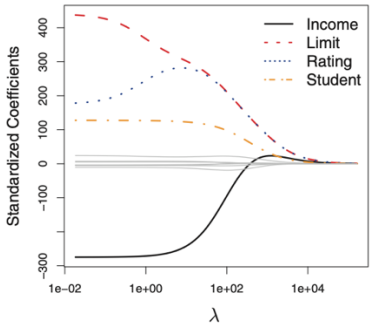
where $\hat{\beta}_0$ is the predicted intercept and $\widehat{\boldsymbol{\beta}}$ are the predicted coefficients.

Ridge Regression

Ridge Regression minimizes

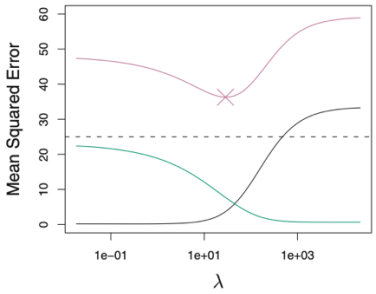
$$\sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \boldsymbol{X_i}\widehat{\boldsymbol{\beta}})^2 + \lambda\sum_{j=1}^{p}\hat{\beta}_j{}^2$$

where λ is a turning parameter, to be determined separately. Ridge regression seeks coefficients that make RSS small, while also trying to shrink the magnitude of $\hat{\beta}_j$ so that the penalty term $\lambda\sum_{j=1}^{p}\hat{\beta}_j{}^2$ can be small. λ controls the relative impact of these two terms. When $\lambda = 0$, ridge regression simplifies to OLS; when $\lambda \to large$, all coefficient would be 0. Note that the shrinkage penalty only applies to the predictor coefficient $\hat{\beta}_1, \hat{\beta}_2 \ldots \hat{\beta}_p$ but not to the intercept $\hat{\beta}_0$. Here is an example of how the coefficients converge to zero as λ becomes large:



- **! scale equivalent**
  One difference between OLS and ridge regression is that OLS is scale equivalent, but ridge regression is scale dependent. By scale equivalent, it means that changing the scaling of the predictors does not influence the OLS model fit, because the coefficient simply changes proportionally along with the scale of the predictors. In contrast, in the case of ridge regression, because the penalty term $\lambda\sum_{j=1}^{p}\hat{\beta}_j{}^2$ depends on the magnitude of the coefficients, scaling of the predictors can substantively influence the model fit. Therefore, it is best to standardize the predictor before applying ridge regression.

- **Why Ridge regression improves over OLS?**
  Ridge regression's advantage over OLS is rooted in the bias-variance trade-off. As λ increases, the flexibility of the model decreases (consider the extreme situation where $\lambda = 0$, then all coefficients would be zero. The model is simply the mean), and thus the variance decrease and the bias increases. This may permit the model to generalize better to test data if the increase in bias is offset by the decrease in variance.
  Here is an example showing that as λ increases, the bias (black) increases but the variance (green) first decreases to a larger extent, resulting in an decrease in the MSE. But later, the decrease in variance is not enough to compensate for the increase in bias and the MSE increases.



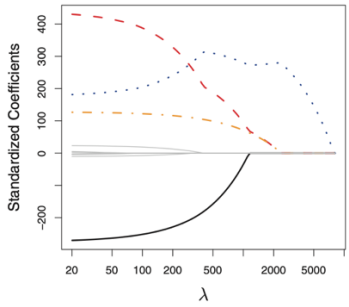
  In general, if the relationship between Y and X is linear, then OLS would have low bias, but may have high variance. This means that a small change in the training data can cause a high change in the coefficient estimates. Under such consideration, ridge regression works best in situation where the OLS estimates have high variance.

Lasso

Lasso has a similar logic as Ridge regression, but it minimize a different function

$$\sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \boldsymbol{X_i}\widehat{\boldsymbol{\beta}})^2 + \lambda \sum_{j=1}^{p} |\hat{\beta}_j|$$

The setup of lasso solves a major disadvantage of Ridge regression—while ridge regression shrinks the coefficients of the p predictors to zero, eventually none of the p coefficients will go to exactly zero, so ridge regression will not result in exclusion of any predictor.
In contrast, lasso force some of the coefficient estimates to be exactly zero when λ is sufficient large. That is, lasso can perform variable selection and yield sparse model. Here is an example showing how coefficients becomes zero as λ increases:



Why lasso can select variables but ridge regression cannot?

The minimization function of ridge regression and lasso can be rewritten as

$$minimize\left\{\sum_{i=1}^{n}\left(y_i-\hat{\beta}_0-\boldsymbol{X_i}\widehat{\boldsymbol{\beta}}\right)^2\right\}\ subject\ to\ \sum_{j=1}^{p}\hat{\beta}_j{}^2\le s$$

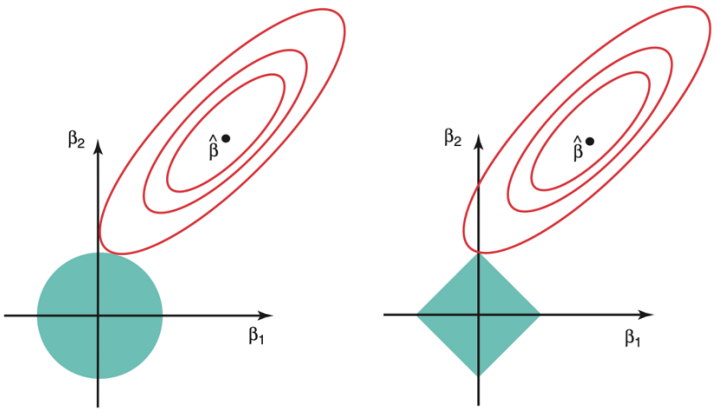
and

$$minimize\left\{\sum_{i=1}^{n}\left(y_i-\hat{\beta}_0-\boldsymbol{X_i}\widehat{\boldsymbol{\beta}}\right)^2\right\}\ subject\ to\ \sum_{j=1}^{p}|\hat{\beta}_j|\le s$$

respectively.

That is to say, the constraints imposed by λ can be rewritten in the form of s. For every value of λ, there is some s such that the two equations above would give the same coefficient estimates. Minimizing ridge and lasso functions is like minimizing RSS with a budge constraint (i.e., s) on $\hat{\beta}_j$.

The budge of ridge regression $\sum_{j=1}^{p}\hat{\beta}_j{}^2\le s$ can be geometrically expressed as a sphere/circle; and the budge of lasso $\sum_{j=1}^{p}|\hat{\beta}_j|\le s$ as graphed as a diamond. Here are the examples in a two-dimensional setting (two predictors $\hat{\beta}_1$ and $\hat{\beta}_2$)



The ellipse that are centered around $\widehat{\boldsymbol{\beta}}$—the OLS estimate that minimize RSS—represent regions of constant RSS (different combinations of $\hat{\beta}_1$ and $\hat{\beta}_2$ can result in the same RSS, but only one combination can minimize RSS). As the ellipse expands away from the least squares coefficient estimates, the RSS increases.

Ridge and lasso estimates are given by the point where the ellipse contacts the constraint region (green).

- Since ridge regression has a circular constraint with no sharp point, this intersection will not generally occur on an axis, and so the ridge regression coefficient estimates will be exclusively non-zero.
- The lasso constraint has corners at each of the axis, and so the ellipse will often intersect the constraint region at an axis, resulting in the coefficients equal to zero.

Comparing ridge regression and lasso

The relative advantage of ridge regression and lasso follows directly from their properties—lasso can performs variable selection; ridge regression shrinks coefficients in a relatively even way, all toward but not equal to zero.

- Lasso performs better in a setting where a relatively small number of predictors have substantial coefficients and the remaining predictors have coefficients that are very small or close to zero. With its variable selection property, the resulting lasso model is easier to interpret.
- Ridge regression perform better when the response is a function of many predictors, all with coefficients of roughly equal size.

Cross validation can be used to test which approach is better in practice.

Linear model – dimension reduction

Subset selection and shrinkage method both utilize the original predictors, $X_1\ X_2\ \dots\ X_p$. Dimension reduction is an approach that transform the p predictors and fit a OLS model using the transformed variables.

Let $Z_1\ Z_2\ \dots\ Z_M$ represents $M\ (< p)$ linear combinations of the original p predictors. That is

$$for\ the\ mth\ factor:\quad Z_m = \sum_{j=1}^{p} X_j \phi_{jm} = [X_1 \quad X_2 \quad \dots \quad X_p] \begin{bmatrix} \phi_{1m} \\ \phi_{2m} \\ \vdots \\ \phi_{pm} \end{bmatrix}$$

$$for\ all\ M\ factors:\quad Z = [X_1 \quad X_2 \quad \dots \quad X_p] \begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1M} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{p1} & \phi_{p2} & \cdots & \phi_{pM} \end{bmatrix}$$

for some constant $\phi_{1m}\phi_{2m} \dots \phi_{pm}$, $m = 1, 2, \dots M$.

Considering the transformation with respect to the shape of the dataset, where n denotes the number of observation (e.g., n respondents), and thus,

- X_{1n} indicates the $n$the person's response on variable X_1
- Z_{1n} indicates the $n$the person's score on the transformed variable Z_1

$$\boldsymbol{Z} = \underbrace{\begin{bmatrix} Z_{11} & \dots & Z_{M1} \\ Z_{12} & \dots & Z_{M2} \\ \vdots & \ddots & \vdots \\ \vdots & \ddots & \vdots \\ \vdots & \ddots & \vdots \\ \vdots & \ddots & \vdots \\ Z_{1n} & \dots & Z_{Mn} \end{bmatrix}}_{n*M} = \boldsymbol{X\phi} = \underbrace{\begin{bmatrix} X_{11} & \dots & \dots & X_{p1} \\ X_{12} & \dots & \dots & X_{p2} \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ X_{1n} & \dots & \dots & X_{pn} \end{bmatrix}}_{n*p} \underbrace{\begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1M} \\ \phi_{21} & \phi_{22} & \dots & \phi_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{p1} & \phi_{p2} & \dots & \phi_{pM} \end{bmatrix}}_{p*M}$$

We then fit the linear regression with Y regressing on $\boldsymbol{Z}$:

$$Y_i = \theta_0 + \boldsymbol{Z_i\theta} + \varepsilon_i, \qquad i = 1, 2 \dots n$$

where the regression coefficient $\boldsymbol{\theta}$ are $\theta_1, \theta_2 \dots \theta_M$.

Dimension reduction comes from the name that the original problem of estimating $p + 1$ coefficients (i.e., a $p + 1$ -dimension problem) is reduced to estimating $m + 1$ coefficients (i.e., a $m + 1$ -dimension problem).

$\boldsymbol{\theta}$ are the coefficients of $\boldsymbol{Z}$ on Y; $\boldsymbol{\beta}$ are the coefficients of **X** on Y

$$\mathrm{E}(\mathrm{Y}_i|\mathbf{Z}_i) = \mathbf{Z}_i\boldsymbol{\theta} = (\mathbf{X}_i\boldsymbol{\phi})\boldsymbol{\theta} = \mathbf{X}_i \underbrace{(\boldsymbol{\phi\theta})}_{\boldsymbol{\beta}_{constrained}}$$

Transforming the predictor **X** and then input the transformed variables into the regression is a re-parameterization of the regression. This transformation imposes a constraint on the effect of $\boldsymbol{X}$ on Y. Here is an example:

$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \varepsilon$ ①

if $Z_1 = \frac{1}{2} X_1 + \frac{1}{3} X_2$ ②

$Y = \theta_0 + \theta_1 Z_1$ ③

$= \theta_0 + \theta_1 (\frac{1}{2} X_1 + \frac{1}{3} X_2)$

$= \theta_0 + \frac{1}{2}\theta_1 X_1 + \frac{1}{3}\theta_1 X_2$

⇓

Coefficient of X_1 on Y is $\frac{1}{2}\theta_1$

Coefficient of X_2 on Y is $\frac{1}{3}\theta_1$

} different from the β_1 and β_2 in ①, the two coefficients can no longer freely take any value.

Principle component regression

Principle components analysis is a popular approach to derive a low-dimensional set of features from a large set of variables—finding an informative transformation matrix $\boldsymbol{\phi}$.

The first principle component direction of the data is that along which the observations vary the most.

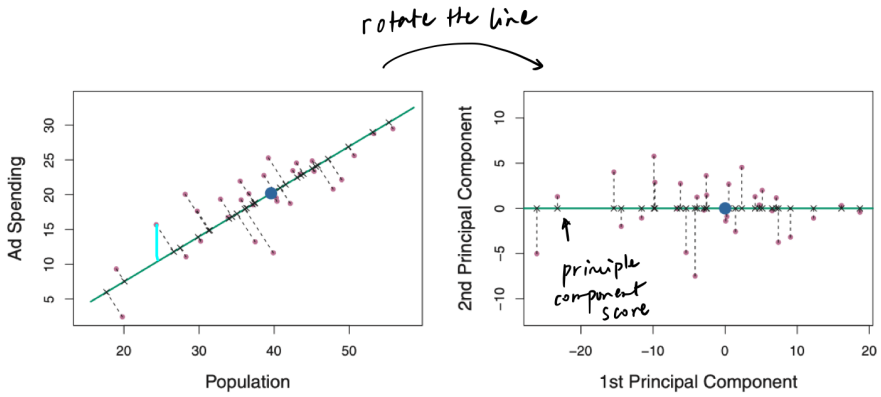
$$Z_1 = X_1\phi_{11} + X_2\phi_{21} + \cdots + X_p\phi_{p1}$$

A constraint is put on the magnitude of the ϕ values: $\phi_{11}^2 + \phi_{21}^2 + \cdots + \phi_{p1}^2 = 1$, because PCA is looking for Z_1 with the largest variance and large values of ϕ would artificially inflate the

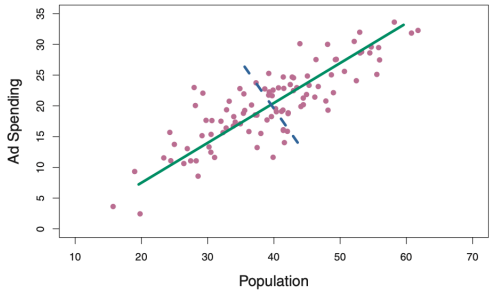
variability. For the same reason, it is generally recommended to make sure that the predictors are on a similar scale by for example standardizing the predictors.

For an individual case $\boldsymbol{x_i}$, plugging in the $\boldsymbol{x}$ values will results in a z_i score, which is case i's first principle component score.

Another interpretation of PCA is that the first principle component vector defines the line that is as close as possible to the data. As shown in the figure below, the line minimizes the sum of the squared perpendicular distances between each point and the line.
Notice how this line is minimizing a different distance in comparison to OLS. In OLS, only Y is the random variable and X is deterministic. OLS is minimizing the vertical distance from the point to the line (see the bright blue line I drawn in the figure). But in the case of PCA, the two axis consists of two Xs, both of which are of interest. So the line is minimizing the perpendicular distance.



The second principle component vector must be perpendicular to the first principle component. After removing the variance along the direction of the first principle component, PCA continues to find the next direction ($\boldsymbol{\phi_2}$) along which the remaining variance is maximized. For example, as shown in the figure below. The line defined by the second principle component is orthogonal to the line defined by the first principle component.



In the case that there are only two X variables, we can defined at most two principle components. Therefore, given the first principle component, the second component is fixed. If we were working with p X variables, the last component is fixed after finding the first $p - 1$ principle components.

The principle component regression uses the first M principle components as predictors in regression. These M principle components summarizes the variability (information) in the

original p predictors. The assumption is that the directions in which $X_1 \dots X_p$ vary the most are the directions that are associated with Y. There is no guarantee this assumption holds true, but PC regression generally works well in practice.

- The reason why PC regression improves over OLS goes back to the bias-variance trade-off. By reducing the dimensions, principle components extract information from the predictors and eliminate noise. Using M principle components (where $M < p$) would introduce bias but reduce variance. This might permit the model to generalize better to the test data.
  As the number of principle component increases (M increases), principle component model increasingly approximate the original model. When $M = p$, PC regression is the same as the OLS regression based on the p X predictors.

- PC regression tends to do well in settings when the first few PC are sufficient to capture most of the variation in the predictors as well as the relationship with the response.

- Note that PC regression is *not* a feature selection method. Each of the principle components is a linear combination of all p of the original features.

Partial least squares
Recall that one drawback of principle component regression is that there is no guarantee that the first M principle components identified as the directions along which the original p features vary the most are good predictors of Y. Partial least squares address this problem by using Y to supervise the construction of the $\boldsymbol{\phi}$ matrix (recall $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{\phi}$). The goal of $\boldsymbol{Z}$ is to extract information from $\boldsymbol{X}$ that is most relevant to Y.

PLS computes the Z_1 by setting each ϕ_{j1} equal to the coefficient from the simple linear regression of Y onto X_j. That is,

$$Z_1 = X_1\phi_{j1} + X_2\phi_{j2} + \cdots + X_p\phi_{jp}$$

where ϕ_{j1} is the coefficient of the simple linear regression $Y = X_j\phi_{j1} + \varepsilon$.
The logic is that PLS places the highest weight on the variables that are most strongly related to the response.

To identify the second component, each variable $X_1 \dots X_p$ is regressed on Z_1. The *residuals* of the p regressions are the variances of $X_1 \dots X_p$ that are not captured by Z_1. These *residuals* can be used in the exact same way to construct the second component.

Repeating this process M times will result in $Z_1 \dots Z_M$. These M components can then be used as the input of regression.

Nonlinearity – polynomial regression

For a predictor x, include its higher order terms:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 {x_i}^2 + \beta_3 {x_i}^3 + \ldots + \beta_d {x_i}^d + \varepsilon_i$$

The coefficients are still linear, but the relationship between x and y is non-linear. Generally speaking, it is unusual to use d larger than 3 or 4 because for large value of d, the polynomial curve can become overly flexible and take on very strange shapes.

Nonlinearity – Step function

For a continuous predictor x, break it into categorical dummy variables. For example, rather than including age (e.g., range= [0, 100]) as a continuous variable and has one single coefficient for it, age can be broken down to five dummy variables with indicator functions:

$$C_0(X) = I(X < c_1)$$
$$C_1(X) = I(c_1 \leq X < c_2)$$
$$C_2(X) = I(c_2 \leq X < c_3)$$
$$C_3(X) = I(c_3 \leq X < c_4)$$
$$C_4(X) = I(c_4 \leq X < c_5)$$

where, for example, c_1= 20-year old, c_2= 40-year old, c_3= 60-year old, c_4= 80-year old, and c_5= 100-year old.

Taking $X < c_1$ as the reference group, the four dummy variables $C_1(X)$, $C_2(X)$, $C_3(X)$, $C_4(X)$ can be included in the model. Each age group would have a separate estimated value on the response variable y, this gives age a non-linear effect on y.

Basic functions

Polynomial and step function are special cases of a basis function approach. The idea is to have a family of function or transformations that can be applied to a variable X, $b_1(X)$, $b_2(X)$ ... $b_k(X)$, and include all transformation to the linear regression as predictors:

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_k b_k(x_i) + \varepsilon_i$$

For polynomial function, the transformation $b_j(x_i) = {x_i}^j$.
For step function, the transformation $b_j(x_i) = I(c_j \leq x_i < c_{j+1})$.

$b_j(x_i)$ is a known function and the transformation is fixed. Thus, the above model is basically a standard linear with predictor $b_1(x_i)$ ... $b_k(x_i)$. Hence, ass inference tools and diagnostics are applicable.

Nonlinearity – regression splines

Regression splines combines and extends the polynomial and step functions introduced above. Instead of fitting a high-degree polynomial over the entire range of X, piece-wise polynomial regression fits separate low-degree polynomials over different region of X. The full range of X is separated by *knots*. A, for example, piece cubic polynomial fits a cubic regression of the form

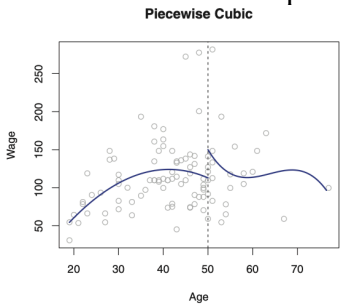
$$y_i = \beta_0 + \beta_1 x_i + \beta_2 {x_i}^2 + \beta_3 {x_i}^3 + \varepsilon_i$$

to different region of X, separated by knots. The coefficients $\beta_0,\ \beta_1,\ \beta_2,\ \beta_3$ differ in different part of the range of X. Consider an example with a single knot

$$y_i = \begin{cases} \beta_{01} + \beta_{11} x_i + \beta_{21} {x_i}^2 + \beta_{31} {x_i}^3 + \varepsilon_i & if\ x_i < c_1 \\ \beta_{02} + \beta_{12} x_i + \beta_{22} {x_i}^2 + \beta_{32} {x_i}^3 + \varepsilon_i & if\ x_i \geq c_1 \end{cases}$$

Each polynomial has four parameters, there are a total of eight degree of freedom in fitting this piecewise polynomial model.

Freely estimate these two pieces would result in two pieces that are discontinuously.



To join the two pieces appropriately, constraints need to be imposed. We need to force the two pieces to have the same values (be continuous and no jump in value):

$$\beta_{01} + \beta_{11} c_1 + \beta_{21} {c_1}^2 + \beta_{31} {c_1}^3 = \beta_{02} + \beta_{12} c_1 + \beta_{22} {c_1}^2 + \beta_{32} {c_1}^3,$$

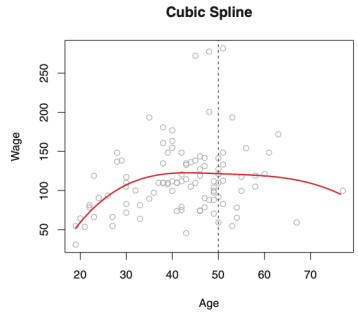
the same first derivative (same slope and no abrupt change):

$$\beta_{11} + 2\beta_{21} c_1 + 3\beta_{31} {c_1}^2 = \beta_{12} + 2\beta_{22} c_1 + 3\beta_{32} {c_1}^2,$$

or even the same second derivative (same tendency of changes in slope)

$$2\beta_{21} + 6\beta_{31} c_1 = 2\beta_{22} + 6\beta_{32} c_1,$$

at the point c_1. Each of these constraints would take away one degree of freedom, df= 8-3= 5. The resulting pieces pitch together like this:



- For cubic spline with K knots, K+1 cubic polynomials will be fitted. Degree of freedom equal to $4(K+1) - K$ constraints on the knot values $- K$ constraint on the first derivative $- K$ constraint on the second derivative = $4K + 4 - 3K = K + 4$

To fit a spline, we need to choose the number and location of the knots. The regression spline is more flexible in regions that contains a lot of knots because it allows the parameters to change rapidly to fit to the data. In practice, it is common to place knots in a uniform fashion. One way is to specify the desired degree of freedom and let the software automatically pace the corresponding number of nots at uniform quantiles of the data. The optimal degree of freedom can in turn be determined through cross validation.

Spline regression vs. polynomial regression

Spline regression often gives superior results to polynomial regression. Unlike polynomials, which must use a high degree to produce flexible fits, spline introduce flexibility by increasing the number of knots but keeping the degree fixed. Generally, the spline approach produces more stable results (i.e., less variance).

Nonlinearity – smoothing splines

Smoothing spline is an alternative approach to also produce a spline.
The logic of smoothing spline refers back to the fundamental goal of fitting a model to the data – the goal is to find a model $g(x)$ that can minimize test RSS. This goal can be translated to finding a model $g(x)$ that can minimize training RSS and has the property that permits generalizability (i.e., *smooth*). To obtain such a g function, a natural approach is to try to minimize

$$\sum_{i=1}^{n}(y_i - g(x_i))^2 + \lambda \int [g''(x)]^2 \, dx$$

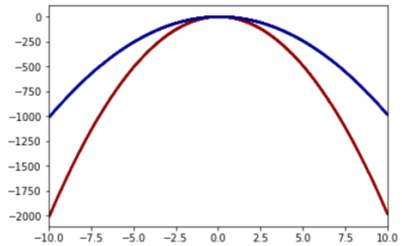
where λ is a turning parameter. The function $g(x)$ that can minimize the above function is the *smoothing spline*.

- The logic of "loss + penalty" is the same as ridge regression and lasso.
- The intuition behind $\int [g''(x)]^2 \, dx$:
  - The first derivate $g'(x)$ is the slope of $g(x)$ → how fast $g(x)$ changes.
  - The second derivative $g''(x)$ is the slop of $g'(x)$ → how fast *the slope of* $g(x)$ changes. Second derivative is a concept that measures the roughness of a function. A large absolute value of $\int g''(x_0)$ means that $g(x)$ is wiggly around the point x_0. Therefore, $\lambda \int [g''(x)]^2 \, dx$ is a term that encourages g to be smooth. The large the λ, the smoother g will be.

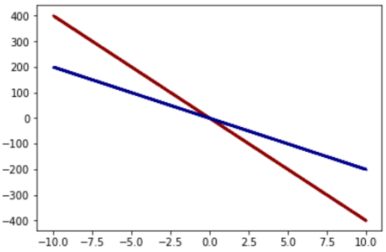
    For an example about the meaning of the second derivative, consider two functions:

    $$f_1(x) = 5 + x - 20x^2 \;\; A\ steeper\ function\ (red)$$
    $$f_2(x) = 5 + x - 10x^2 \;\; A\ smoother\ function\ (blue)$$

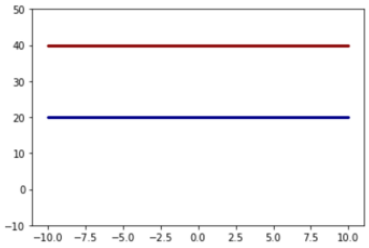
    The shapes of them



    The first derivative of them



The second derivative of them



The steeper-red function's $\int second\ derivative^2$ would be larger than the smoother-blue function.

The function $g(x)$ that minimizes the "loss+ penalty" function above is shown to have special properties:

> *it is a piece wise cubic polynomial with notes at the unique values of* $x_1, x_2 \ldots x_n$ *and continuous first and second derivatives at each knot. Furthermore, it is linear in the region outside of the extreme knots. In other words,* $g(x)$ *is a natural cubic spline with knots at* $x_1, x_2 \ldots x_n$.

However, it is not the same if we were to fit a cubic regression spline with knots at $x_1, x_2 \ldots x_n$. It is a shrunken version of such a cubic spline with the parameter λ controlling the level of shrinkage.

Recall that in the case of regression splines, degree of freedom equal to the number of free polynomial parameters minus the number of constraint. In the case of smoothing spline, the calculation of degree of freedom is no longer this straightforward. The parameter λ controls the shrinkage of the parameters and hence the *effective degree of freedom* df_λ. As λ increases from 0 to ∞, the roughness of the model decreases and the effective degree of freedom decreases from n to 2.

- The definition of effective degree of freedom is technical. To give an idea, let $\hat{g}_\lambda$ be the smoothing spline for a particular choice of λ and $\hat{g}_\lambda(x)$ be the predicted value on the response variable. We can write

  $$\hat{\boldsymbol{g}}_{\boldsymbol{\lambda}}(\boldsymbol{x}) = S_\lambda \boldsymbol{y}$$

  analogous to $\hat{Y} = HY$, where H is the hat matrix. S_λ is a $n * n$ project matrix that transform the observed outcome value to the predicted values. Effective degree of freedom is the trace of the S_λ matrix.

In smoothing splines, we do not need to set the number and location of knots because there is a knot at every unique value of x. But we need to choose the value of $\boldsymbol{\lambda}$**.** This can be done through cross-validation. Specifically, leave-one-out-cross-validation is efficient in this context (for every $\boldsymbol{\lambda}$ value, perform LOOCV to gauge its test error). Refer to p.279 of the book for details.

Nonlinearity – local regression

Local regression is a different approach for fitting flexible non-linear functions, which involves computing the fit at a target point x_0 using only the nearby training observations. The steps of local regression are summarized below.

Algorithm 7.1 *Local Regression At $X = x_0$*

1. Gather the fraction $s = k/n$ of training points whose x_i are closest to x_0.
2. Assign a weight $K_{i0} = K(x_i, x_0)$ to each point in this neighborhood, so that the point furthest from x_0 has weight zero, and the closest has the highest weight. All but these k nearest neighbors get weight zero.
3. Fit a *weighted least squares regression* of the y_i on the x_i using the aforementioned weights, by finding $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize

$$\sum_{i=1}^{n} K_{i0}(y_i - \beta_0 - \beta_1 x_i)^2. \qquad (7.14)$$

4. The fitted value at x_0 is given by $\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0$.

- Local regression is a memory-based procedure. Like nearest-neighbors, we need all the training data each time we wish to compute a prediction.
- The most important choice is the span s, defined in the Step 1 above. It controls the flexibility of the non-linear fit. The smaller the value of s, the more local and wiggly will the fit be.
- Local regression suffers from high dimensionality just like nearest neighbor. When the number of predictors is larger than 3 or 4, there will generally be very few observations close to $\boldsymbol{x_0}$ to support the local regression.

Generalized additive models

All the techniques above discuss how to fit a non-linear relationship between one predictor x and the response variable Y. These approaches can be applied to settings of multiple regression $X_1, X_2 \ldots X_p$. Generalized additive model is a general framework for extending a standard linear model by allowing non-linear functions of each of the variables, while maintaining additivity:

$$y_i = \beta_0 + f_1(x_{1i}) + f_2(x_{2i}) + \cdots + f_p(x_{pi}) + \varepsilon_i$$

f_j is specified separately for different predictor X_j. All the techniques above can be used to specify f_j.

For example,

- $f_1(x_{1i}) = \beta_{11}x_{1i} + \beta_{12}{x_{1i}}^2 + \beta_{13}{x_{1i}}^3$ make X_1 being included as polynomial terms.

- $f_2(x_{2i}) = \begin{cases} \beta_{21a}x_{2i} + \beta_{22a}{x_{2i}}^2 + \beta_{23a}{x_{2i}}^3 + \varepsilon_i & if\ x_i < c_1 \\ \beta_{21b}x_{2i} + \beta_{22b}{x_{2i}}^2 + \beta_{23b}{x_{2i}}^3 + \varepsilon_i & if\ x_i \geq c_1 \end{cases}$ allow X_2 to be included as natural splines.
- Specifying a smoothing spline for X_j is not as simple.

The advantages of GAM are:

- By fitting a non-linear f_j to each x_j we can automatically model non-linear relationships that standard linear or logistic regression will miss.
- Hence we can potentially make more accurate predictions.
- An additive model still allows us to examine the effect of each x_j on y individually while holding all the other x's fixed. Thus we still have some interpretation and inference.

The disadvantages of GAM are:

- The model is restricted to be additive.
- Therefore cannot model interactions; for example, the simple model $y = x_1x_2 + \varepsilon$ cannot be fitted well by $y = f_1(x_1) + f_2(x_2) + \varepsilon$ (can still manually add an interaction term)
- Sometimes interpretation is far from clear
- Tuning each function separately is computationally prohibitive (e.g. λ in a smoothing spline); choosing the same tuning parameter for all the functions does not always work well.

Tree

Decision tress can be applied to both regression and classification problems.

Regression Trees

Roughly speaking, there are two steps:

a. The predictor space—the space constructed by $X_1, X_2 \ldots X_p$—is divided into J distinct and non-overlapping regions $R_1, R_2 \ldots R_J$.
b. 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.

To make the division on the predictor space (step a), *recursive binary splitting* is made. This so-called recursive binary splitting is a top-down greedy approach that begins at the top of the tree and successively make the best split at every step.

At every step, the algorithm considers all possible splits: Let (j, s) denotes one split at value s on predictor j. The algorithm considers all possible (j, s) and divide the observation cases into the ones whose $X_j < s$, noted as $R_1(j, s) = \{X | X_j < s\}$, and whose $X_j \geq s$, noted as $R_2(j, s) = \{X_j \geq s\}$. The mean responses of the training observations in $R_1(j, s)$— $\hat{y}_1$—is the estimated/prediction value for test cases that end up in the R_1 region; the mean responses of the training observation in $R_2(j, s)$— $\hat{y}_2$—is the estimated/prediction for test cases that end up in the R_2 region.

The algorithm chooses the (j, s) can result in the tree having the lowest RSS. For every potential split, the tree develops two children nodes from the mother node. The change in RSS is:

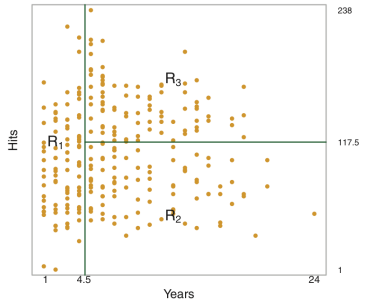
$$RSS_{mother} = \sum_{i \in R_{mother}} (y_i - \bar{y})^2$$

$$RSS_{children} = \sum_{i \in R_1(j,s)} (y_i - \hat{y}_1)^2 + \sum_{i \in R_2(j,s)} (y_i - \hat{y}_2)^2$$

$$identify\ the\ (j, s)\ that\ maximizes\ RSS_{mother} - RSS_{children}$$

Because every split is a yes-no split on a single variable (s at the predictor X_j), every step of the division is orthogonal and the resulting region R_j is a hypercube.

For example, as shown in the graph below, the second split is orthogonal to the first split. This generalizes to higher dimensions.



However, The tree building does not look ahead, so there is no guarantee that these local best decisions can accumulate to a global best tree.

For decision tree, overfitting means building a very large tree and the end nodes include only a few cases. To regulate the complex of decision, the ***tree pruning*** technique is used: A very large tree T_0 is first built, and then it is pruned back to obtain a subtree. Again, recall that the goal is to obtain a tree that can generalize well to test data.

Since there are an extremely large number of possible subtrees, it is not realistic to use cross validation method to examine the performance of every subtree. A better method is *cost complexity pruning* (weakest link pruning) which examine the generalizability of the subtrees upon turning a turning parameter α. For each value of α, there corresponds a subtree T such that

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

is as small as possible. Here $|T|$ indicates the number of terminal nodes of tree T; R_m is hypercube of the predictor space corresponds to the terminal node m. The logic here is similar to the "Loss + Penalty" minimization problem of ridge regression, lasso, and smoothing splines. The $\alpha|T|$ is a penalty term that punishes the complex tree that has many terminal nodes. When $\alpha = 0$, the tree only cares about loss and will just be T_0. The larger the α, the simpler the tree will be. For every given α, there is a way to identify the subtree that minimize the function above. I do not understand the technicality behind tree pruning; I should probably come back to this if I have the time. Cross-validation is used to select the appropriate value of α.

Classification Trees

Classification trees are similar to regression trees except that it is used to predict nominal response variables rather than continuous ones. In the case that the response variables are not continuous, we can no longer use RSS as the evaluation criterion.

> A natural alternative would be classification error rate—the fraction of the training observations that do not belong to the most commonly class$1 - \hat{p}_{max,k}$; but it turns out that this classification error is not sensitive for tree growing. Recall that percentage cannot be interpret by itself. An increase from 0.5 to 0.6 is not the same as an increase from 0.8 to 0.9.

Alternative evaluation criteria are Gini index and entropy.

Gini index is defined as

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

where K denotes classes and m denotes the mth node. Gini index is calculated at each node.

> For example, if a node contains 10 A, 20 B and 15 C, then the Gini index of this node $= \frac{10}{45}\left(1 - \frac{10}{45}\right) + \frac{20}{45}\left(1 - \frac{20}{45}\right) + \frac{15}{45}\left(1 - \frac{15}{45}\right)$.

Gini index would be small if $\hat{p}_{mk}$ is close to 1 or 0. Thus, Gini index is a measure of *purity*—a node that contains predominantly observations from a single class would have a small Gini index.

Entropy is an alternative measure, given by

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

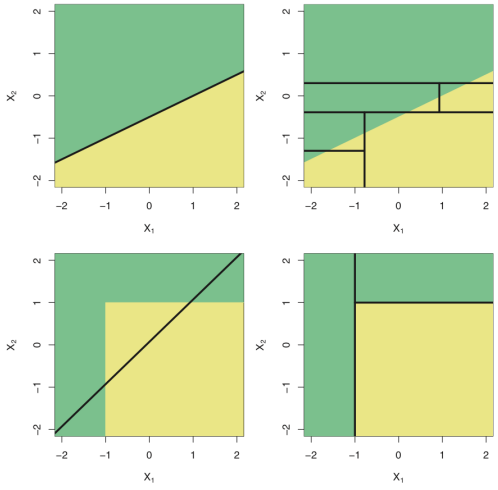
The entropy is a log-likelihood function.

> At each node, the distribution of the K classes is assumed to follow a multinomial distribution and the probability of each category is noted as p_{mk}. The likelihood function of the node is then $\prod_{k=1}^{K}(p_{mk})^{n_{mk}}$, and the loglikelihood function is $\sum_{k=1}^{K} n_{mk} log(p_{mk})$. The log-likelihood defined in this way is influenced by the size of the node; with the same distribution, a larger node would have a smaller loglikelihood. Thus, we normalize the loglikelihood by dividing it by n_m: $\frac{\sum_{k=1}^{K} n_{mk} log(p_{mk})}{n_m} = \sum_{k=1}^{K} p_{mk} log(p_{mk})$.
>
> Furthermore, log-likelihood is negative, add a negative sign in front of it (actually adding -2 in front of it would make D follow a Chi-squared distribution, but this is not how D is defined here). The smaller the likelihood, the smaller the log-likelihood (negative), the larger the – loglikelihood (positive). Thus, D is a measure of deviance, which we wish to minimize.

The Gini index and the entropy are used the same way. For each potential split, a mother node v_{mother} is split to two children node v_{child1} and v_{child2}. The Gini index and entropy are calculated for the v_{mother}, v_{child1}, and v_{child2} node respectively. At every step, the split that results in the largest $G_{mother} - (G_{child1} + G_{child2})$ or $D_{mother} - (D_{child1} + D_{child2})$ is selected.

Comparison between decision tree and linear model

Whether decision tree of linear model is better depends on the problem at hand. If the decision boundary is highly non-linear, then decision tree will outperform linear regression, as demonstrated in the graphs below.



Decision tree can also automatically interactions between predictors. A small decision tree also has visualization and interpretability advantages.

Ensemble

A disadvantage of trees is that they are typically non-robust. A small change in the data can cause a large change in the final estimated tree.

Bagging, random forests, and boosting uses tree as building blocks to construct more powerful and less variable prediction models. Bagging and boosting are general-purpose procedures for reducing variance in statistical learning method. They are particularly useful and frequently used in the context of decision tree.

Bagging

The theory behind bagging is that averaging a set of observations reduce variance. A natural way to reduce the variance and increase prediction accuracy is to take many training sets from the population, build a separate prediction model using each training set and average the results of the predictions. (This is the "simple replicated sampling" in the applied sampling theories). While this is a useful conceptual idea, it is impossible to implement in practice because we do not have multiple independent training sets from the population.

Instead, the full training dataset is treated as the population and subsamples are repeated drawn from the full training dataset with the technique of bootstrapping—sampling with replacement. With this approach, we generate B different bootstrapped training dataset; we then train the model on the bth sample to get $\hat{f}^b(x)$; and we finally average all predictions from B models to obtain

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

This is bagging.

In the context of regression tree, each of the $\hat{f}^b(x)$ model is a regression tree. These trees are grown deep and not pruned. Hence, each individual trees is highly variable but lowly biased.

The extension to classification tree is straightforward. Rather than averaging across the predicted values made by B regression trees, the bagged classification model takes the majority vote from classification decisions of the B classification trees.

$$\hat{C}^{bag}(x) = majority\ vote\{\hat{C}^b(x)\}_{b=1}^{B}$$

where $\hat{C}^b(x)$ denotes a classification tree.

When bootstrapping a sample of size n from the original sample of size n, on average, the bootstrapped sample consists of around $\frac{2}{3}$ of the elements in the original sample.

This is easily provable:

Given the technique is sampling with replacement, for an element i, its chance of not being included in the bootstrapped sample is $(\frac{n-1}{n})^n$ and its chance of being included in the bootstrapped sample is $1-(\frac{n-1}{n})^n$. As n gets larger, the $1-(\frac{n-1}{n})^n$ approaches to around 0.63. Each element has around 0.63 chance of being included and thus the bootstrapped sample consists of around 0.63 of the elements in the original sample.

The setting of bootstrapping results in a natural split of the original dataset that can be used for training set and validation set, respectively: The two-third of the original training observations will be sampled to build a decision tree $\hat{f}^b(x)$, and the remaining one-third of the observations are referred to as the *out-of-bag* observations and serve as the validation data. That is, each tree makes prediction for around $\frac{1}{3}$ of the observations. If in total, B trees are built, then each observation will get around $\frac{B}{3}$ predictions. These predictions are averaged or taken majority vote to yield a single out-of-bag prediction.

Variable importance measures. While bagging improves prediction accuracy in comparison to single decision trees, the resulted models of bagging are hard to interpret.

- For regression trees, one way to measure the importance of each predictor variable is to sum up the total amount of decreases in RSS due to splitting a predictor and average this value over all B trees.
- For classification trees, similarly, we can sum up the total amount of decrease in Gini index due to splitting a predictor and average this value over all B trees.

Random Forest

As in bagging, random forests builds decision tress on bootstrapped training samples. The difference between random forests and bagging is that random forest subsets predictors at each split.

That is, each time a split in a tree is considered, a random subset of m predictors is chosen as split candidates from the full set of p predictors. For the next split, a fresh subset of predictors of size m is redrawn. Typically, $m=\sqrt{p}$ (e.g., $\sqrt{13}\approx 4$). Therefore, in the process of tree building, at each split in the tree, the algorithm is not even allowed to consider a majority of the available predictors.

The rationale behind this technique is that the trees are forced to consider different predictors rather than holding on to several strong predictors, which results in the trees being very similar to each other. As random forests de-correlate the trees, averaging across B uncorrelated trees tends to give better results than averaging across B highly correlated trees.

Boosting

Boosting is different from bagging and random forests. In bagging and random forests, each of the decision trees is built independently. But in boosting, trees are grown sequentially: each tree is grown using information from previously grown trees. The new tree is built to the residuals of the previous tree and thus the new tree specially tackle the mistakes; and then the new decision tree is added into the function to update the residuals. By fitting small trees to the residuals, the $\hat{f}$

is improved slowly on where it did not perform well. In general, statistical learning approaches that learn slowly tend to perform well.

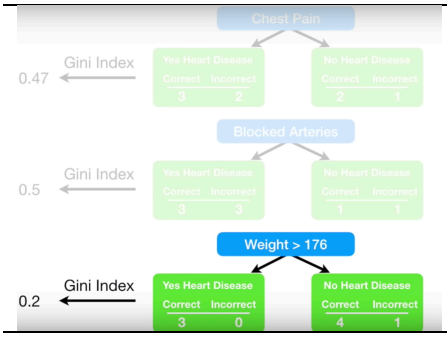
Let's consider ***AdaBoost***[1] to demonstrate the idea of boosting. There are ***three main concepts*** behind AdaBoost (in contrast to random forest):

1) In a forest of trees made with AdaBoost, the trees are usually typically a node and two leaves (consist of a single split). A tree that has only one node and two leaves is called a "*stump*". That is, AdaBoost virtually builds a forest of stumps. One stump is obviously not good at making predictions, so stumps are weak learners.
2) In random forest, each tree has an equal vote on the final classification. In a forest of stumps made with AdaBoost, some stumps get more say in the final classification than others.
3) In random forest, each decision tree is made independently from each other, so the order doesn't matter. In contrast, in a forest of stumps, order is important. The error that the first stump makes influences how the second stump is made, etc. etc..

How to create a forest of stumps:

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

Consider a sample consisting of 8 individuals. In the beginning, each individual in the sample receive the same weight $\frac{1}{8}$.

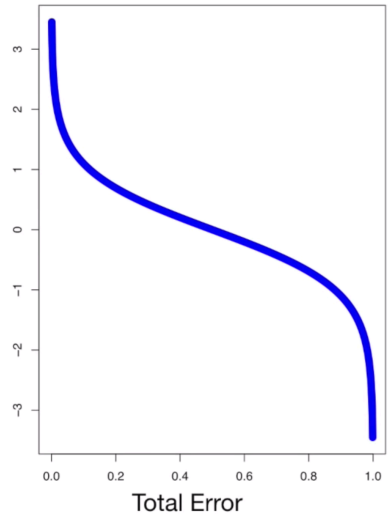


We build the first stump by looking for the variable that best at classifying heart disease.

In this case, patient weight is the best predictor because it gives the smallest Gini index (how the cutoff point = 176 is not discussed here.)

Now that we have decided the first stump, we determine how much say this stump will have in the final classification. This depends on the quality of the classification.
The *total error* for a stump *is the sum of the weights associated with the incorrectly classified sample cases*. Because sample weights sum up to 1, the total error is between the range of [0, 1].

[1] https://www.youtube.com/watch?v=LsK-xG1cLYA



Amount of say is a function of total error:

$$amount\ of\ say = L \log\left(\frac{1 - Total\ Error}{Total\ Error}\right)$$

where $L \leq 1$ is the learning rate. Let $L = \frac{1}{2}$ for this example.

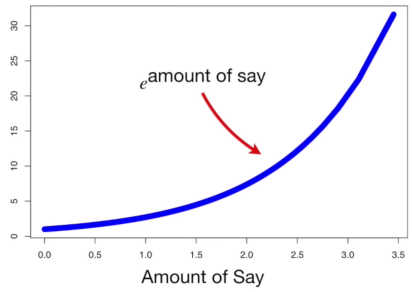
If we plot the `amount of say` as a function of `total error`, the plot is shown in the left cell.

For the first stump, it makes one error and the weight of the error is $\frac{1}{8}$, and thus, the total error of this stump is $\frac{1}{8}$.

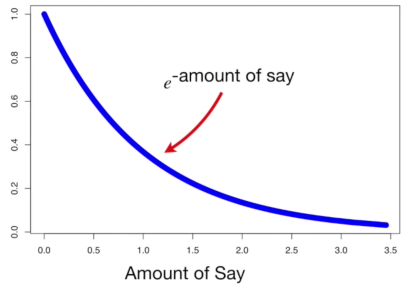
Plugging in the value, this stump's amount of say is $\frac{1}{2}\ln\left(\frac{1-\frac{1}{8}}{\frac{1}{8}}\right) = 0.97$.

---

a.



b.



Now we modify the weights, so that when building the next stumps, it takes the error of the current stump into account.

a. Increase the sample weights for the cases that were incorrectly classified.

$New\ weight = sample\ weight * e^{amount\ of\ say}$

To understand the rationale of this re-weighting, consider the graph a. in the left cell:

If the `amount of say` is large, meaning that the current stump does a good job in classification, then the original sample weight is inflated by a large extent. This makes intuitive sense. If the current stump is a good classifier, then we treat its mistakes more seriously.

b. Decrease the sample weights for the cases that were correctly classified.

$New\ weight = sample\ weight * e^{-amount\ of\ say}$

To understand the rationale of this reweighting, consider the graph b. in the left cell:

If the `amount of say` is large, meaning that the current stump does a good job in classification, then the original sample weight is shrunken to a large extent.

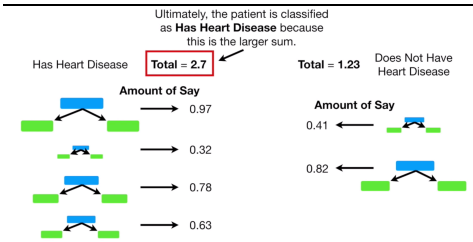
Combining a. and b., a good stump will results in wide divisions between the incorrectly classified cases (a.) and the correctly classified cases (b.)

The new weights are re-normalized to sum up to 1.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	Sample Weight
Yes	Yes	205	Yes	1/8	0.07
No	Yes	180	Yes	1/8	0.07
Yes	No	210	Yes	1/8	0.07
Yes	Yes	167	Yes	1/8	0.49
No	Yes	156	No	1/8	0.07
No	Yes	125	No	1/8	0.07
Yes	No	168	No	1/8	0.07
Yes	Yes	172	No	1/8	0.07

Here is a demonstration of the reweighting.
The highlighted case is incorrectly classified. Its new weight is $\frac{1}{8}e^{0.97} = 0.33$
The rest cases are correctly classified. Their new weight is $\frac{1}{8}e^{-0.97} = 0.05$
Normalizing them to sum to one, 0.33 is scaled to 0.49, and 0.05 is scaled to 0.07.
Now we have a new column of sample weights to repeat the procedure to make the second stump by, for example, using weighted Gini index.

This is how error that the first tree makes influence the error that the second tree makes, etc.



Ultimately, we have a forest of stumps that have different amount of say in the final classification.
We sum up the amount of say of the stumps that vote "yes", and sum up the amount of say of the stumps that vote "no".
The larger number gives the final result.

The idea of boosting regression tress is similar.

Algorithm 8.2 *Boosting for Regression Trees*

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).
   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \quad (8.10)$$
   (c) Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \quad (8.11)$$
3. Output the boosted model,
   $$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \quad (8.12)$$

There are three turning parameters:

1) The number of tree B. Unlike bagging and random forest, boosting can overfit if B is too large, although this overfitting tend to occur slowly.
2) The shrinkage parameter λ, a small positive value. This controls the rate at which boosting learns. Typically values are 0.01 or 0.001.

   Linking to the AdaBoost example above, in the re-weighting step, the new weight is obtained by $new\ weight = sample\ weight * e^{amount\ of\ say}$ and $amount\ of\ say = L \log(\frac{1-Total\ Error}{Total\ Error})$. The L is the λ that can be tuned. The smaller the L, the less the weights are updated, the less a stump's mistake influences the next stump.
3) The number d of splits in each tree, which controls the complexity of the boosted ensemble.

   In the AdaBoost example above, $d = 1$ as each tree is a stump. $d = 1$ typically works well.

   Note that when $d = 1$, the ensemble is fitting an additive model, since each split involves only a single variable and each split is performed on all cases. (Interactions is modeled in trees as hierarchically splitting on different variables, so not all splits are performed on all cases.)

   Thus, more generally, d *is the interaction depth* and control the interaction order of the boosted model.

Support Vector Machines

Distinguish between three notions:

1. Maximal marginal Classifiers—making perfect classification between the two classes for the training observations
2. support vector classifiers—extend maximal marginal classifiers by allowing incorrect classification
3. support vector machines—generalize the support vector classifier to non-linear settings

1. Maximal marginal classifiers

First, let's discussion the concept of hyperplane and how it is used for classification.
In a p-dimensional space, a hyperplane is a flat subspace of dimension $p-1$. The mathematical definition of a hyperplane is

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0$$

In other words, a hyperplane of dimension $p-1$ separates the p dimensional space in to two halves.

> Rewriting the mathematical definition of the hyperplane $X_p = \beta_0' + \beta_1' X_1 + \cdots + \beta_{p-1}' X_{p-1}$. We can see that the last dimension is defined by the other dimensions. This shrinks the last dimension. (somehow reminds me of 降维打击…). For example:
>
> - In a two-dimensional case, $\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$ can be rewritten as one dimension being defined by the other one. This constrains the hyperplane as a line.
> - In a three-dimensional case, $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 = 0$ can be rewritten as one dimension being defined by the other two. This constrains the hyperplane as a plane.

Since a hyperplane divides the space into two halves, it can be used determine which side of the hyperplane a point lies. If a case has $\boldsymbol{X}$ that

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p > 0$$

then it lines to one side of the hyperplane; if its $\boldsymbol{X}$ satisfies

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p < 0$$

then it lines to the other side of the hyperplane.

Now suppose that we have a $n \times p$ data matrix $\boldsymbol{X}$ that consists of n training observations in a p-dimensional space, and these cases fall into two classes $\{-1,1\}$. Suppose there is a separating hyperplane ($\boldsymbol{X}$) that can perfectly separate the responses of the observation (Y) so that

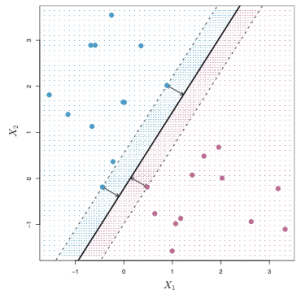
$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} > 0$ if $y_i = 1$
$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} < 0$ if $y_i = -1$

Then the hyperplane would have the property that

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) > 0$$

We can then use the hyperplane to make prediction for test observations. Note that the magnitude of the $f(x^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + \cdots + \beta_p x_p^*$ matters. If $f(x^*)$ is very far from zero, then it's far away from the hyperplane and we are more confident in the prediction.

If there is one hyperplane that can perfectly separate the observations, then there are effectively infinite number of hyperplanes that can make the separation because we can move or rotate the hyperplane a little bit without making an difference.
Maximal marginal classifiers is the method that found the optimal separating hyperplanes among all workable hyperplanes—"optimal" defined as the hyperplane that has the furthest minimum distance to the training observations and "minimum distance" defined as the perpendicular distance from each training observation to the hyperplane. For an example of an optimal hyperplane, see the figure below:



This is a line that is furthest to all observations in the two-dimensional space. Three observations lie along the dashed lines indicating the width of the margin. These three observations are the ***support vectors*** of this hyperplane.

- They are vectors because they are in the p dimensional space; they "support" the hyperplane in the sense that if they move, then the hyperplane would move as well.

Interestingly, the maximal margin hyperplane depends directly on the support vectors, but not on the other observations: A movement to any of the other observations would not affect the separating hyperplane, provided that the observation's movement does not cause it to cross the boundary set by the margin.

To construct the maximal margin classifier, the following maximization problem is to be solved:

Subject to the constraint that $\sum_{j=1}^{p} \beta_j^2 = 1$, the perpendicular distance from an observation $\boldsymbol{x_i}$ to a hyperplane is given by $y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip})$.
For each of the observation $\boldsymbol{x_1}$ ... $\boldsymbol{x_n}$ in the training set, it has a perpendicular distance to the hyperplane.
Let M denotes the smallest distance out of the n distances. The maximization problem tries to find $\boldsymbol{\beta}$ that can maximize the smallest distance M.
This idea is formally expressed as:

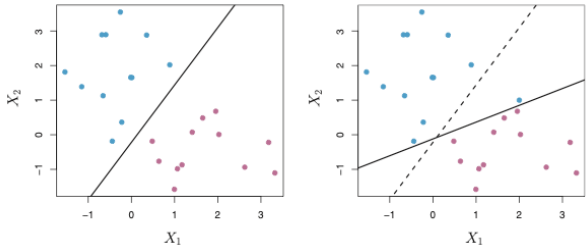
maximize M
subject to $\sum_{j=1}^{p} \beta_j^2 = 1$,
$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M \ \ \forall i = 1, 2, \ldots n.$

2. Support vector classifiers

The maximal marginal classifier is a very natural way to perform classification, if a separating hyperplane exist. But realistically, it is more common the case that the two classes cannot be exactly separated. In this situation, the goal is refined to developing a hyperplane that almost

separates the classes, using a so-called soft margin. The generalization of the maximal margin classifier to the non-separable case is known as the support vector classifier.

Support vector classifier is not only suitable to situations in which the two classes are not separable. Sometimes, there exist an exact separating hyperplane, but if we can tolerate a few incorrect classification, we can make better predictions for the rest of the cases. Also, because maximal marginal classifier looks for exact separation, a few changes in the data can result in huge change in the hyperplane. See the graph below for an example showing how one case can make a dramatic difference in maximal marginal classifier's hyperplane.

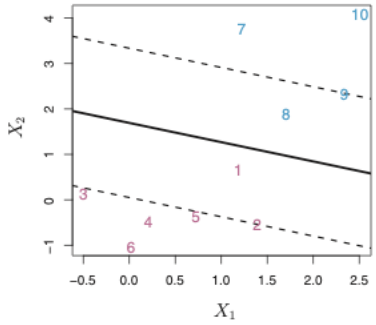


Recall that the distance from the observation to the hyperplane is of interest. The further away the case it is, the more confident we are in its prediction. Shifting the hyperplane results in shorter distances for the rest of the observations, and this means lower quality prediction for the rest of the cases.

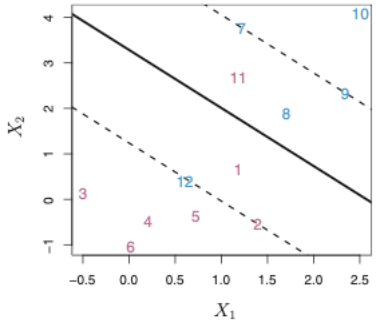
To summarize, we might be willing to consider imperfect separation in the interest of

- Getting a more robust classifier
- Getting better classification of most of the training observation.

For such imperfect classifications



This is called on the incorrect side of the margin



This is called on the incorrect side of the hyperplane

Support vector classifier revises the maximization problem of the maximal marginal classifier:

maximize M
subject to $\sum_{j=1}^{p} \beta_j^2 = 1$,
$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M(1 - \varepsilon_i) \ \forall i = 1, 2, \ldots n.$
$\varepsilon_i > 0, \sum_{i=1}^{n} \varepsilon_i < C.$

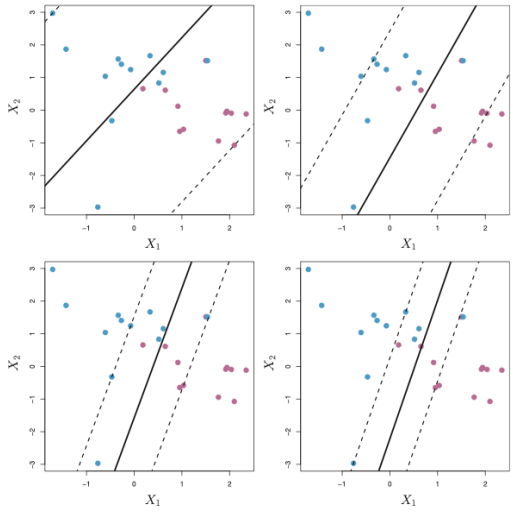
The insights behind the maximization problem is actually intuitive.
Recall that $y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip})$ should be larger than 0 if the classification prediction is correct (i.e., $\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} > 0$ and y_i is indeed 1; or $\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} < 0$ and y_i is indeed -1).

a. If we let a case ith's $\varepsilon_i > 1$, it means that we can accept its $y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip})$ to be negative. In this case, the prediction for the case i would be wrong since $(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip})$ does not have the same sign as y_i. However, the error that we can tolerate is constrained by $M(1 - \varepsilon_i)$, meaning that the error cannot result in an overly off negative value and the observation does not lie too far away in the wrong side of the hyperplane.
b. If we let a case ith's $0 < \varepsilon_i < 1$, it means that we can accept its $y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip})$ to be smaller than M. Since $y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip})$ is still positive, the prediction is at least correct about the side of the case, but the case lie within the margin.

$\varepsilon_1 \ldots \varepsilon_n$ are the slack variables that allow the individual observation to on the wrong side of the hyperplane (a.) or the wrong side of the margin (b.) if needed.

C is a nonnegative turning parameter. C bounds the sum of the ε_i so it determines the number and severity of the violation that we tolerate. It is like a budget for the amount of violation by the n observations. When C is 0, we do not tolerate any violation, the classifier is effectively maximal marginal classifier; when $C > 0$, no more than C observations can be on the wrong side of the hyperplane because each violation of hyperplane results in $\varepsilon_i > 1$ and costs at least 1 from the budget of C. In practice, C is a generally chosen via cross-validation. C controls the bias-variance trade-off. A small C results in a low-bias-but-high-variance classifier and a large C results in a high-bias-but-low-variance classifier.

Recall that in the case of maximal marginal classifier, only observations lie on the margins are support vectors that can influence the hyperplane.
Here in the case of support vector classifier, *only observations that lie directly on the margin or on the wrong side of the margin for their class are known as* ***support vectors***. The other observations that lie strictly on the correct side of the margin do not affect the support vector classifier.
This is linked to the turning parameter C. When C is large and the margin is wide, many observations violate the margin and so there are many support vectors. On the other hand, if C is small and the margin is narrow, fewer observations violate the margin and there are fewer support vectors. See the graph below for how changes in the width of the margin associate with different number of support vectors:



3. Support Vector Machine

Support vector machine generalizes support vector classifier to non-linear situations.

Turns out that the solution to the maximization problem of support vector classifier

maximize M
subject to $\sum_{j=1}^{p} \beta_j^2 = 1$,
$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M(1 - \varepsilon_i) \ \forall i = 1, 2, \ldots n.$
$\varepsilon_i > 0, \sum_{i=1}^{n} \varepsilon_i < C.$

involves only the inner product of the test case with the support vectors—feeding a new observation to the solution can be conceptualized as a function that involves the inner product of this new observation and support vectors. The solution to classifying a new test case x_0 can be expressed as

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i \langle x_0, x_i \rangle$$

where S is the collection of support vectors;
$\langle x_0, x_i \rangle$ is the inner product of x and x_i: $\langle x_0, x_i \rangle = \sum_{j=1}^{p} x_{0j} x_{ij}$
$\alpha_1 \ldots \alpha_S$ can be estimated through the inner products $\langle x_i, x_{i'} \rangle$ of training observations.

To summarize, in representing the linear classifier and in computing its coefficients, all we need are inner products. The inner products between training observations are used to estimate the coefficients; the inner products between the test cases and the support vectors of the classifier are used to classify the test case.

A kernel is a function that generalizes the inner product

$$K\langle x_i, x_{i'} \rangle$$

For example, rather than computing $\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j}$, $K\langle x_i, x_{i'} \rangle$ computes $(1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d$, where d is a positive integer that can turn the linear relationship to a non-linear relationship. If $d = 1$, then it reduces to support vector classifier.
When the support vector classifier is combined with a non-linear kernel like this, the result is called a support vector machine.

In this case, the non-linear function has the form

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i \, K\langle x_0, x_i \rangle$$

Unsupervised Learning

In supervised learning, there is a distinction between predictors $\boldsymbol{X}$ and the outcome variable Y, and the goal is to make predictions on the Y using the $\boldsymbol{X}$. In unsupervised learning, there are only $\boldsymbol{X}$ variables and the goal is to discover patterns and information from them. The two particular types of unsupervised learning are principle components analysis and clustering.

Principle component analysis

This topic has been covered a little bit above under the principle component egression. Here, this concept is further discussed. Recall that the goal of principle component analysis is basically to rotate the data space and find the directions along which the data are the most, the second most, the third most etc. variable. For a dataset of p dimensions, there are at most p principle components. Ideally, the first few (i.e., k, $k < p$) principle components capture the most variability of the original data, then we can use these k principle components to represent the original data and achieve dimension reduction.

The first principle component of a set of feature $X_1, X_2, \ldots X_p$ is the normalized linear combination of the features:

$$Z_1 = X_1\phi_{11} + X_2\phi_{12} + \ldots + X_p\phi_{1p}$$

that has the largest variance. The elements $\phi_{11} \ldots \phi_{1p}$ are refer to as loadings of the first principle component. Their squared sum ${\phi_{11}}^2 + {\phi_{12}}^2 + \cdots + {\phi_{1p}}^2 = 1$. Together, they make up the principle component loading vector for the first principle component: $\phi_1 = \begin{bmatrix} \phi_{11} \\ \vdots \\ \phi_{1p} \end{bmatrix}$. An individual i in the dataset has a principle component score by plugging in his or her values on the $\boldsymbol{X}$ variables:

$$z_{1i} = x_{1i}\phi_{11} + x_{2i}\phi_{12} + \ldots + x_{pi}\phi_{1p}$$

How to find the first principle component?
For a simple explanation, assuming that the p predictors $X_1, X_2, \ldots X_p$ are all centered around their mean. Then Z_1 as a linear function of $\boldsymbol{X}$ would have its mean being zero as well. The goal is therefore, to

$$maximize\left\{\frac{1}{n}\sum_{i=1}^{n}(X_{1i}\phi_{11} + \ldots + X_{pi}\phi_{1p} - 0)^2\right\}, subject\ to\ \sum_{j=1}^{p}{\phi_{1j}}^2 = 1$$

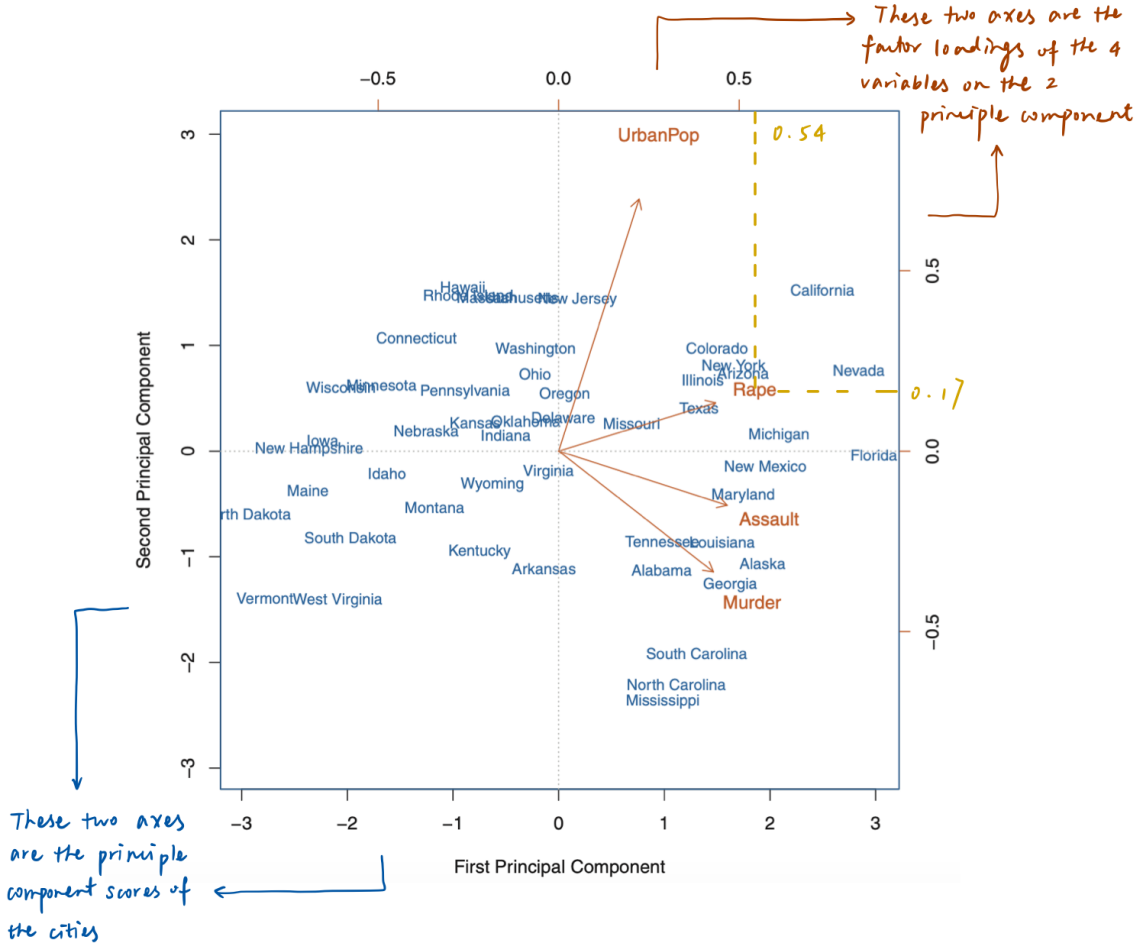
This problem can be solved by eigen decomposition of linear algebra.

The second principle component is the linear combination of $X_1 \ldots X_p$ that has maximal variance and is perpendicular to the first principle component. Given a two-dimensional data, the second principle component is exactly defined by the first principle. But for p-dimensional data, there are $p - 1$ distinct principle components.

Once we have computed the principle component, we can plot them against each other to produce low-dimensional view of the data. Consider this example, the original data are 4-dimensional, recording the characteristics of 50 American cities. These four variables' loadings on the first two principle components are summarized in the table below:

	PC1	PC2
Murder	0.5358995	−0.4181809
Assault	0.5831836	−0.1879856
UrbanPop	0.2781909	0.8728062
Rape	0.5434321	0.1673186

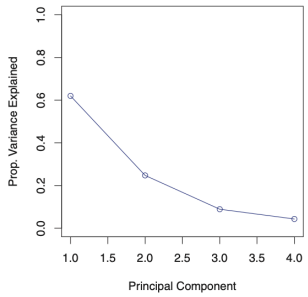
Correspondingly, given each city's scoring on the four crime variables, it has a principle component score on PC1 and a principle component score on PC2.
The two pieces of information 1) the two principle component scores of each of the cities and 2) the loadings of the four variables on two principle components can be plotted in the same graph. Such a display that graph both the principle component scores and the loading vectors is referred to as a *single biplot*.



From this plot, we can see that the first principle component is mainly characterized by rape, assault, and murder, whereas the second principle component is mainly characterized by UrbanPop.

A few extra notes on PCA:

- PCA is scale dependent. Recall that it's goal is to look for the direction along which the data vary the most. A large scale of one variable can dwarf the effects of all other variables. Therefore, it is typically recommended to standardize the predictors before implementing PCA.
- To evaluate how good the first few principle components summarize the original data, we can calculate the *proportion of variance* explained by each principle component.
    - The total variance presented in a data set (assuming that all variables are centered around the mean) is defined as
    $$\sum_{j=1}^{p} Var(X_j) = \sum_{j=1}^{p} \left[\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - 0)^2\right]$$
    - The variance captured by the mth principal component is
    $$var(Z_m) = \frac{1}{n}\sum_{i=1}^{n}(z_{mi} - 0)^2$$
    where z_{mi} is the linear combination of the variable $\boldsymbol{X}$: $z_{mi} = x_{1i}\phi_{m1} + x_{2i}\phi_{m2} + \ldots + x_{pi}\phi_{mp}$
    - The *proportion of variance* is the ratio of these two:
    $$\frac{var(Z_m)}{\sum_{j=1}^{p} Var(X_j)}$$
- The proportions of variance of the principle components can be used to graph a *scree plot* and determine the number of principle components we wish to obtain.



We find the "elbow" of the plot where the curve starts to lever off. Additional principle components after the elbow no longer carry much variance.

Clustering

While PCA looks to find a low-dimensional representation of the observations that explain a good fraction of the variance, clustering looks to find homogeneous subgroups among the observations. Two best-known clustering techniques are K-means clustering and hierarchical clustering.

K-means clustering. To perform K-means lustering, we first specify the desired number of clusters K; then the algorithm assign each observation to exactly one of the K clusters.
The K-means clustering is based on a simple and intuitive mathematical problem. The good clustering is one for which the within-cluster variation is as small as possible. This can be formally expressed as

$$minimize\left\{\sum_{k=1}^{K} W(C_k)\right\}$$

where $W(C_k)$ is the within-cluster variance, defined, for example, as the sum of all pairwise squared Euclidean distances between the observation in the kth cluster, divided by the total number of observations in the kth cluster:

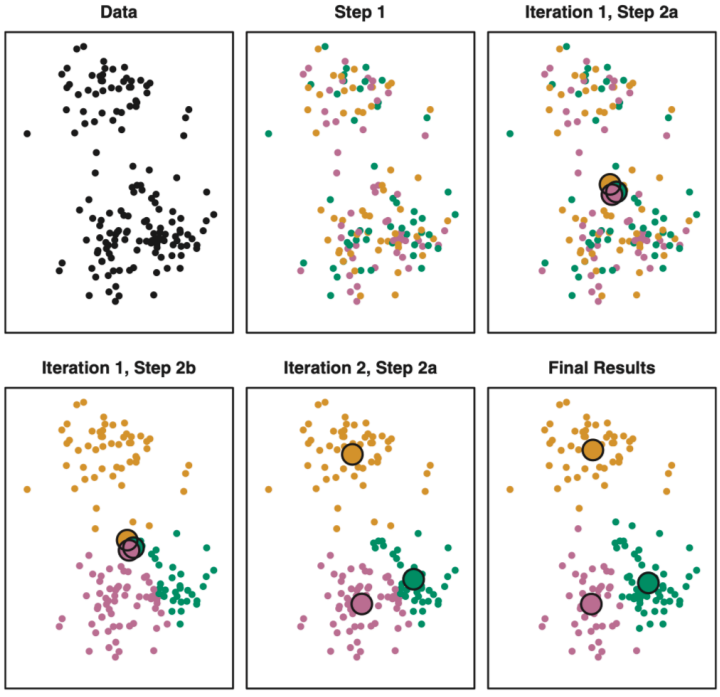
$$W(C_k) = \frac{1}{|C_k|}\sum_{i,i\prime\in C_k}\sum_{j=1}^{p}(x_{ij} - x_{i\prime j})^2$$

This minimization is a problem that is hard to solve precisely because there are too many ways to partition the n observations into K categories in order to find the best partition.

There is a simple to solve this problem that will generate reasonably good solution:

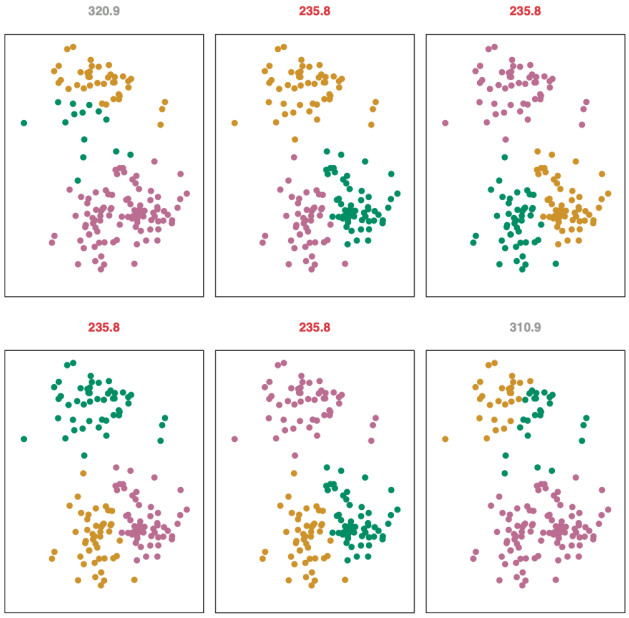
1. Randomly assign a number, from 1 to K, to each of the observations (i.e., randomly assign each observation to one cluster).
2. For each of the K clusters, compute the centroid → This gives K centroids.
3. Assign each observation to the cluster whose centroid is closest.
4. Repeat step 2 and 3 until the cluster assignment stop changing.

Here is an example demonstrating how the algorithm works:



However, this procedure only gives a local optimum. It does not guarantee that the result is the global optimum. Depending on the random starts, the results might not be the same.

Hence, it is important to run the algorithm multiple times with different random starts then select the best solution that is the most commonly appear. Here is an example showing how different random starts result in different clustering solution:



But one solution is clearly the best as it occurs multiple times with different random starts.

Hierarchical clustering. Comparing to K-means clustering, one advantage of hierarchical clustering is that it does not require us to prespecify the number of clusters K. It results in a tree-based representation of the observations, called *dendrogram*.

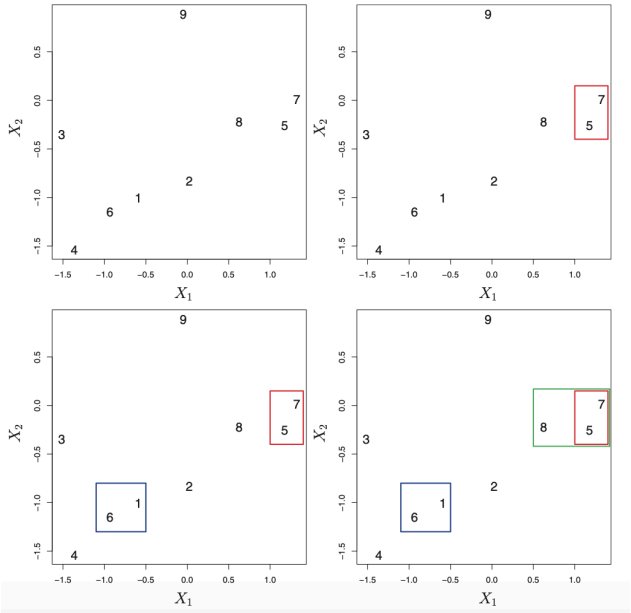
Hierarchical clustering is a bottom-up or agglomerative process:

1. Beginning with n observations and a measure of pairwise dissimilarity (e.g., Euclidean distance), treat each observation ass its own cluster.
2. Examine all pairwise inter-cluster dissimilarities and identify the pair that are the least dissimilar, fuse these two clusters.
   [In the first round, reduce from n to $n-1$ clusters].
3. Compute the new pairwise inter-cluster dissimilarities the remaining clusters, repeat.

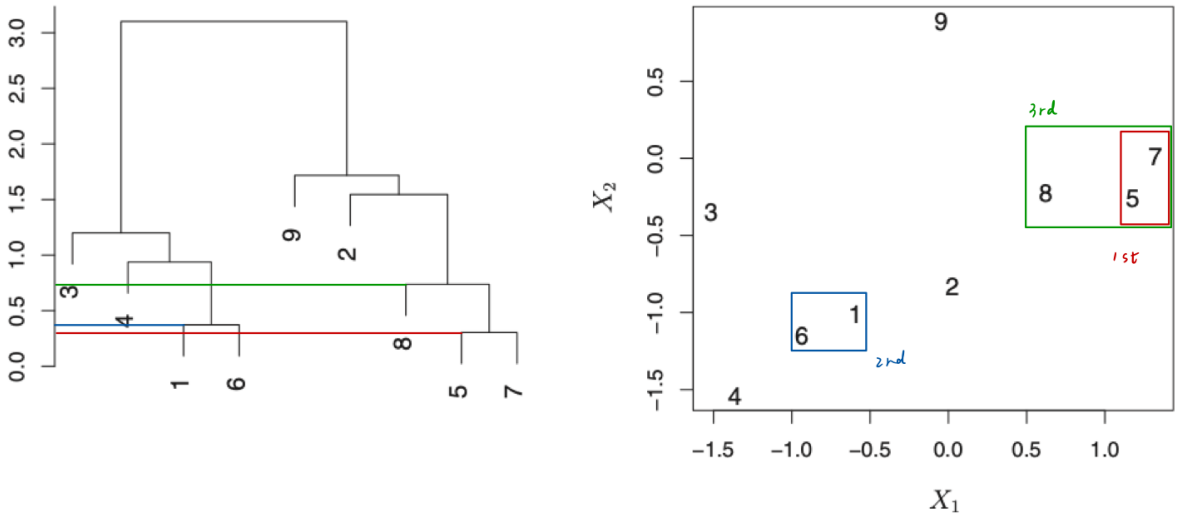
The inter-cluster dissimilarity can be defined in many different ways:

- Centroid: dissimilarity between the centroids of the two clusters.
- Average: compute all pairwise dissimilarities between observations in the two clusters, then take the average.
- Single: compute all pairwise dissimilarities between observations in the two clusters, then take the minimum.
- Complete: compute all pairwise dissimilarities between observations in the two clusters, then take the maximum.

Here is one example showing how clusters are fused in steps:



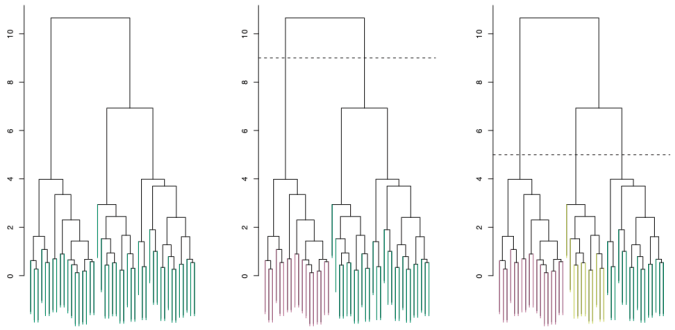
The fusion process can be expressed in a tree-form. The dissimilarity between the two clusters when they are fused is recorded in the height of the tree. In the example, 5 and 7 are the first two clusters that are fused, then 1 and 6, and then 8 and 5&7.



For each two observations, their dissimilarity is defined as the height where they are first fused. Note that once clusters are fused, they are regarded as a collective. Hence, the dissimilarity between 9 and 2 is the same as the dissimilarity between 9 and 7. This is because 2 and 7 are fused first, before them as a collective meet 9. The height at which 2 fuses with 9 is the same as the height at which 7 fuses with 9.

Linking back to our purpose, this dendrogram can be used for clustering by making horizontal cut across it. The sets of observations beneath the cut can be interpreted as clusters.

Cutting it at different heights result in different numbers of clusters. This is a very attractive feature because one single dendrogram can be used to obtain any number of clusters. Since the structure and similarity between the observations are well displayed by the tree shape, we can look at the dendrogram and select a sensible place to cut the tree:



Extra points to note about hierarchical clustering:

- The dissimilarity function used to indicate dissimilarity between clusters is very important. The discussion above relies on Euclidean distance. There are other measures such as correlation-based distance, defined as the correlation between each two clusters (e.g., A has 100 data points on 100 variables; B has 100 data points on 100 variables. A correlation can be calculated between these 100 pairs of data points. This is an unusual use of correlation.).
  An example demonstrating the difference between Euclidean and correlation-based distance:
  Consider a dataset of customer shopping records. Each row is a customer, each column is a product, each cell is 1 or 0.
  If Euclidean distance is used, customers who bought very few things would have a small distance and will be grouped together.
  If correlation distance is used, even for the customers who bought very few things, what they bought is still important. The customers who have similar preference would have a high correlation. s
- Scaling variables also have an effect on the dendrogram.