Lecture 1: An Overview of Machine Learning

Beidi Qiang

SIUE

What is Machine learning?

Machine learning (ML) is a category of an algorithm that enables computer to learn and adapt from data, draw inference and become more accurate in predicting outcomes. It refers to a vast set of tools used for understanding data.

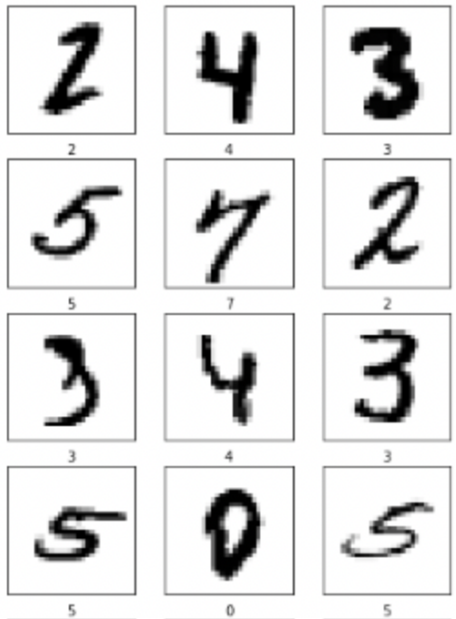
Types of Machine Learning

Machine learning can be classified into 2 types of algorithms.

- Supervised Learning: building a statistical model for predicting, or estimating, an output (label) based on one or more inputs (features). Linear regression is a simple example of supervised learning.
- Unsupervised Learning (Chapter 12 ISLR): there are inputs but no supervising output (label); nevertheless we can learn relationships and structure from such data, such as grouping similar data together (clustering).

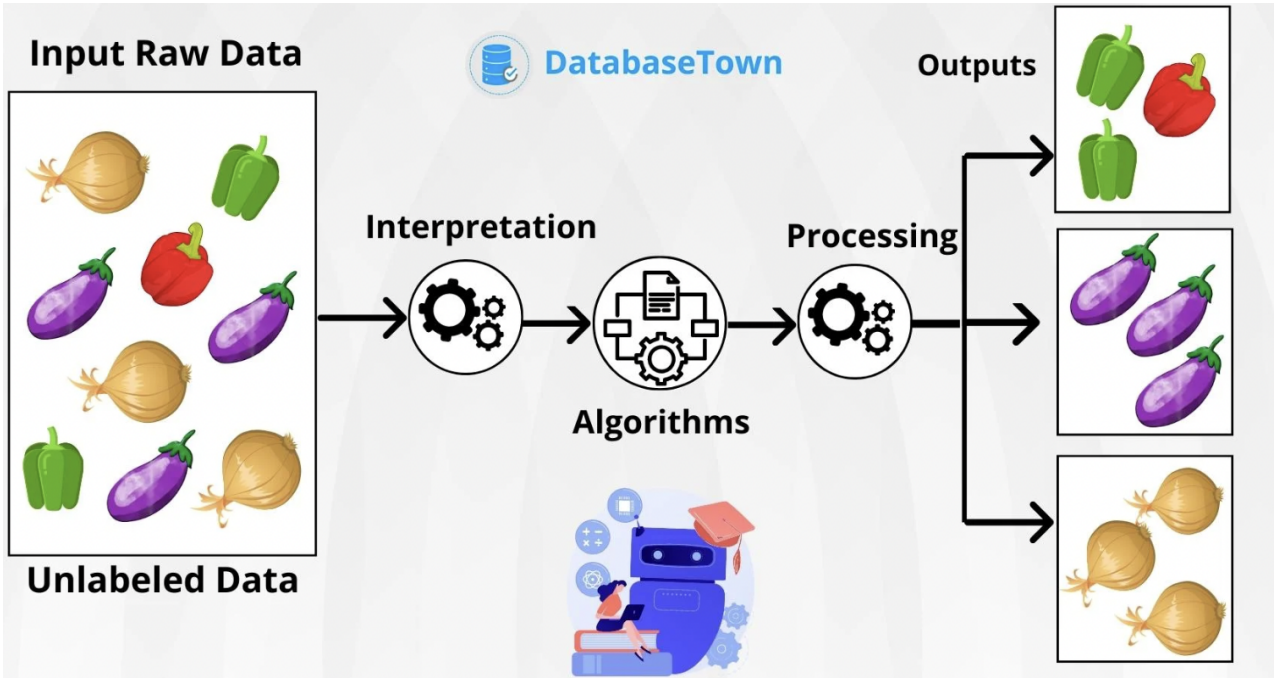
Examples of Supervised Learning

- Predict the salary by years of expert. (for example using linear regression)
- Recognize images of handwritten digits (classify an image as 0,1,2,... or 9).
- Financial industry and trading: companies use ML in fraud investigations (classify a transaction as fraud or not)



Example of Unsupervised Learning

- Clustering: you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior, or grouping images by their characters.



A Quick History of Machine Learning

- Linear discriminant analysis was proposed in 1936 and in the 1940s, logistic regression was developed.
- In 1970s, many more non-linear techniques for learning from data were available and the computation become feasible in 1980s by the improvement of computing technology. that non-linear methods were no longer computationally prohibitive
- In the mid 1980s, classification and regression trees were developed. Neural networks gained popularity in the 1980s, and support vector machines arose in the 1990s.
- Machine learning became very famous in the 1990s. The intersection of computer science and statistics gave birth to probabilistic approaches in AI. This shifted the field further toward data-driven approaches.

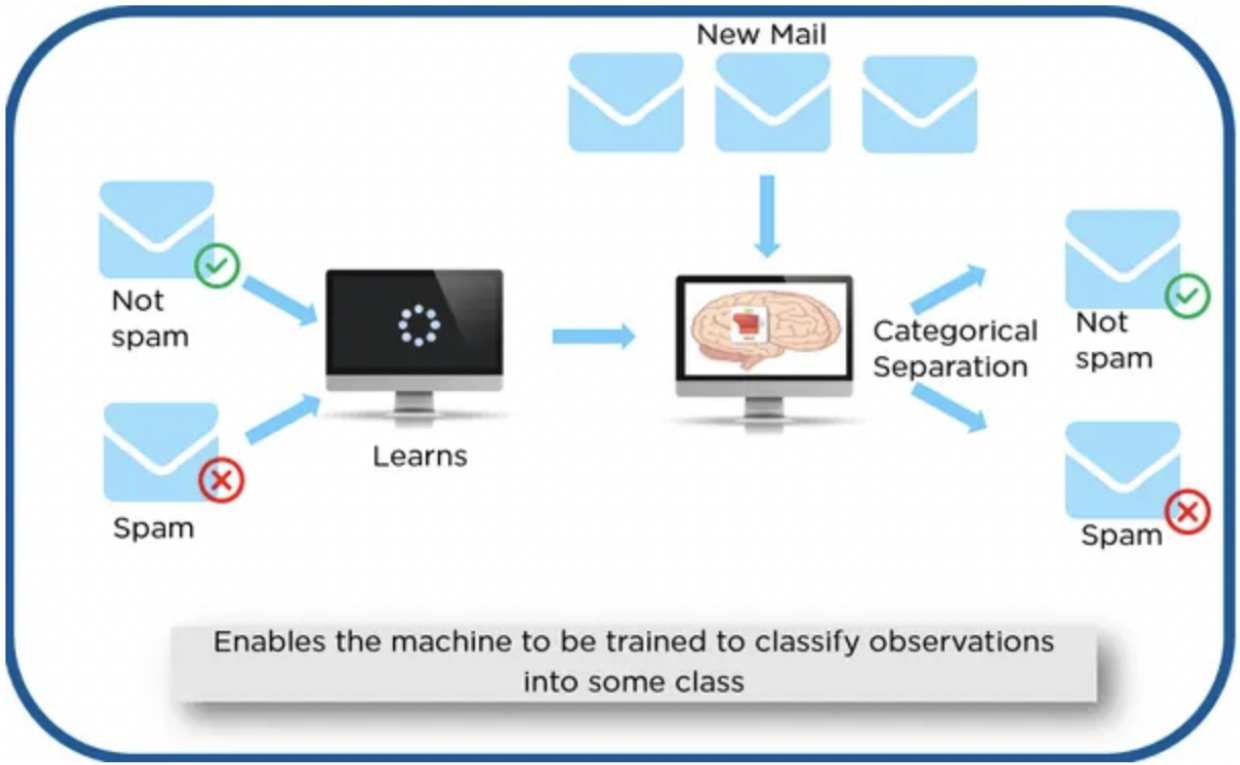
Key ML Terminology

- Labels: The thing we're predicting. Other names: output, response, target. Example: the y variable in simple linear regression.
- Features: The input variables. Other names: input, predictors, independent variables. Example: the x variable in simple linear regression.
- Examples: An example is a particular instance of data, labeled or unlabeled.

Example: Email spam filtering
Labels: spam or not spam.
Features: words in the email text, sender's address, time of day the email was sent, etc...

Key ML Terminology, cont.

- Model: A model defines the relationship between features and label.
- Model Training: Means creating or learning the model. That is, you show the model examples and enable the model to gradually learn the relationships between features and label.
- Inference means applying the trained model to unlabeled examples.



Regression vs. Classification

We tend to select statistical learning methods on the basis of whether the label is quantitative (numerical) or qualitative (categorical).

- A regression model predicts continuous values. For example, regression models make predictions that answer questions like the following:
  - What is the value of a house in Edwardsville, IL?
  - What is the probability that a user will click on this ad?
- A classification model predicts discrete values or classes. For example, classification models make predictions that answer questions like the following:
  - Is a given email message spam or not spam?
  - Is this an image of a dog, a cat, or a hamster?

Lecture 2: Assessing ML Model Accuracy

Beidi Qiang

SIUE

Training and Loss

In supervised learning, a machine learning algorithm builds a model by examining many examples and attempting to find a model that minimizes loss.

- Assume that there is some relationship between the label Y and features $\mathbf{x}$. Say
$$Y = f(\mathbf{x}) + \epsilon$$
- Training a model simply means learning (determining) good values for all model parameters in f.
- The result of training can be expressed as an estimate of the function $f(\mathbf{x})$, denoted as $\hat{f}$. which takes features $\mathbf{x}$ as input and that generates a prediction $\hat{Y}$.
- Loss is the penalty for a bad prediction. That is, loss is a number indicating how bad the model's prediction was on a single.

Reducible error and irreducible error

The accuracy of a prediction for Y depends on two quantities, which we will call the reducible error and the irreducible error.

- $\hat{f}$ will not be a perfect estimate for f, and this inaccuracy will introduce some error. This error is reducible because we can potentially improve the accuracy of $\hat{f}$
- variability associated with ϵ also affects the accuracy of our predictions. This is known as the irreducible error, because no matter how well we estimate f, we cannot reduce the error.

Split the Data

A machine learning model aims to make good predictions on new, previously unseen data. To see how model performs on unseen data, we introduced the idea of dividing your data set into training and test. Our test set serves as a proxy for new data.

- Training set—a subset to train a model.
- Test set—a subset to test the model.

Make sure that your test set meets the following conditions:

- Is large enough to yield statistically meaningful results.
- Is representative of the data set as a whole. In other words, don't pick a test set with different characteristics than the training set.

Basic assumptions of ML models

The following are basic assumptions when collecting data and train the ML model:

- We draw examples independently and identically (i.i.d) at random from the distribution. This is usually achieved by a simple random sample from the population.
- The distribution is stationary; that is the distribution doesn't change within the data set.
- We draw examples from partitions from the same distribution.

Assessing Model Accuracy: Regression

- In the regression setting, the most commonly-used measure for loss is the mean squared error:

$$MSE = \frac{1}{n}\sum(y_i - \hat{f}(x_i))^2.$$

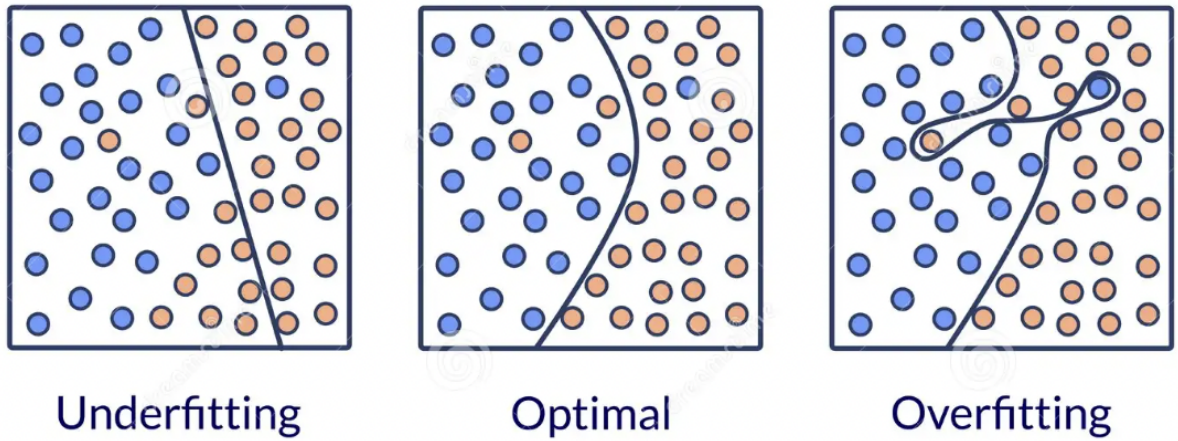
- The MSE computed using the training data is referred to as the training MSE.
- The MSE computed using the testing data (previously unseen) is referred to as the testing MSE.
- We want to choose the method that gives the test MSE as small as possible.

Assessing Model Accuracy: Classification

- In the classification setting, the most commonly-used measure is the error rate:
$$\frac{1}{n}\sum I(y_i \neq \hat{f}(x_i)).$$
- The error rate computed using the training data is referred to as the training error. The error computed using the testing data is referred to as the testing error.
- A good classifier is one for which the test error is smallest..

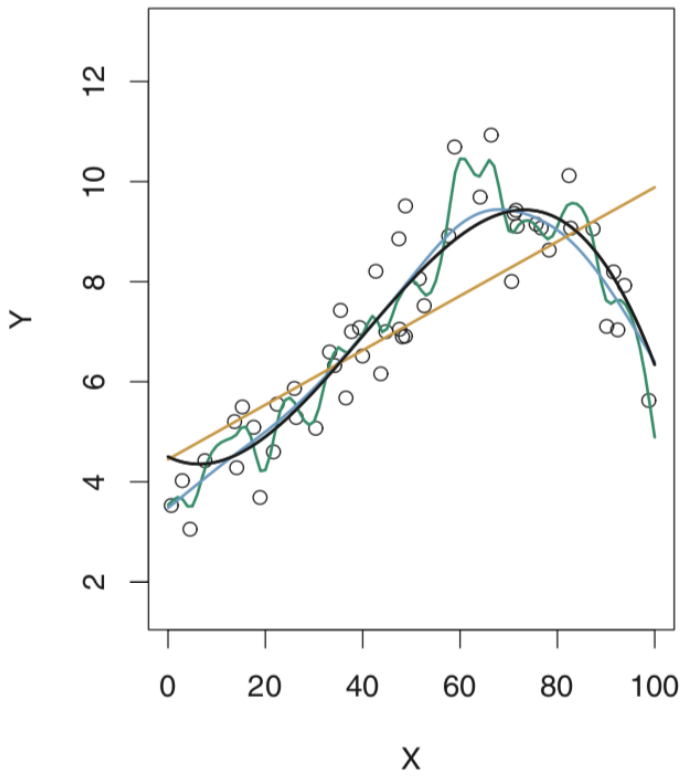
Peril of Overfitting

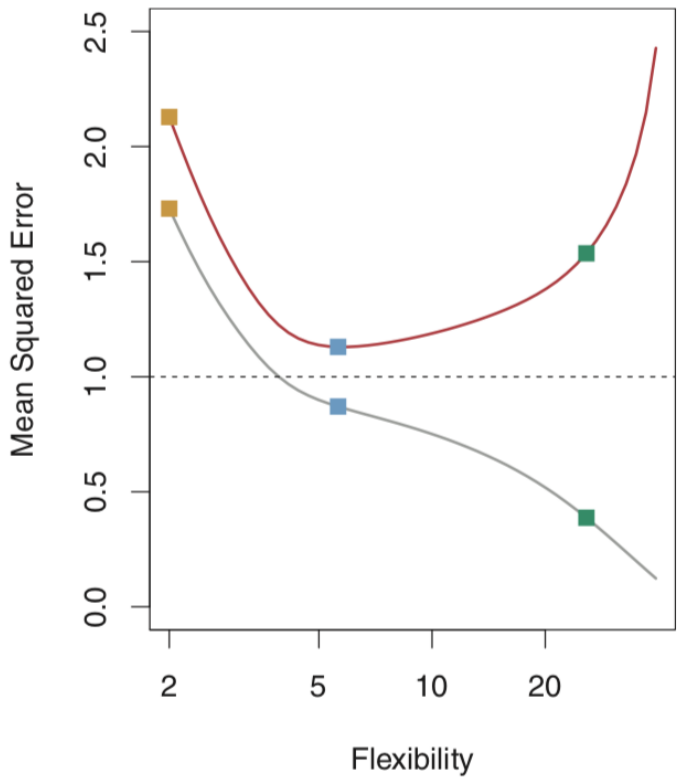
Overfitting occurs when a model tries to fit the training data so closely that it does not generalize well to new data.



Model Complexity, simulated example

In the follow plot, we have data simulated from a distribution f (black). Three estimates of f are based on the data are shown: the linear regression line (orange), and two higher order polynomial fits (blue and green curves). Training data MSE (grey), test MSE data (red). It is clear that as the level of flexibility (complexity) increases, the curves fit the observed data more closely, but not necessarily the true f.





The Bias-Variance Trade-Of

The U-shape observed in the test MSE curves turns out to be the result of two competing properties:

$$\mathrm{E}\left[y_0 - \hat{f}(x_0)\right]^2 = \mathrm{Var}\left(\hat{f}(x_0)\right) + \left[Bias(\hat{f}(x_0))\right]^2 + \mathrm{Var}\left(\epsilon\right)$$

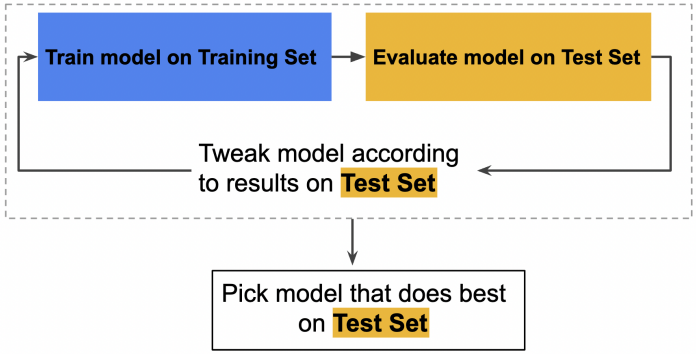
- $\mathrm{E}\left[y_0 - \hat{f}(x_0)\right]^2$ is the expected test MSE.
- $Bias(\hat{f}(x_0)) = \mathrm{E}\left[y_0 - \hat{f}(x_0)\right]$.
- To minimize the expected test error, we need to select a statistical learning method that simultaneously achieves low variance and low bias.

The Bias-Variance Trade-Of

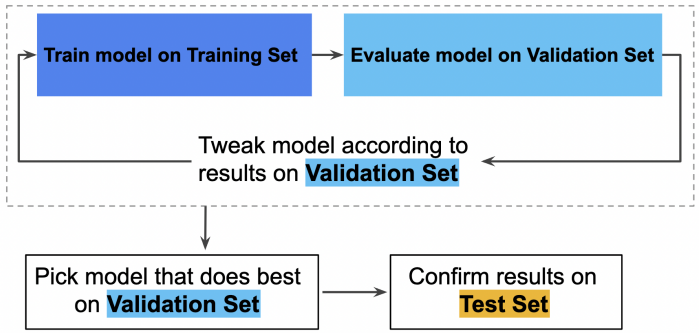
- Variance refers to the amount by which $\hat{f}$ would change if we estimated it using a different training data set. In general, more flexible statistical methods have higher variance.
- Bias refers to the error in estimating y that is introduced by using simplified model. It is unlikely that any real-life problem truly can be represented by any simple model. Generally, more flexible methods result in less bias.
- 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 increases or decreases.

Model Tweak and Another Partition

Partitioning a data set into a training set and a test set enabled you to train on one set of examples and then to test the model against a different set of examples. With two partitions, the workflow could look as follows:



You can greatly reduce your chances of overfitting by partitioning the data set into the three subsets by introducing a validation set to tweak your model. This is a better workflow because it creates fewer exposures to the test set.



Lecture 3: KNN Classifier

Beidi Qiang

SIUE

Recall: The Classification Setting

Assume that there is some relationship between the label Y and features $\mathbf{x}$. Say

$$Y = f(\mathbf{x}) + \epsilon$$

We seek to estimate f on the basis of training observations $\{(x_1, y_1), \cdots, (x_n, y_n)\}$, where $y_1, \cdots y_n$ are qualitative (categorical). Most common approach for quantifying the accuracy of our estimate $\hat{f}$ is the error rate, defined as

$$\frac{1}{n} \sum I(y_i \neq \hat{y}_i)$$

- $\hat{y}_i$ is the predicted class label for the ith example using $\hat{f}$, and y_i is the true label.
- $I(y_i \neq \hat{y}_i)$ is an indicator variable that equals 1 if $y_i \neq \hat{y}_i$ and if 0 $y_i = \hat{y}_i$.

The Bayes Classifier

A good classifier is one for which the test error rate is smallest. It is possible to show that the test error rate is minimized, on average, by a very simple classifier called Bayes Classifier. Bayes classifier simply assign a test observation with features x_0 to the class j for which $Pr(Y = j|X = x_0)$ is largest.

- $Pr(Y = j|X = x_0)$ is a conditional probability: it is the probability conditional that $Y = j$, given the feature vector x_0 .
- The Bayes classifier produces the lowest possible test error rate, called the Bayes error rate, which equals to

$$1 - E\left(max_j Pr(Y = j|X)\right).$$

- Bayes error rate is always greater than 0 in any real world problem, because the classes overlap in the population. It is analogous to the irreducible error.

K-Nearest Neighbors

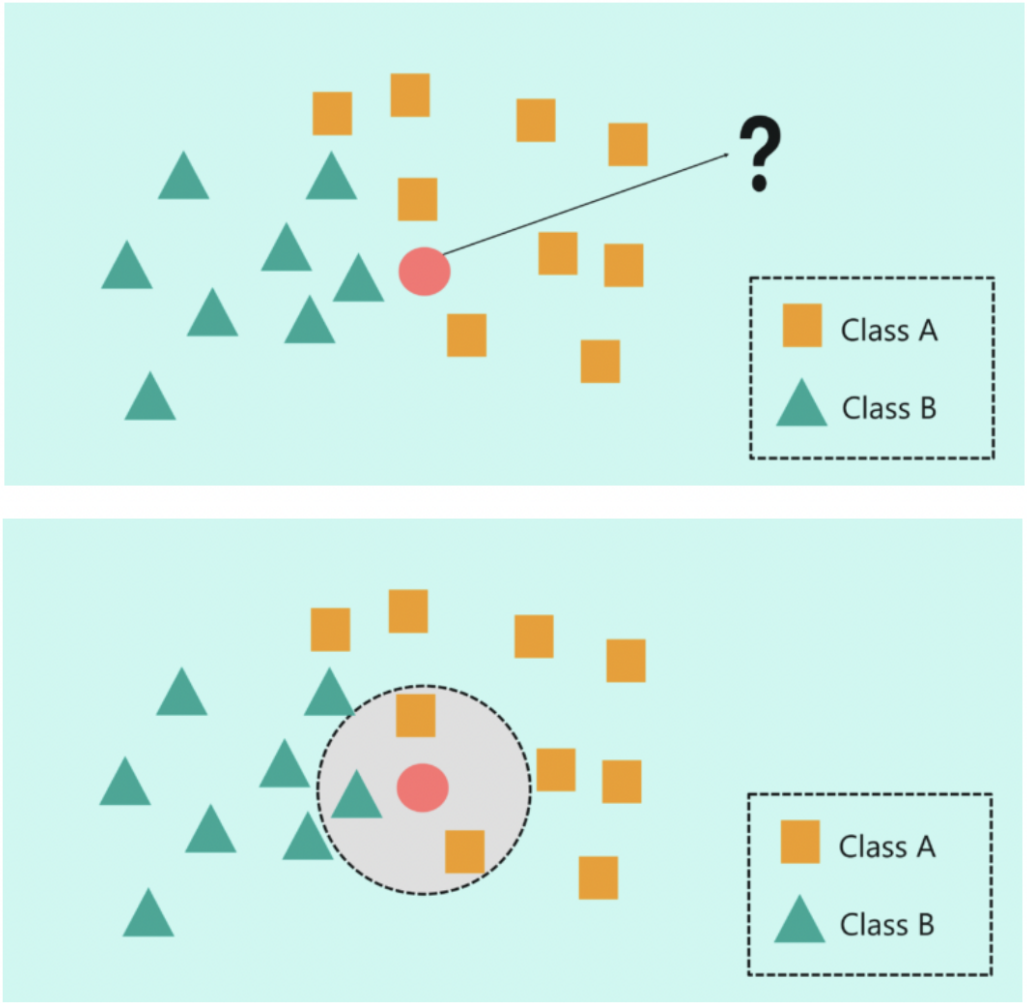
In theory we would always like to predict qualitative labels Y using the Bayes classifier. But for real data, we do not know the conditional distribution of Y given X.

Many approaches attempt to estimate the conditional distribution, and then classify a given observation to the class with highest estimated probability. One such method is the K-nearest neighbors (KNN) classifier.

- Take the K Nearest Neighbor of a new example from the training data.
- Estimates the conditional probability for class j as the fraction of points in neighbors whose label equals to j.
- Classifies the new example to the class with the largest estimated conditional probability, or equivalently the class, where you counted the most neighbors.

KNN Example

Consider the following example:



Distance measure

KNN uses distance measures to check the distance between a new example and its neighbors. Let's use $x_0 = (x_{01}, \cdots, x_{0m})$ to denote the m dimensional feature space of the new example. Commonly used distance measures are:

- Minkowski distance (Lp norm): a metric intended for real-valued feature vector.

$$D = \left(\sum_{i=1}^{m} |x_{0i} - x_{1i}|^p \right)^{1/p}$$

  - Euclidean Distance (L2 norm): $p = 2$
  - Manhattan distance (L1 norm): $p = 1$

- Hamming Distance: a metric for comparing two binary data strings. It is the number of bit positions in which the two bits are different. This is used mostly when you one-hot encode your data.
  - Suppose we have two strings "10100" and "11001". The hamming distance here will be 3.
- Gower distance: a metric that measures the dissimilarity with mixed feature vector (both quantitative and categorical variables).

Normalization/Standardization

When calculating distances, we have to take into account the units used. If the scale of features is very different, normalization or standardization is required. This is because the distance calculation done in KNN uses feature values. When the one feature values are large than other, that feature will dominate the distance hence the outcome of the KNN.

- Normalization: recalculation of feature to values between 0 (the minimum) and 1 (the maximum) with the formula:

$$x_{i,normalized} = (x_i - x_{min})/(x_{max} - x_{min})$$

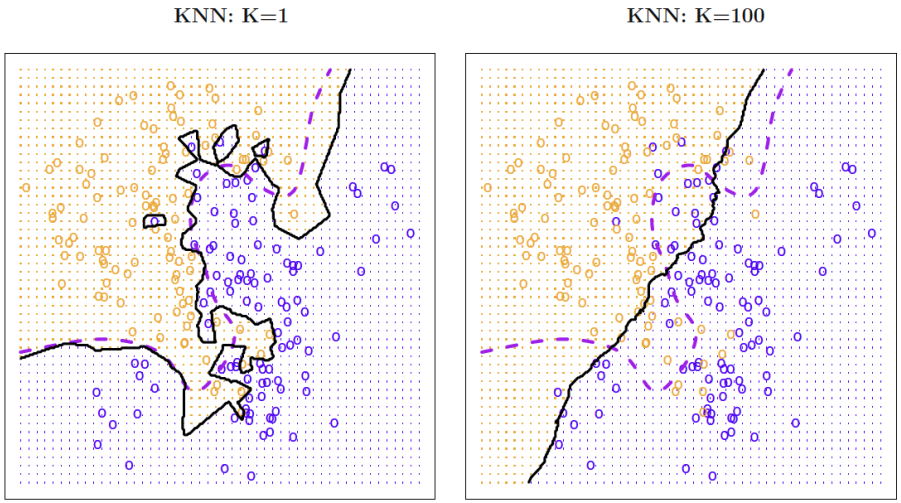
- Standardization: we determine the Z-scores of the data.

$$x_{i,standardized} = (x_i - mean(x))/sd(x)$$

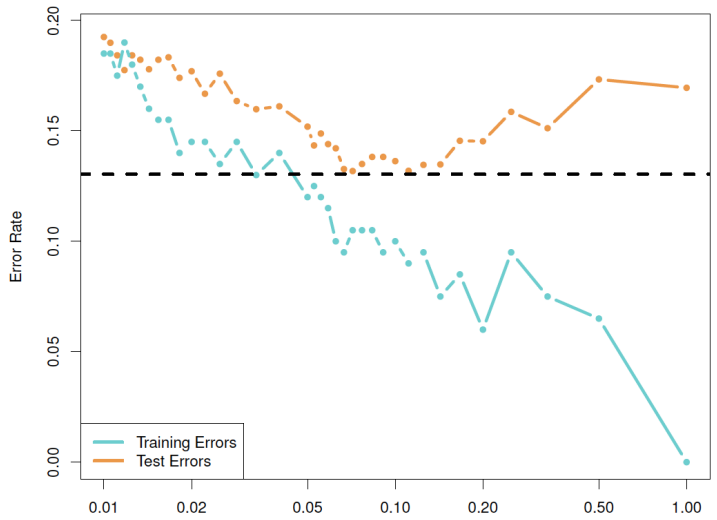
  Standardization is preferable to normalization if the criteria are unbounded (e.g., ranging from 0 to ∞) and/or there are outliers in either direction

The choice of K

The choice of K has a drastic effect on the KNN classifier.



The following plot gives the KNN test/training errors v.s. $1/K$.



Application of KNN in R

Let's apply the KNN algorithm step by step to an example. The steps are as follows:

1. Reading and describing the data
2. Preparing the data: Normalize (or standardize) the data and splitting the data into a training set and a test set
3. Training the model on the data
4. Evaluating the model
5. Improving the model.

STAT562-L3-KNN

Loading data

Iris dataset consists of 50 samples from each of 3 species of Iris (Iris setosa, Iris virginica, Iris versicolor) and a multivariate dataset introduced by British statistician and biologist Ronald Fisher in his 1936 paper. Four features were measured from each sample i.e length and width of the sepals and petals and based on the combination of these four features, we'd like to develop a KNN model to distinguish the species from each other. The data is built-in with R base package.

```
data(iris)
```

Structure of the data

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 ...
```

Normalization

The normalization step is optional for this data set because the features (width and length) are measured with the same scale. But you may apply normalize anyway.

```
normalize = function(x) {
return ((x - min(x)) / (max(x) - min(x))) }
iris.normalized=apply(iris[,1:4],2,normalize)
```

Splitting data into train and test data

I did a 70-30 split here.

```
n=nrow(iris)
set.seed(11)
train.index=sample(1:n,size=n*0.7,replace = FALSE)
train.x=iris.normalized[train.index,]
```

```
test.x=iris.normalized[-train.index,]
dim(train.x)
```

```
## [1] 105   4
```

```
dim(test.x)
```

```
## [1] 45  4
```

```
train.y=iris[train.index,5]
test.y=iris[-train.index,5]
```

Apply KNN

We will use the KNN function in class package. Load the library first.

```
library(class)
knn_3  = knn(train = train.x,
                      test = test.x,
                      cl = train.y,
                      k = 3)
```

##Calculate the errors with confusion matrix

Confusiin Matrix

```
cm <- table(test.y, knn_3)
cm
```

```
##             knn_3
## test.y       setosa versicolor virginica
##   setosa         14          0         0
##   versicolor      0         15         2
##   virginica       0          1        13
```

Calculate out of test error rates

```
test.error=mean(knn_3 != test.y)
test.error
```

```
## [1] 0.06666667
```

Repeat with different choice of K

```
knn_5  = knn(train = train.x,
                      test = test.x,
                      cl = train.y,
                      k = 5)
test.error=mean(knn_5 != test.y)
test.error
```

```
## [1] 0.04444444
```

```
knn_8  = knn(train = train.x,
                      test = test.x,
                      cl = train.y,
                      k = 8)
test.error=mean(knn_8 != test.y)
test.error
```

```
## [1] 0.02222222
```

```
knn_15  = knn(train = train.x,
                      test = test.x,
                      cl = train.y,
                      k = 15)
test.error=mean(knn_15 != test.y)
test.error
```

```
## [1] 0.04444444
```

```
knn_25  = knn(train = train.x,
                      test = test.x,
                      cl = train.y,
                      k = 25)
test.error=mean(knn_25 != test.y)
test.error
```

```
## [1] 0.08888889
```

Lecture 4: Logistic Regression

Beidi Qiang

SIUE

Introduction

The linear regression model that we discussed in STAT560 assumes that the response variable is numerical that takes values over the real line. But in many situations, we require a probability estimate (a value between 0 and 1) as output.

- Predicting probability of Heads for bent coins. You might use features like angle of bend, coin mass, etc.
- An online banking service must be able to determine the probability a transaction being performed on the site is fraudulent, on the basis of features such as the IP address, past transaction history, and so forth.

The Logistic Model

Consider a binary response Y. Using the generic 0/1 coding for the response. We want to model the relationship between $p(X) = Pr(Y = 1|X)$ and the p-dimensional features $X = (X_1, \cdots, X_p)$.

- We can't use a simple linear model: $p(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$, since there is no guarantee the predicted value will be a proper probability, i.e. in between 0 and 1
- We must model $p(X)$ using a function that gives outputs between 0 and 1 for all values of X.
- We use the following logistic (or sigmoid) function:

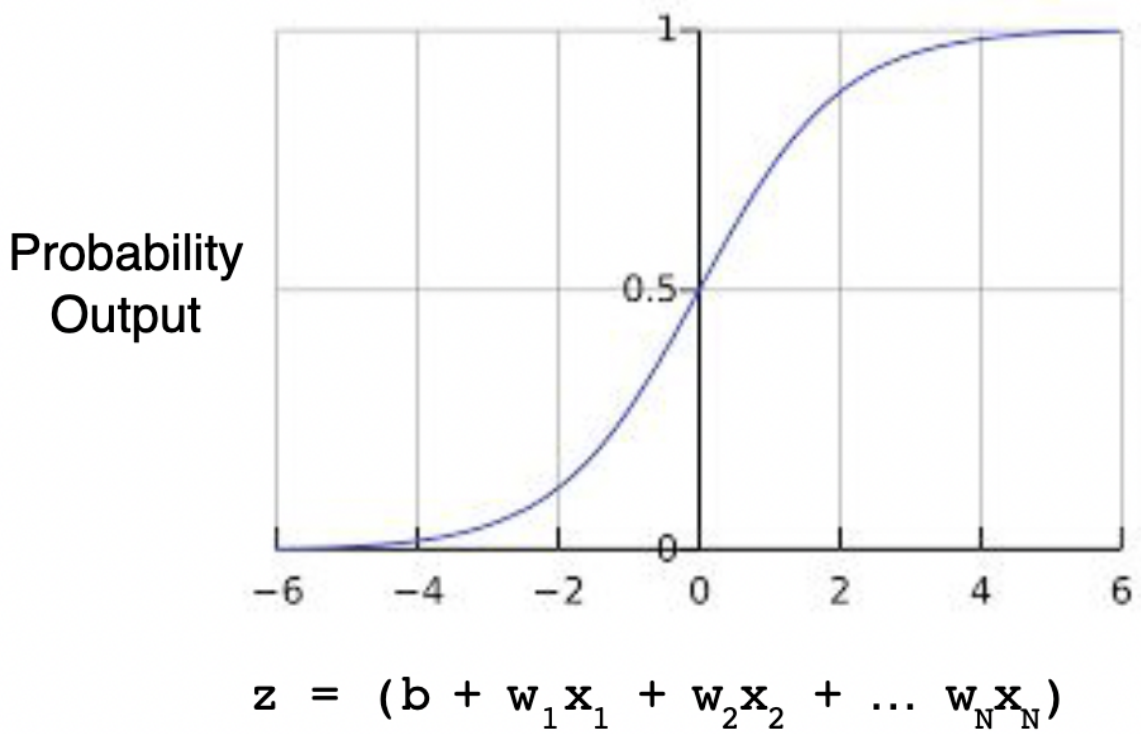
$$p(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p)}}.$$

The Sigmoid Function

The sigmoid function,

$$y = \frac{1}{1 + e^{-z}},$$

yields the following plot:



More on Logistic Model

- After a bit of manipulation of sigmoid function, we get

$$\log\left[\frac{p(X)}{1-p(X)}\right] = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

- The β values are the model's learned weights, sometimes also denoted as w.
- The quantity $p(X)/[1-p(X)]$ is called the odds, and can only be non-negative.
- Values of the odds close to 0, indicate low probabilities of $Y = 1$. Odds close to ∞, indicate high probabilities of $Y = 1$.
- The quantity $\log\left(p(X)/[1-p(X)]\right)$ is called the log-odds or logit.
- Logistic regression is just a linear regression on the log-odds.

Estimating the Model Weights

β's in the logistic regression model are unknown, and must be estimated based on the training data (labeled examples),

$$(x_1, y_1), \cdots, (x_n, y_n).$$

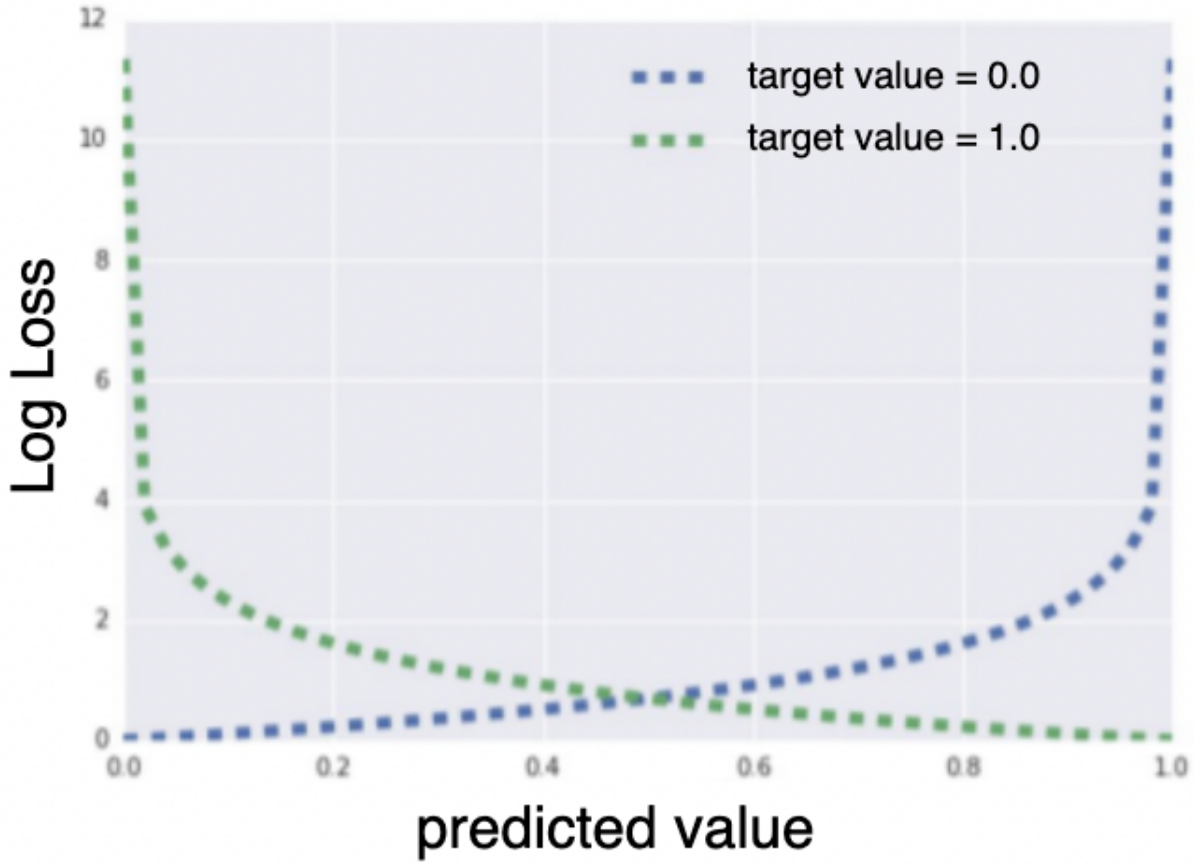
We determine β values by minimizing the loss, defined as follows:

$$logloss = \sum_{i=1}^{n} -y_i \log p(x_i) - (1 - y_i) \log(1 - p(x_i)),$$

$$\text{where } p(x_i) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip})}}.$$

- We seek estimates of β such that the predicted probability $p(x_i)$ close to one for all individuals with $y = 1$, and close to zero for all individuals with $y = 0$.

The log loss


target value = 0.0
target value = 1.0
Log Loss
predicted value
0
2
4
6
8
10
12
0.0
0.2
0.4
0.6
0.8
1.0

Example: Default data

The data deals with whether an individual will default on his or her credit card payment, on the basis of annual income, monthly credit card balance and student status. Logistic model out put from R is given below:

```
Call:
glm(formula = default ~ balance + income + student, family = binomial,
    data = Default)

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.087e+01  4.923e-01 -22.080  < 2e-16 ***
balance      5.737e-03  2.319e-04  24.738  < 2e-16 ***
income       3.033e-06  8.203e-06   0.370  0.71152
studentYes  -6.468e-01  2.363e-01  -2.738  0.00619 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Model Coefficient Interpretation

- In logistic regression, the relationship between $p(X)$ and X is not a straight line.
- Increasing X_1 by one unit changes the log odds by β_1, or equivalently it multiplies the odds by e^{β_1}.
- If β_1 is positive then increasing X_1 will be associated with increasing $p(X)$, and vice versa.
- The rate of change in probability of $Y = 1$ per unit change in X is not a constant value. It depends on the current value of X since the relationship is not linear.

Significance of Coefficients

Many aspects of the logistic regression output are similar to the linear regression output.

- We measure the accuracy of the coefficient estimates by computing their standard errors (details omitted).
- The z-statistic $= \hat{\beta}/SE(\hat{\beta})$ plays the same role as the t-statistic in the linear regression. A large (absolute) value of the z-statistic indicates evidence against $\beta = 0$.
- $\beta_1 = 0$ in logistic regression implies the probability of $Y = 1$ does not depend on X.
- If p-value associated with β_1 is small enough, we conclude that there is indeed an association between X_1 and probability of $Y = 1$.

Making Prediction

Once the coefficients have been estimated, it is a simple matter to compute the probability of $Y = 1$ for any given $X = x$.

$$\hat{p}(Y = 1|X = x) = \hat{p}(x) = \frac{1}{1 + e^{-(\hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p)}}.$$

In example, for a non-student user with balance \$1000 and income \$50000, the predicted probability of default is

$$\hat{p}(default) = \frac{1}{1 + e^{-(-10.87 + 0.005737 * 1000 + 0.000003033 * 50000)}} = 0.0068.$$

Categorical Features

If X is a categorical feature, we creates dummy variables and treat it the same way as in the linear regression, for example,

$$\hat{p}(Y = 1|X = 1) = \frac{1}{1 + e^{-(\hat{\beta}_0 + \hat{\beta}_1)}},$$

$$\hat{p}(Y = 1|X = 0) = \frac{1}{1 + e^{-\hat{\beta}_0}}.$$

For example, being a student, versus a non-student the log odds of default decreases by 0.6468

Logistic Regression for Multi-level Response

- With a response of more than 2 levels (say k levels), we need to model the probability of each of $k-1$ levels.
- Set of models

$$\log\left[\frac{P(Y=1|X)}{P(Y=k|X)}\right]=\beta_1 X,\cdots,\log\left[\frac{P(Y=k-1|X)}{P(Y=k|X)}\right]=\beta_k X.$$

- This is called multinomial logit model.
- In practice multinomial logit model is not used all that often. We use discriminant analysis (next lecture) instead for multiple-class classification.

STAT562 Lecture 5: Discriminant Analysis

Beidi Qiang

SIUE

Introduction

Logistic regression involves directly modeling

$$P(class|feature) = Pr(Y = k|X = x),$$

the conditional distribution of the response Y given the predictor(s) X. We now consider an alternative approach, Discriminant Analysis.

- In Discriminant Analysis, we model $P(feature|class)$
- Discriminant analysis is more stable when the classes are well-separated.
- Discriminant model is again more stable when n is small with normally distributed predictor.
- Discriminant analysis is popular when we have more than two response classes.

Bayes' Thm

Suppose the categorical response variable Y can take on K possible classes. Bayes' theorem states that

$$p(class = k|feature = x) = Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{i=1}^{K} \pi_i f_i(x)}$$

- $\pi_k = P(Y = k)$ is called prior probability that a randomly chosen observation comes from the kth class.
- $f_k(x) = Pr(X = x|Y = k)$ denote the density function of X for an observation that comes from the kth class.
- $Pr(Y = k|X = x)$ is called the posterior probability that an observation belongs to the kth class, given the featrue value for that observation.
- Instead of directly computing $Pr(Y = k|X = x)$ like in the logistic regression, we seek estimates of π_k and $f_k(x)$.

Linear Discriminant Analysis for One Predictor

Assume that $p = 1$, that is, we have only one predictor.

- To estimate π_k: we simply compute the fraction of the training observations that belong to the kth class.
- Estimate $f_k(x)$: we assume that $f_k(x)$ has is a normal density function with parameter μ_k and common σ^2.

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

- we estimate μ_k and σ^2. The common estimators are just the sample mean and variance in each class.

$$\hat{\mu}_k = \frac{1}{n_k}\sum_{i:y_i=k} x_i$$

We further assume all classes share the same variance, then

$$\hat{\sigma}^2 = \hat{\sigma}_k^2 = \frac{1}{n-K}\sum_{k=1}^{K}\sum_{i:y_i=k}(x_i-\hat{\mu}_k)^2$$

Discriminant Function

$$\hat{p}_k(x) = \hat{P}(class = k|x) = \frac{\hat{\pi}_k \frac{1}{\sqrt{2\pi}\hat{\sigma}} exp(-(x-\hat{\mu}_k)^2/2\hat{\sigma}^2)}{\sum_{i=1}^{K} \hat{\pi}_i \frac{1}{\sqrt{2\pi}\hat{\sigma}} exp(-(x-\hat{\mu}_i)^2/2\hat{\sigma}^2)}$$

The LDA classifier will assign an observation to the class for which $\hat{P}(class = k|x)$ is largest.

- Taking the log of $\hat{p}_k(x)$ and rearranging the terms, it is not hard to show that this is equivalent assign class that gives the largest

$$\hat{\delta}_k(x) = x \cdot \frac{\hat{\mu}_k}{\hat{\sigma}^2} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + \log(\hat{\pi}_k)$$

- Note $\hat{\delta}_k(x)$ is called "the discriminant functions", which are linear in x.

A simple example

$K = 2$ and $n_1 = n_2 = 20$ observations were drawn from each of the two classes. The mean and variance parameters for the two density functions are $\mu_1 = -1.25$, $\mu_2 = 1.25$, and $\sigma_1 = \sigma_2 = 1$.

- $\hat{\pi}_1 = \hat{\pi}_2 = 0.5$
- $\hat{\delta}_1(x) = (x\hat{\mu}_1 - \hat{\mu}_1^2/2)/\sigma^2$, and $\hat{\delta}_2(x) = (x\hat{\mu}_2 - \hat{\mu}_2^2/2)/\sigma^2$
- Assigns an observation to class 1 if $2x(\hat{\mu}_1^2 - \hat{\mu}_2^2) > \hat{\mu}_1 - \hat{\mu}_2$, i.e. $x > \frac{\hat{\mu}_1+\hat{\mu}_2}{2}$ is assigned to one class and $x < \frac{\hat{\mu}_1+\hat{\mu}_2}{2}$ is assigned to another.
- $x = \frac{\hat{\mu}_1+\hat{\mu}_2}{2}$ is the decision boundary.

A simple example, cont.

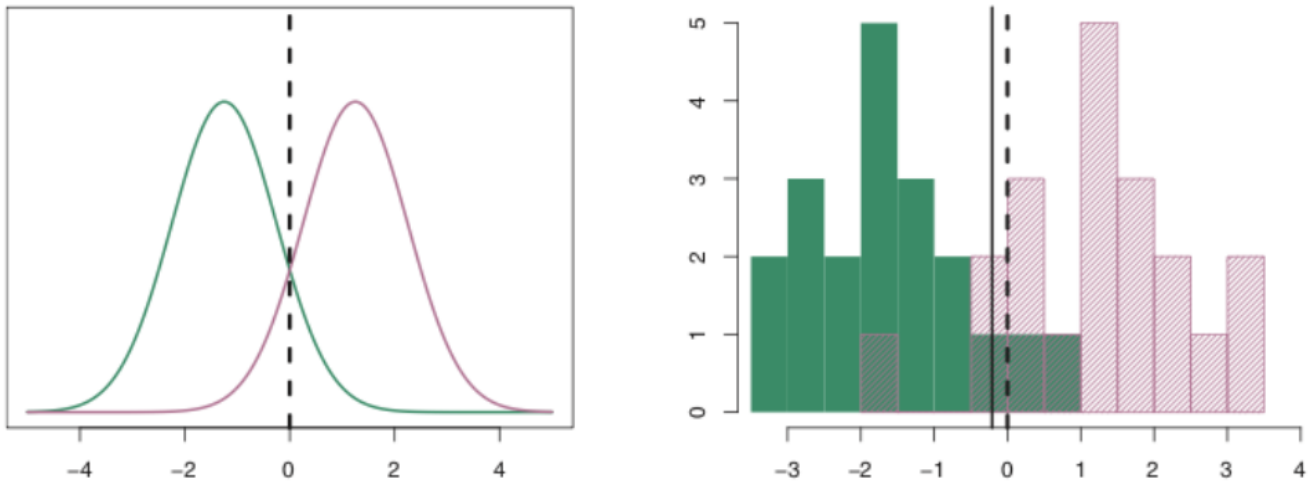


FIGURE 4.4. Left: *Two one-dimensional normal density functions are shown. The dashed vertical line represents the Bayes decision boundary.* Right: *20 observations were drawn from each of the two classes, and are shown as histograms. The Bayes decision boundary is again shown as a dashed vertical line. The solid vertical line represents the LDA decision boundary estimated from the training data.*

LDA for Multi-dimensional feature

Extend the LDA classifier to the case of multi-dimensional features.

- We assume the observations in the kth class are drawn from a multivariate Gaussian distribution $N(\mu_k, \Sigma)$ with pdf:

$$f_k(x) = \frac{1}{\sqrt{(2\pi)^p |\Sigma|}} exp\left[-\frac{1}{2}(x-\mu_k)^T \Sigma^{-1}(x-\mu_k) \right]$$

- μ_k is a p-dimensional class-specific mean vector, and Σ is a $p \times p$ covariance matrix that is common to all K classes.
- We need to estimate the unknown parameters μ_k and Σ; the formulas are similar but more technical, we will skip the details.

Prediction

The LDA classifier will again assign an observation to the class for which $\hat{p}_k(x)$ is largest.

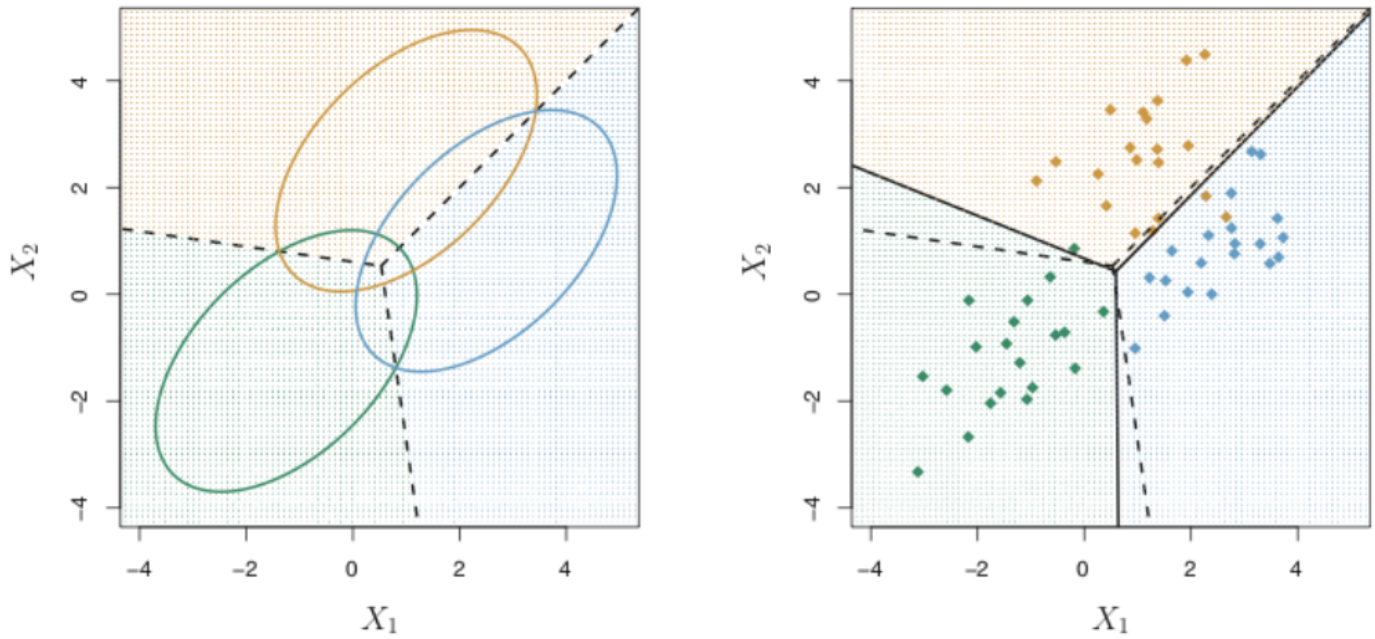
- Taking the log and rearranging the terms, we can show that this is equivalent assign class that gives the largest

$$\hat{\delta}_k(x) = x^T \hat{\Sigma}^{-1} \hat{\mu}_k - \frac{1}{2} \hat{\mu}_k^T \hat{\Sigma}^{-1} \hat{\mu}_k + \log(\hat{\pi}_k)$$

- This is the vector/matrix version of the $\hat{\delta}_k(x)$ in single predictor case.

An example with three classes

Simulated data. Left: The true underling distribution. Right: LDA classification. LDA decision boundaries are indicated using solid black lines.



Example: Default data

The data deals with whether an individual will default on his or her credit card payment, on the basis of annual income, monthly credit card balance and student status. LDA model out put from R is given below:

```
> lda.out=lda(default~income+student+balance)
> lda.out
Call:
lda(default ~ income + student + balance)

Prior probabilities of groups:
    No    Yes
0.9667 0.0333

Group means:
      income studentYes   balance
No  33566.17  0.2914037  803.9438
Yes 32089.15  0.3813814 1747.8217

Coefficients of linear discriminants:
                    LD1
income      3.367310e-06
studentYes -1.746631e-01
balance     2.243541e-03
> lda.class=predict(lda.out,Default)$class
> table(lda.class,default)
         default
lda.class   No  Yes
      No  9645  254
      Yes   22   79
```

Quadratic Discriminant Analysis

Like LDA, the QDA classifier results from assuming that the observations from each class are drawn from a normal distribution, but unlike LDA, QDA assumes that each class has its own covariance matrix.

- QDA assumes that an observation from the kth class is of the form $N(\mu_k, \Sigma_k)$
- The QDA classifier will again assign an observation to the class for which $\hat{p}_k(x)$ is largest, which now is equivalent assign class that gives the largest

$$\hat{\delta}_k(x) = -\frac{1}{2}x^T\hat{\Sigma}_k^{-1}x + x^T\hat{\Sigma}_k^{-1}\hat{\mu}_k - \frac{1}{2}\hat{\mu}_k^T\hat{\Sigma}_k^{-1}\hat{\mu}_k + \log(\hat{\pi}_k)$$

- Note the discriminant functions $\hat{\delta}_k(x)$, are now quadratic in x.

Naive Bayes

Unlike LDA and QDA classifier that assumes the probability distribution $Pr(x|class)$ follows a p-dimensional multivariate normal distribution, Naive Bayes classifier assumes the p features are independent, i.e.

$$f_k(x) = f_{k1}(x_1) f_{k2}(x_2) \cdots f_{kp}(x_p)$$

To estimate the one-dimensional density function f_{kj}:

- If X_j is quantitative, then we can assume a univariate normal distribution for each feature. This is similar to QDA, but with an additional assumption that the covariance matrix Σ_k is diagonal.
- If X_j is quantitative, another option is to use a non-parametric estimate, such as kernel density estimator.
- If X_j is qualitative, then we can simply count the proportion of training observations for the jth predictor corresponding to each class.

Comparison of LDA and Logistic Regression

We have considered different classification approaches: logistic regression, LDA, and QDA, and K-nearest neighbors (KNN) method. How do they compare?

- In the case of two-class setting, logistic regression and LDA methods are closely connected, since the log-odds in logistic framework can also be expressed as a linear function of x, i.e. $\log[p_1(x)/p_2(x)] = c_0 + c_1 x$. The difference is only in their fitting procedures.
- LDA assumes a Gaussian distribution with a common covariance matrix. It performs better when this assumption approximately holds.
- Logistic regression can outperform LDA if these Gaussian assumptions are not met.

Compared with KNN

KNN takes a completely different approach. It is a non-parametric approach.

- KNN makes no assumptions about the shape of the decision boundary, or distribution of predictors.
- KNN will perform the best when the decision boundary is highly non-linear.
- KNN does require a larger training set so form an accurate decision boundary
- LDA and logistic regression performs better when the decision boundary is approximately linear and/or when training set is small.
- We lose some interpretability with such a non-parametric approach. We can't discuss the effect of predictors on the response with the KNN approach.

Compared with QDA

QDA serves as a compromise between the completely non-parametric KNN method and the linear LDA.

- QDA allows curvature by assuming a quadratic decision boundary.
- The QDA boundary is not as flexible as KNN.
- But QDA performs better than KNN when training set is small. Because it makes assumption of the form of the decision boundary.

Types of errors

A confusion matrix is a convenient way to display error information.

- Two types of errors: it can incorrectly assign an individual who defaults to the no default category, or it can incorrectly assign an individual who does not default to the default category.
- Note the error rate is low (22/9667) for individuals who did not default, but (unacceptably) high (254/333) among individuals who defaulted.
- Why does LDA do such a poor job of classifying the customers who default?
  LDA is trying to get the lowest total error rate, i.e. smallest possible total number of misclassified observations, irrespective of which class the errors come from!

STAT 562 Lecture 6 Evaluating Classification Model

Beidi Qiang

SIUE

Thresholding

Many classification models returns a probability. You can use the returned probability "as is" or convert the returned probability to a binary value.

- To map a logistic regression value to a binary category, you must define a classification threshold.
- Value above that threshold indicates positive class; a value below indicates negative class 0.
- Thresholds are problem-dependent, and are therefore values that you must tune.
- Choosing a threshold is assessing how much you'll suffer for making a mistake.

Accuracy

Accuracy is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right.

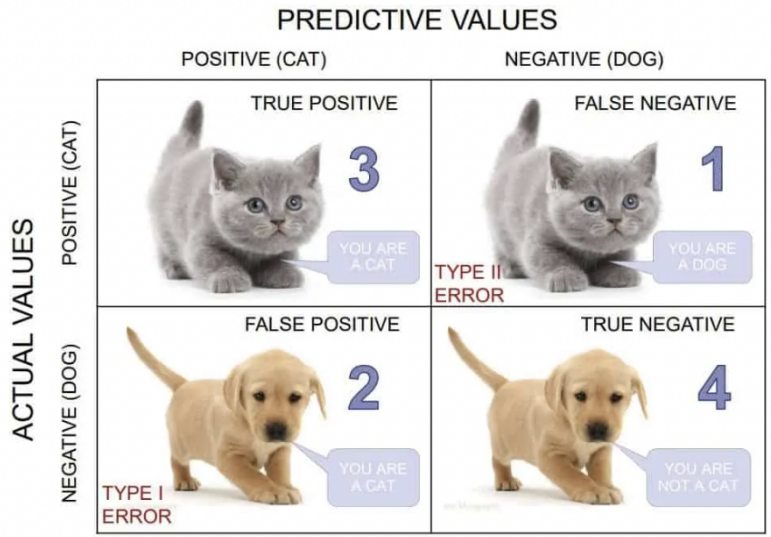
$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

In many cases, accuracy is a poor or misleading metric.

- Most often when different kinds of mistakes have different costs
- Class imbalance, when positives or negatives are extremely rare

True Positives and False Positives

For class-imbalanced problems, useful to separate out different kinds of errors using a 2×2 confusion matrix (error-matrix):



- A true positive (TP) is an outcome where the model correctly predicts the positive class.
- A true negative (TN) is an outcome where the model correctly predicts the negative class.
- A false positive (FP) is an outcome where the model incorrectly predicts the positive class.
- A false negative (FN) is an outcome where the model incorrectly predicts the negative class.

Precision and Recall

Using the terms of positives and negatives, we define the following metrics.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

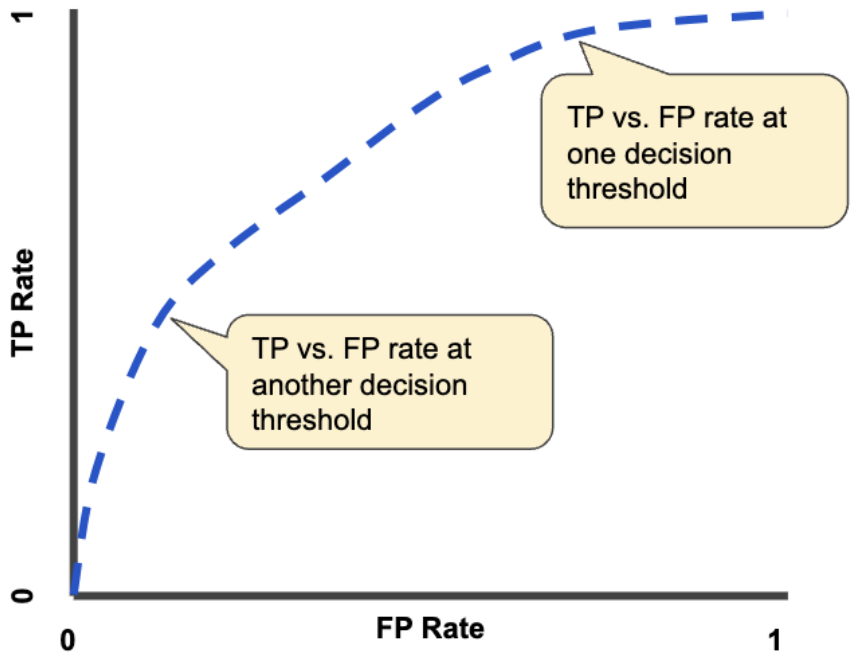
- Precision is the proportion of positive identifications was actually correct.
- Recall is the proportion of actual positives was identified correctly.

ROC curve

An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

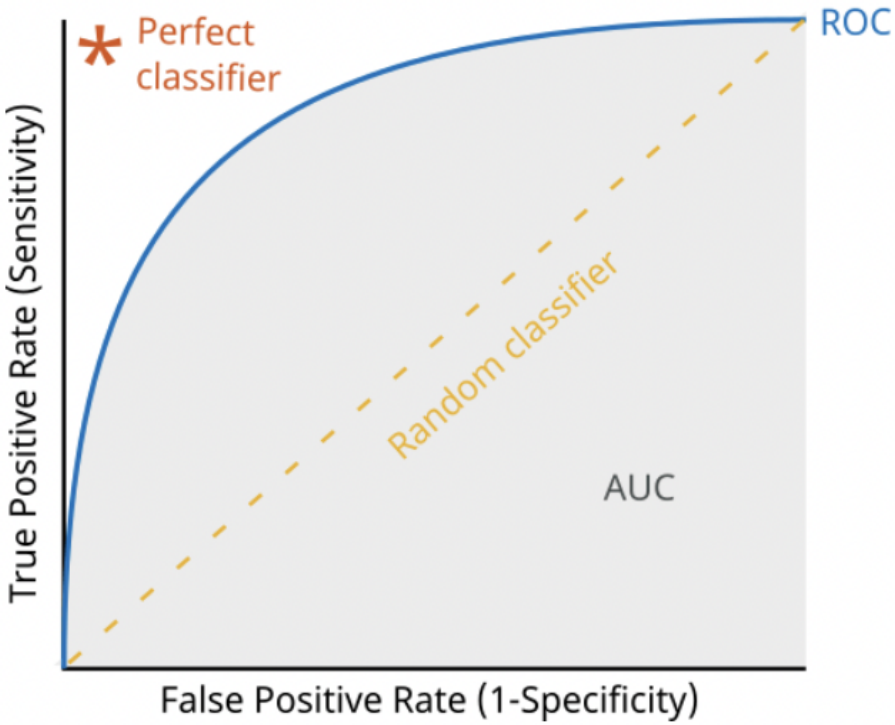
- True Positive Rate (TPR) = TP/(TP+FN). This is the same as recall.
- False Positive Rate (FPR) = FP/(FP+TN).

An ROC curve plots TPR vs. FPR at different classification thresholds.



AUC

Area under the ROC Curve (AUC) measures the entire two-dimensional area underneath the entire ROC curve (0,0) to (1,1).



AUC provides an aggregate measure of performance across all possible classification thresholds. It represents the probability that a random positive example has higher score (predicted probability to be positive) than a random negative example.

STAT562 Lecture 7 Cross Validation

Beidi Qiang

SIUE

Introduction

Cross-Validation is a well-used resampling method, which involves repeatedly drawing samples from set and refitting a model in order to obtain additional information about the fitted model.

- Cross-Validation is used to estimate the test error associated with a given model.
- Very general method, and can be used with any kind of predictive modeling.
- It is used for model assessment, i.e. evaluating a model's performance.
- It can also be used for model selection, i.e. selecting the proper level of model flexibility, or tuning the model parameters

Training and Test Error

- The test error is the average error that results from using a trained model to predict the response on a new set of observation
- The training error is the average error by applying the trained model to the observations used in its training.
- The training error rate often can be quite different from the test error rate, especially when overfitting.
- The test error can be easily calculated if a designated test set is available, what if not?

The Training - Validation - Test split

Dividing the available set of observations into three parts, a training set, validation set (optional) and a test set.

- First, the model is fit on the training set.
- If you'd like to tweak model hyper-parameters or perform model selection, use the fitted model to predict the labels for the observations in the validation set. Tweak the model based on the errors in the validation set. This step can be optional depending on whether model tweaking is needed.
- Then the final model is used to predict the labels for the observations in the test set. And test error rate (or MSE) can be calculated.

Drawbacks

- The estimate of the test error rate can be highly variable, depending how the data is slit into training and test.
- Only a subset of the observations (those that are included in the training set) are used to fit the model. The model training tends to suffer from reduced number of training observations.

Leave-One-Out Cross-Validation

Leave-one-out cross-validation (LOOCV) involves splitting the set of training samples into two parts repeatedly. A test set should still be held out for final evaluation, but the validation set is no longer needed when doing. At each iteration i,

- The ith observation is used as the validation set, and the remaining observations make up the training set.
- The model is fit on the $n-1$ training observations, and a prediction is made for the excluded ith observation, which provides a test error estimate (MSE_i).

Repeat the steps for each observation i, and the LOOCV estimate for the test MSE is the average of these n test error estimates:

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

The LOOCV schematic

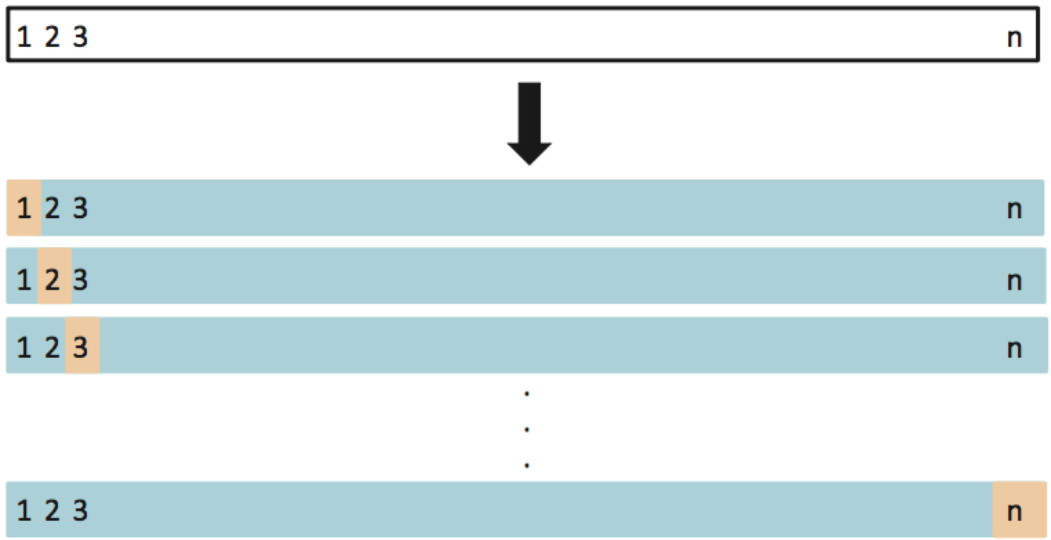


FIGURE 5.3. *A schematic display of LOOCV. A set of n data points is repeatedly split into a training set (shown in blue) containing all but one observation, and a validation set that contains only that observation (shown in beige). The test error is then estimated by averaging the n resulting MSE's. The first training set contains all but observation 1, the second training set contains all but observation 2, and so forth.*

Advantages and Drawbacks of LOOCV

- We repeatedly fit the model using the $n-1$ training data, almost as many as are in the entire data set.
- There is no randomness in the training/validation set splits, LOOCV give the same result each time.
- However, LOOCV can be very time consuming if n is large, since the model has to be fit n times and each fitting procedures can be computationally intensive.
- The MSE_i's from LOOCV are highly (positively) correlated with each other since the model is trained each time on an almost identical set of observations.

k-Fold Cross-Validation

K-fold CV involves randomly dividing the set of observations into k groups, or folds, of approximately equal size. At each iteration i,

- The ith fold is treated as a validation set, and the model is fit on the remaining $k-1$ folds.
- The estimated error rate Err_i or MSE_i is then computed on the observations in the held-out fold.

Repeat the steps for each fold i, and the K-fold CV estimate for the test error rate is the average of these k test error estimates:

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} MSE_i \text{ or } CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} Err_i$$

The k-Fold Cross-Validation schematic

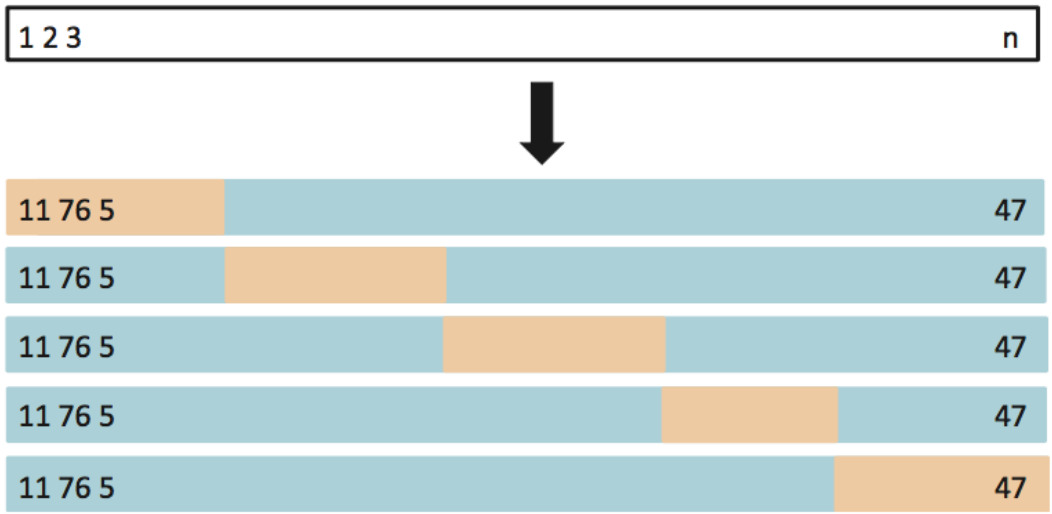


FIGURE 5.5. *A schematic display of 5-fold CV. A set of n observations is randomly split into five non-overlapping groups. Each of these fifths acts as a validation set (shown in beige), and the remainder as a training set (shown in blue). The test error is estimated by averaging the five resulting MSE estimates.*

Comments on k-Fold Cross-Validation

- LOOCV is a special case of k-fold CV in with $k = n$
- k-Fold CV requires fitting the learning procedure only k times, which provides a computational advantage over LOOCV.
- The MSE_i's from k-Fold CV are less correlated.
- There is some variability in the k-fold CV estimates as a result of the randomness in how the observations are divided into folds. This variability is lower with larger k.
- Typically choice of k are $k = 5$ or $k = 10$. these values have been shown empirically to yield good test error rate estimates.

Example: Iris data

```
> data(iris)
> library(caret)
> train.y=iris[,5]
> train.x=iris[,1:4]
> model = train(train.x,train.y,
+               method = "knn",
+               preProcess = c("center", "scale"),
+               tuneLength = 10,
+               trControl = trainControl(method = "cv",
+                                        number = 10))
> print(model)
k-Nearest Neighbors

150 samples
  4 predictor
  3 classes: 'setosa', 'versicolor', 'virginica'

Pre-processing: centered (4), scaled (4)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 135, 135, 135, 135, 135, 135, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   5  0.9466667  0.92
   7  0.9666667  0.95
   9  0.9533333  0.93
  11  0.9533333  0.93
  13  0.9666667  0.95
  15  0.9733333  0.96
  17  0.9600000  0.94
  19  0.9466667  0.92
  21  0.9600000  0.94
  23  0.9533333  0.93

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 15.
```

STAT562 Lecture 8 The Bootstrap

Beidi Qiang

SIUE

Introduction

The Bootstrap is a widely applicable resampling method to quantify the uncertainty associated with a given estimator.

- Can be used to estimate the standard errors of model coefficients (for example in regression models)
- Can be easily applied to a wide range of statistical learning methods
- Especially useful when a measure of variability is otherwise difficult to obtain

Idea of Boostrap

- Recall the concept of population and sample, sample statistics, sampling distribution.
- The observed sample should be similar to the true population.
- So we repeatedly resample from the observed sample (pseudo-population) to mimic new samples from the population.
- We calculate the desired quantity based on each of the bootstrap datasets.
- We can then get the standard error and construct confidence interval based on the bootstrap estimates.

The Bootstrap schematic

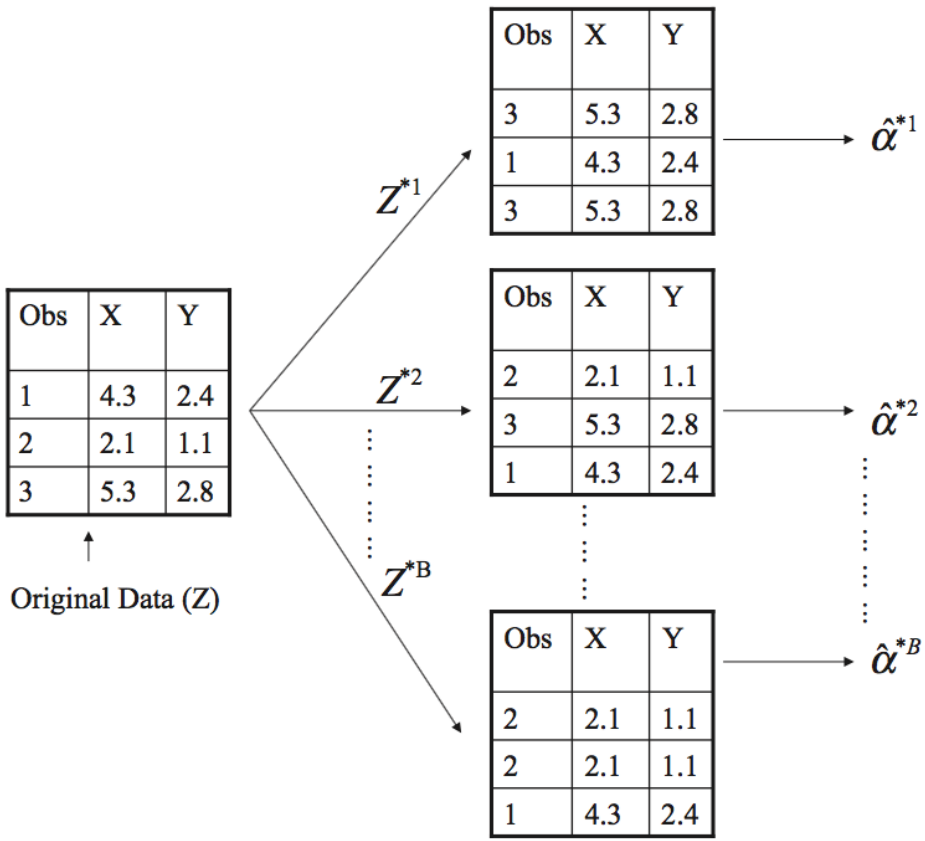


FIGURE 5.11. *A graphical illustration of the bootstrap approach on a small sample containing $n = 3$ observations. Each bootstrap data set contains n observations, sampled with replacement from the original data set. Each bootstrap data set is used to obtain an estimate of α.*

Algorithm

In each iteration $b = 1, 2, \cdots, B$,

- generate bootstrap sample $(x_1^{(b)}, \cdots, x_n^{(b)})$ by resampling with replacement from the original data $(x_1, \cdots x_n)$.
- Compute the desired quantity $\hat{\theta}^{(b)}$ from the bootstrap sample.

$\hat{\theta}^{(1)} \ldots \hat{\theta}^{(B)}$ is approximately a sample from the sampling distribution of $\hat{\theta}$. And can be used to estimate $SE(\hat{\theta})$.

Comments

- We don't make any assumption of the population distribution. So bootstrap is completely non-parametric.
- The number of iteration B is usually a large number, so bootstrap is computationally demanding.
- If the original sample is not a good representative of the population, the bootstrap sample will not be either. Increasing B will make to bootstrap samples closer to the **empirical distribution** given by the observed sample , but not **the population distribution**.

Example: Bootstrap from a Poisson

```
set.seed(17)
x=rpois(10, lambda=2)
table(x)/10
x.uniq = unique(x)
prob0 = as.data.frame(table(x))[,2]/length(x)
m=1000
x.star= sample(x.uniq, size = m, replace = TRUE, prob = prob0)
table(x.star)/m
```

Bootstrap estimation of std. error

- Recall we generate bootstrap sample $(x_1^{(b)}, \cdots, x_n^{(b)})$ and compute the desired quantity $\hat{\theta}^{(b)}$ from each of the bootstrap sample.
- sample standard deviation of $\hat{\theta}^{(1)} \cdots \hat{\theta}^{(B)}$ is an estimate of the $SE(\hat{\theta})$. i.e.

$$\hat{SE}(\hat{\theta}) = \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}(\hat{\theta}^{(b)} - \bar{\theta}^*)^2}, \text{ where } \bar{\theta}^* = mean(\hat{\theta}^{(1)} \cdots \hat{\theta}^{(B)})$$

- a choice of $B = 200$ is usually enough for estimating std. err, but it will take a larger B if seeking good estimation of confidence interval (Efron, 1993).

Example: Std .Err of Regression Coefficients

```
library(boot)

bs =function(formula, data, indices) {
d =data[indices,]
fit = lm(formula, data=d)
return(coef(fit)) }

results=boot(data = mtcars, statistic = bs, R =500, formula = mpg
wt + hp)
```

Bootsrap Confidence Intervals

- Based on the bootstrap estimation of std.error and CLT assumption, we have Standard Normal Bootstrap CI:

$$\hat{\theta} \pm Z_{\alpha/2}\hat{SE}(\hat{\theta})$$

- Based on the percentiles of the generated bootstrap estimates $\hat{\theta}^{(1)} \dots \hat{\theta}^{(B)}$, we can get the percentile Bootstrap CI:

$$(\hat{\theta}_{\alpha/2}, \hat{\theta}_{1-\alpha/2})$$

- Another one based on the percentiles. basic Bootstrap CI:

$$(\hat{\theta} - (\hat{\theta}_{1-\alpha/2} - \hat{\theta}), \hat{\theta} + (\hat{\theta} - \hat{\theta}_{\alpha/2})) = (2\hat{\theta} - \hat{\theta}_{1-\alpha/2}, 2\hat{\theta} - \hat{\theta}_{\alpha/2})$$

Example: Logistic Regression

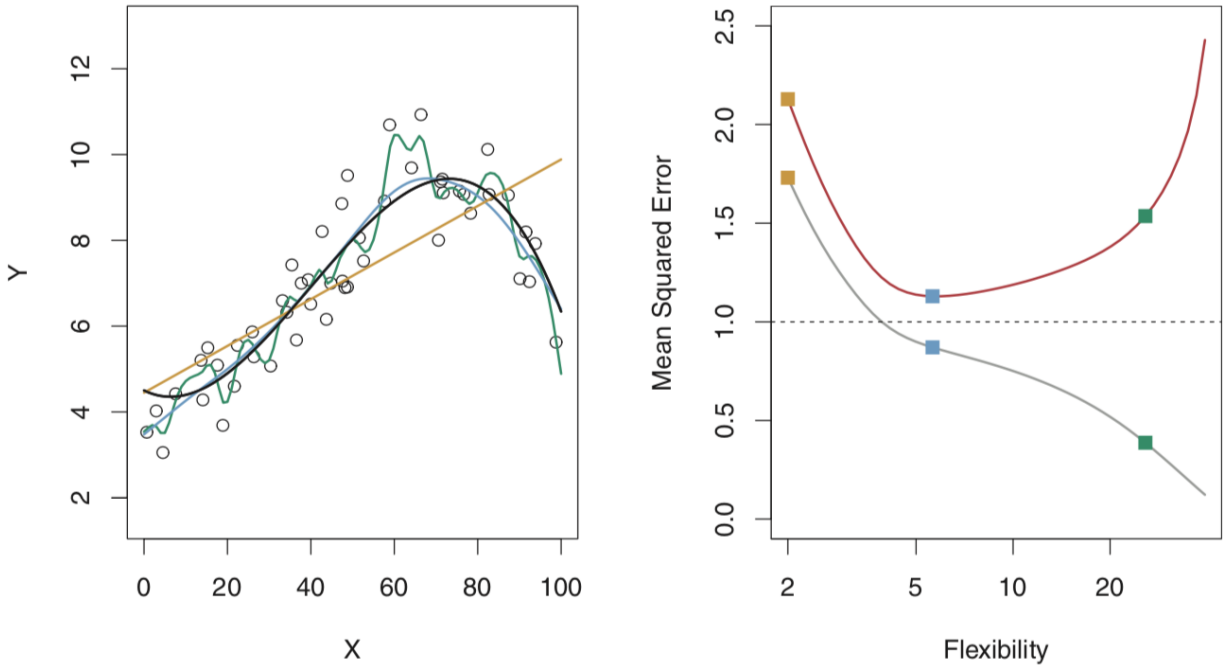
```
library(ISLR)
library(boot)

lr=function(data,indices){
d =data[indices,]
fit=glm(default balance+income,data=d,family="binomial")
return(coef(fit)) }

b=boot(Default,lr,100)
print(b)
summary(glm(default balance+income, data=Default,
family="binomial"))
plot(b,index=1)
plot(b,index=2)
boot.ci(b,index=1,type="perc")
boot.ci(b,index=2,type="norm")
```

STAT562 L9 Regularization

Beidi Qiang

SIUE

Recall the following curve, which shows the loss (error) for both the training set and test set against the model complexity.



You see that training loss gradually decreases, but test loss eventually goes up when we overfit. We could prevent overfitting by penalizing complex models, a principle called regularization.

Regularization Methods

Instead of simply aiming to minimize loss, we'll now minimize loss plus complexity, which is called structural risk minimization:

$$minimize\Big(Loss(Data|Model) + \lambda \times Complexity(model)\Big)$$

Our training optimization algorithm is now a function of two terms: the loss term, which measures how well the model fits the training data, and the regularization term, which measures model complexity.

Two common ways:

- Model complexity as a function of the weights of all the features in the model: use L_2 penalty for model weights.
- Model complexity as a function of the total number of features with nonzero weights: use L_1 penalty for model weights.

Regularization rate, λ

In practice, we'd like to tune the overall impact of the regularization term by multiplying its value by a scalar known as λ. This is also called the regularization rate.

- Increasing the λ value strengthens the regularization effect.
- When choosing a λ value, the goal is to strike the right balance between simplicity and training-data fit.
- The ideal value of λ is data-dependent, so you'll need to do some tuning, using a validation set or method like using cross-validation.

L2 Regularization and Ridge Regression

Suppose we have a model with weights $w_1, \cdots, w_p$.

$$\text{L2 regularization term} = ||\mathbf{w}||^2 = w_1^2 + \cdots + w_p^2$$

Apply this to the linear regression problem, where the model is

$$\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

Previously, we find the estimates of β using least squares estimates, which are the values minimizing

$$\text{SSE (sum of squared error)} = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} + \cdots \beta_p x_{ip})^2$$

Adding the L2 regularization, we now find estimates of βs by minimizing

$$SSE + \lambda \sum_{i=1}^{p} \beta_j^2$$

This is called a **Ridge Regression**.

Remarks: Ridge Regression

- Ridge regression is sensitive to the scale of predictors. We recommend standardizing predictors by their standard deviation.
- Bias-Variance trade-off: As λ increases, the flexibility of the ridge regression decreases, leading to decreased variance but increased bias.
- Will shrink estimated coefficients towards 0 but will not reach 0 exactly. Ridge regression will include all predictors in the final model.

L1 Regularization and the LASSO

In some problems, feature vectors can often be high-dimensional. It would be nice to encourage weights to drop to exactly 0 where possible. A weight of exactly 0 essentially removes the corresponding feature from the model. To achieve this, we introduce L1 regularization, that is penalizing the absolute value of all the weights.

$$\text{L1 regularization term} = |\mathbf{w}| = |w_1| + \cdots + |w_p|$$

Apply this regularization to the linear regression problem, we find estimates of βs by minimizing

$$SSE + \lambda \sum_{i=1}^{p} |\beta_j|$$

This is called **LASSO**.

Comparing Ridge and LASSO Constraints

An alternative formulation for Ridge and the Lasso is to minimize SSE subject to the constraint $\sum(\beta_j)^2 \leq s$ or $\sum|\beta_j| \leq s$, respectively.

- Consider $p = 2$. The constraint with LASSO is a diamond shape and the constraint with Ridge is a circle.

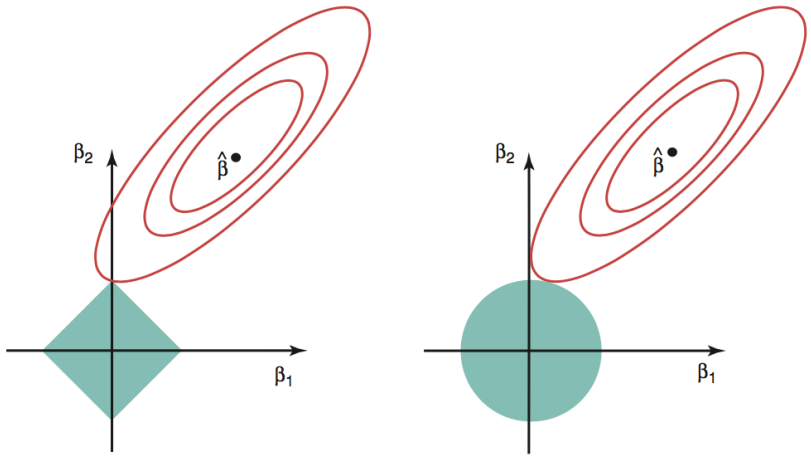


FIGURE 6.7. *Contours of the error and constraint functions for the lasso* (left) *and ridge regression* (right). *The solid blue areas are the constraint regions,* $|\beta_1| + |\beta_2| \leq s$ *and* $\beta_1^2 + \beta_2^2 \leq s$, *while the red ellipses are the contours of the RSS.*

Comparing Ridge and LASSO Cont.

- Lasso has similar behavior to ridge regression, in that as λ increases, the variance decreases and the bias increases.
- Lasso performs variable selection. It produces simpler models that are more interpretable.
- We expect the LASSO to perform better in a setting where a small number of predictors have substantial coefficients. And Ridge regression will perform better when the response is a function of many predictors, all with coefficients of roughly equal size.

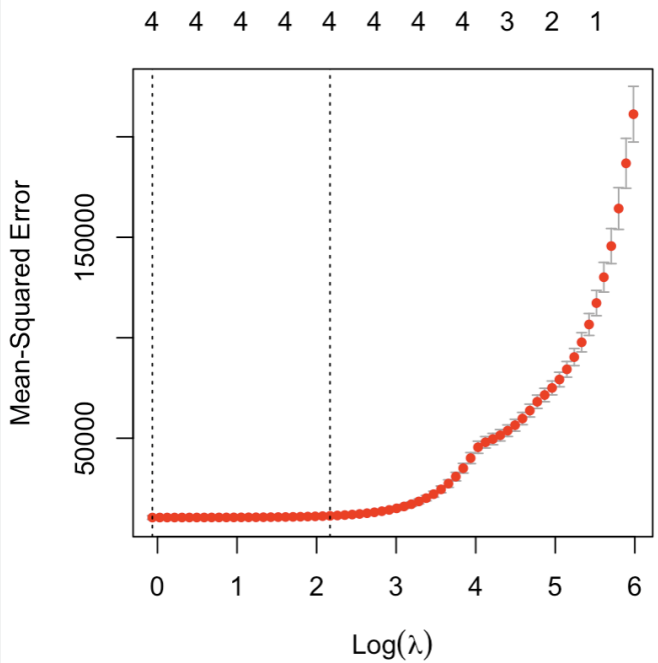
Example: Credit Data

```
> library(glmnet)
> x=model.matrix(Balance~Income+Limit+Rating+Student,data=Credit)[,-1]
> y=Credit$Balance
> ridge10=glmnet(x,y,alpha=0,lambda=10, standardize=T)
> coef(ridge10)
5 x 1 sparse Matrix of class "dgCMatrix"
                     s0
(Intercept) -482.630613
Income        -7.138192
Limit          0.127221
Rating         1.920824
StudentYes   411.816682
> ridge80=glmnet(x,y,alpha=0,lambda=80, standardize=T)
> coef(ridge80)
5 x 1 sparse Matrix of class "dgCMatrix"
                      s0
(Intercept) -349.6774040
Income        -3.7327840
Limit          0.0996093
Rating         1.4981519
StudentYes   350.2088421
> lasso10=glmnet(x,y,alpha=1,lambda=10, standardize=T)
> coef(lasso10)
5 x 1 sparse Matrix of class "dgCMatrix"
                      s0
(Intercept) -467.3726020
Income        -6.5616366
Limit          0.1155487
Rating         1.9674922
StudentYes   385.6329164
> lasso80=glmnet(x,y,alpha=1,lambda=80, standardize=T)
> coef(lasso80)
5 x 1 sparse Matrix of class "dgCMatrix"
                      s0
(Intercept) -205.25360726
Income         .
Limit          0.02819233
Rating         1.62980111
StudentYes   132.79423756
```

Selecting the Tuning Parameter

Cross-validation provides a simple way to selection tuning parameter. We choose a grid of λ values, and compute the cross-validation error for each value of λ.

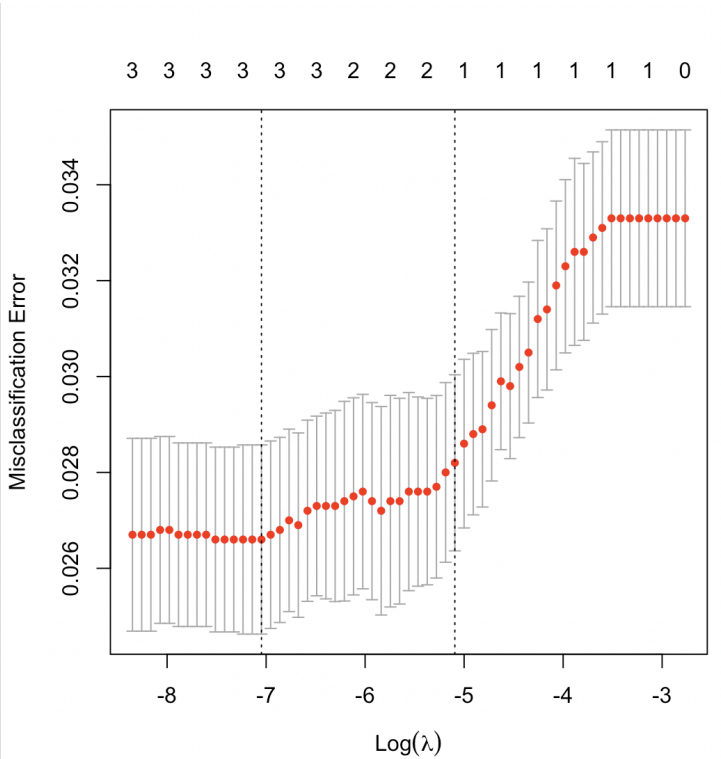
```
> library(glmnet)
> x=model.matrix(Balance~Income+Limit+Rating+Student,data=Credit)[,-1]
> y=Credit$Balance
> cv.lasso=cv.glmnet(x,y,alpha=1)
> plot(cv.lasso)
> cv.lasso

Call:  cv.glmnet(x = x, y = y, alpha = 1)

Measure: Mean-Squared Error

    Lambda Index Measure    SE Nonzero
min  0.938    66   10523 931.2       4
1se  8.745    42   11367 729.3       4
> cv.lasso$lambda.min
[1] 0.9376683
>
```



Applying Regularization to Logistic Regression

You may also apply the L1 or L2 regularization in logistic regression when we have a classification problem.

```
> x=model.matrix(default~.,data=Default)[,-1]
> y=Default$default
> lasso=glmnet(x,y,alpha=1,family="binomial",lambda=0.005, standardize=T)
> coef(lasso)
4 x 1 sparse Matrix of class "dgCMatrix"
                      s0
(Intercept) -9.113074995
studentYes  -0.119292143
balance      0.004561439
income       .
> cv=cv.glmnet(x,y,alpha=1,family="binomial",standardize=T,type.measure="class")
> cv

Call:  cv.glmnet(x = x, y = y, type.measure = "class", alpha = 1, family = "binom
ial",      standardize = T)

Measure: Misclassification Error

      Lambda Index Measure       SE Nonzero
min 0.000870    47  0.0266 0.001973       3
1se 0.006137    26  0.0282 0.001837       2
> plot(cv)
>
```

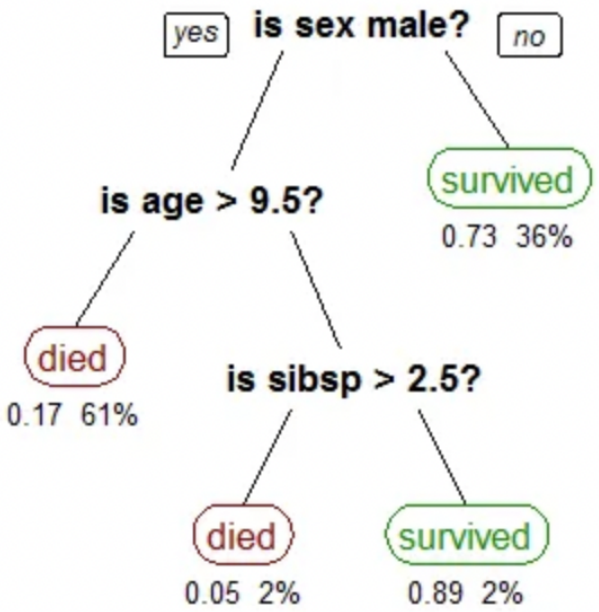


STAT562 Lecture 10: Decision Trees

Beidi Qiang

SIUE

Introduction



- Decision tree involves segmenting the feature space into a number of regions. The set of splitting rules used to segment the feature space is summarized in a tree.
- To make a prediction for a given example, we typically use the mean or the mode of the training examples in the region to which it belongs.
- Decision trees can be applied to both regression and classification problems .

Leaves, Branches and Nodes

- The regions hat doesn't split anymore are known as terminal nodes or leaves of the tree.
- Decision trees are typically drawn upside down, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split are referred to as internal nodes.
- We refer to the segments of the trees that connect the nodes as branches.

Building a Tree

Process of building a tree:

- We divide the feature space (the values of $X_1, X_2, \cdots, X_p$) into J distinct and non-overlapping regions, $R_1, R_2, \cdots, R_J$.
- For every example that falls into the region R_j , we make the same prediction, which is simply the mean (regression) or the mode (classification) of the label values for the training examples in R_j.

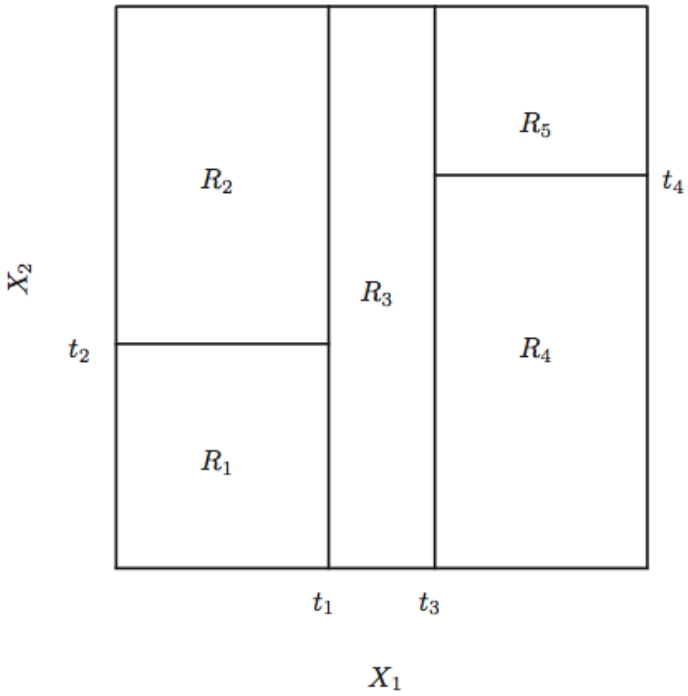
How do we choose regions $R_1, \cdots, R_J$?

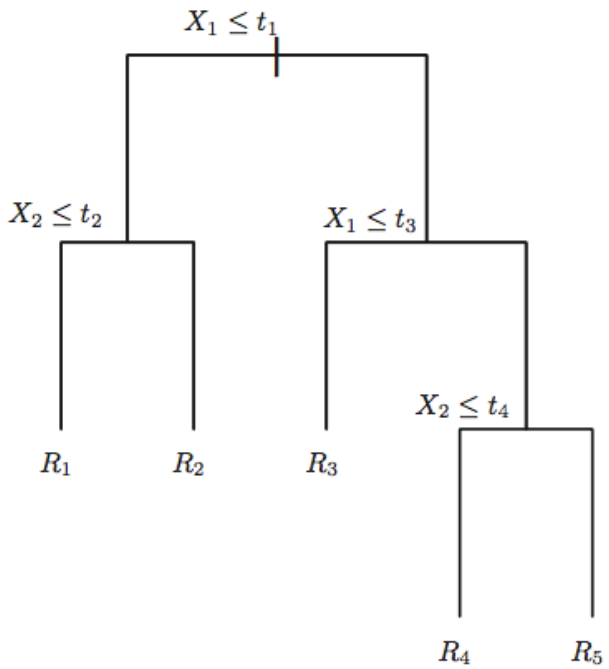
- The goal is to find boxes $R_1, \cdots, R_J$ that minimize the SSE.
- Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes. So we take a top-down approach that is known as "recursive binary splitting".

Algorithm: Recursive Binary Splitting

- We first select the predictor X_j and the cutpoint s such that splitting into the regions $R_1(j, s) = \{X|X_j < s\}$ and $R_2(j, s) = \{X|X_j \geq s\}$ leads to minimum cost.
- We repeat the process, looking for the best predictor and best cut point in order to split the data further so as to minimize the cost within each of the resulting regions (R_1 and R_2). We then split one of them.
- We now have three regions. Again, we look to split one of these three regions further, so as to minimize the cost.
- The process continues until a stopping criterion is reached.

An example of a five-region tree





Split Criterions for Trees

For Regression trees, we use

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

For Classification tree, we use one of the following:

- Gini index: $G = \sum_k \hat{p}_{mk}(1 - \hat{p}_{mk}) = 1 - \sum_k \hat{p}_{mk}^2$
- Cross-entropy: $D = -\sum_k \hat{p}_{mk} log(\hat{p}_{mk})$

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

Comments

Advantages:

- Regression tree model is easier to interpret and has a nice graphical representation.
- Trees can easily handle categorical features directly without the need to create dummy variables.

Disadvantages:

- Complex tree without pruning is likely to overfit the data, leading to poor test set performance.
- Given that decision trees take a greedy search approach during construction, they can be more expensive to train compared to other algorithms.
- Decision trees can be unstable. Small variations within data can produce a very different decision tree.

Tree Pruning

We like a smaller tree with not too many splits. The post-pruning strategy is to grow a very large tree T_0, and then prune it back in order to obtain a subtree. The following algorithm is called "Cost complexity pruning", also known as weakest link pruning.

- For a given tuning parameter α, we pick a subtree $T \in T_0$ that minimize the quantity $cost + \alpha|T|$, where $|T|$ is the number of terminal nodes of the tree. Here cost can be SSE in regression problems or deviance in classification problems.

$$deviance = -2\sum_n \sum_k n_{mk} log(\hat{p}_{mk})$$

- The tuning parameter controls a trade-off between the subtree complexity and its fit to the training data (similar to LASSO in linear regression).
- We can select a value of the tuning parameter α using a validation set or using cross-validation.

Summarized Algorithm to Build a Tree

- Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
- Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, for a sequence of values of α.
- Use K-fold cross-validation to choose α.
- Return the subtree that corresponds to the chosen α.

Example: Boston data

```
library(tree)

tree.b=tree(medv .,Boston)
summary(tree.b)
plot(tree.b);text(tree.b)

#pruning
cv.b=cv.tree(tree.b)
plot(cv.b$size, cv.b$dev, type="b")
prune.b=prune.tree(tree.b,best=5)
plot(prune.b);text(prune.b)
```

Example: Default Data

The tree library is used to construct classification and regression trees:

```
train=sample(1:10000,6000)
tree.d=tree(default .,Default,split="gini",subset=train)
summary(tree.d)
plot(tree.d);text(tree.d)

#pruning
cv.d=cv.tree(tree.d)
plot(cv.d$size, cv.d$dev, type="b")
prune.d=prune.tree(tree.d,best=5)
plot(prune.d);text(prune.d)

#predict class on test data
test=Default[-train,]
pred.d=predict(prune.d,test,type="class")
table(pred.d,test$default)
```

STAT562 Lecture 11: Bagging and Random Forests

Beidi Qiang

SIUE

Introduction

- A single decision trees suffer from high variance: if we split the training data into two parts at random, and fit a decision tree to both halves, the results that we get could be quite different.
- Bagging and random forests are designed for reducing the variance of a statistical learning method. These methods use trees as building blocks to construct more powerful prediction models.
- Idea: to reduce variance, we can average a set of observations. We can take many training sets from the population, build a separate prediction model using each training set, and average the resulting predictions.

Bagging in general

We want to take many training sets from the population. However, we generally do not have access to multiple training sets. So, instead, we can bootstrap.

- We take repeated samples from the training data set and generates B different bootstrapped training data sets.
- We then train our method on each bootstrapped training set in order to get $f^{*b}(x)$, $b = 1, \cdots, B$.
- Finally we average all the predictions, to obtain

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

.

Bagging with trees

Bagging can improve predictions for many regression methods, it is particularly useful for decision trees.

- In regression problems (quantitative outcome), we simply construct B regression trees using B bootstrapped training sets, and average the resulting predictions.
- In classification problems, we simply construct B classification trees using B bootstrapped training sets, record the class predicted from each of the B trees, and take a majority vote.

Estimate the test error: Out-of-Bag Error

Recall that the key to bagging is that trees are repeatedly fit to a subsets of the observations. On average, each bagged tree only makes use of around two-thirds of the observations.

- The remaining one-third of the observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations.
- We can predict the response for the ith observation using each of the trees in which that observation was OOB (not used), which gives about B/3 predictions.
- We can get a single OOB prediction by average these predicted responses or can take a majority vote.
- The resulting OOB error is a valid estimate of the test error (no need to perform cross-validation!).

Feature Importance

Bagging typically improves accuracy over prediction using a single tree. However, it can be difficult to interpret the resulting model. i.e. it improves prediction accuracy at the expense of interpretability. We seek a way to obtain an overall summary of the importance of each predictor.

- We can record the total amount that the SSE (or Gini index in classification case) is decreased due to splits over a given predictor, averaged over all B trees.
- A large value indicates an important predictor.

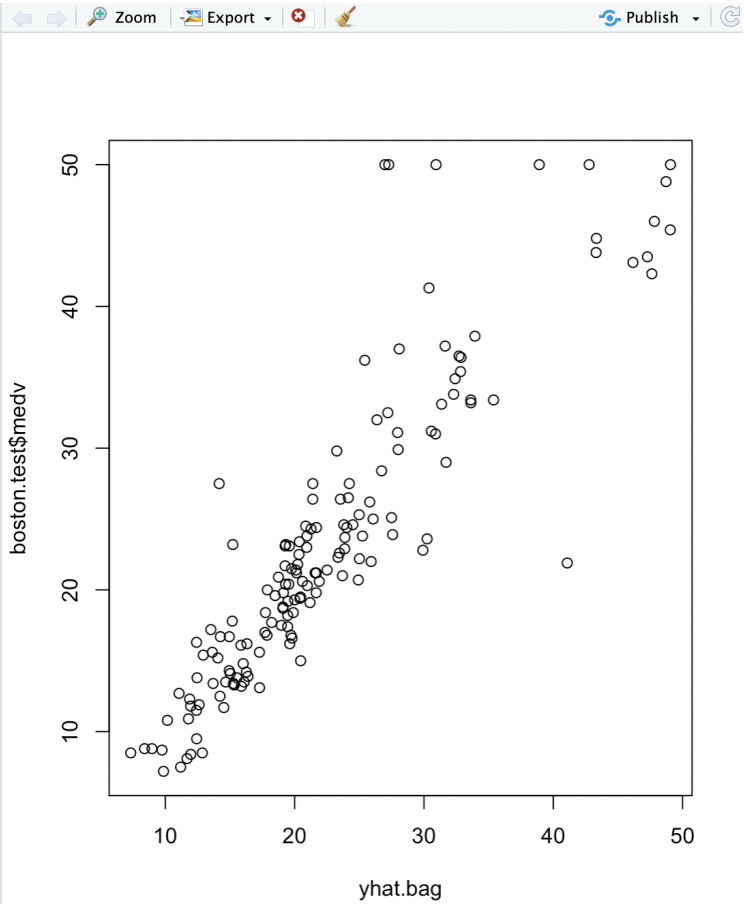
Random Forests

Random forests (RF) provide an improvement over bagged trees by a small tweak.

- At each split in the tree, the RF algorithm only consider a small subset of m predictors (features) out of total p available predictors ($m \approx \sqrt{p}$).
- Random forests eliminate the chance that all of the bagged trees look quite similar to each other (highly correlated).
- We can think of this process as "de-correlating" the trees.
- Random forest will typically be helpful when we have a large number of correlated predictors.

Example: Boston Data

```
R 4.3.0 · ~/
> library(ISLR2)
> library(randomForest)
> set.seed(12345)
> train=sample(1:506, 350)
> boston.test=Boston[-train,]
> bag.boston=randomForest(medv~.,data=Boston,subset=train, mtry=12,importance =TRUE)
> bag.boston

Call:
 randomForest(formula = medv ~ ., data = Boston, mtry = 12, importance = TRUE,
subset = train)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 12

          Mean of squared residuals: 9.809631
                    % Var explained: 86.67
> yhat.bag = predict(bag.boston, newdata=boston.test)
> plot(yhat.bag, boston.test$medv)
> test.MSE=mean((yhat.bag-boston.test$medv)^2)
> test.MSE
[1] 22.34719
> importance(bag.boston)
          %IncMSE IncNodePurity
crim    16.577918    1202.64004
zn       1.299832      22.34421
indus    9.809099     147.88595
chas    -1.226632      18.68005
nox     20.161418     373.09824
rm      64.493715   14222.92614
age     14.851328     289.11590
dis     16.352592    1159.19374
rad      5.508806     102.71307
tax     14.748086     320.36218
ptratio 19.729634     449.44875
lstat   27.872758    7349.81556
>
```



Example: Iris Data

```
> lda.out=lda(default~income+student+balance)
> lda.out
Call:
lda(default ~ income + student + balance)

Prior probabilities of groups:
    No    Yes
0.9667 0.0333

Group means:
      income studentYes   balance
No  33566.17  0.2914037  803.9438
Yes 32089.15  0.3813814 1747.8217

Coefficients of linear discriminants:
                    LD1
income      3.367310e-06
studentYes -1.746631e-01
balance     2.243541e-03
> lda.class=predict(lda.out,Default)$class
> table(lda.class,default)
         default
lda.class   No  Yes
      No  9645  254
      Yes   22   79
```

STAT562 Lecture 12 Boosting

Beidi Qiang

SIUE

Ensemble Methods

The basic idea behind boosting is converting many weak learners to form a single strong learner.

- A weak classifier is one that performs better than random guessing, but still performs poorly at designating classes to objects.
- A model formed using a single predictor/classifier is not powerful individually.

We create an ensemble model by combining several weak leaners.

Type of Ensemble Methods

The modes that combines several base algorithms to form one optimized predictive algorithm is called Ensemble Methods. Ensemble Methods can also be divided into two groups:

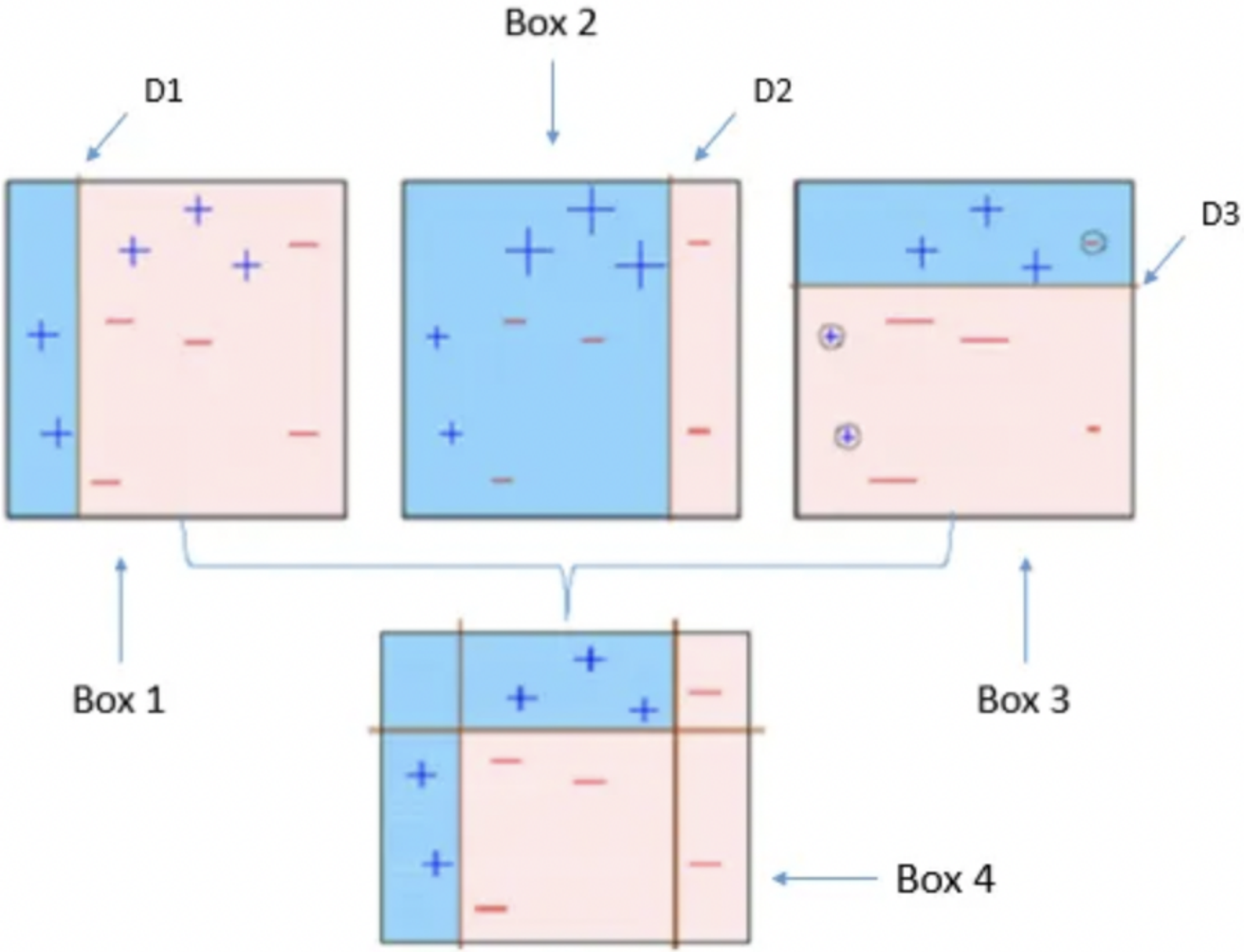
- Sequential Learners, where different models are generated sequentially and the mistakes of previous models are learned by their successors. This aims at exploiting the dependency between models by giving the mislabeled examples higher weights (e.g. AdaBoost).
- Parallel Learners, where base models are generated in parallel. This exploits the independence between models by averaging out the mistakes (e.g. Random Forest).

Boosting

Recall in bagging, each tree is built on a bootstrap data set, independent of the other trees. Boosting works in a similar way, except that the trees are grown sequentially: each tree (Decision Stump) is grown using information from previously trees (stumps).

- The boosting algorithm assigns equal weight to each data sample. It feeds the data to the first model, called the base algorithm. The base algorithm makes predictions for each data sample.
- Now we do the following till maximum number of iteration is reached:
  - Assesses model predictions and increases the weight of samples with a more significant error.
  - We then passes the weighted data to the train next decision tree.
- After the iterations have been completed, we combine weak rules (trees) to form a single strong rule, which will then be used as our model.

Example:



Adaptive Boosting (AdaBoost)

Type of boosting approach described in the previous example is known as Adaptive Boosting or AdaBoost. Weights are calculated as follows:

- x is the set of data features. y is the target variable.
- Initially, $w = 1/n$ for all examples.
- At each step, we calculate:

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

$$\alpha = ln\left(\frac{1 - error}{error}\right)$$

$$\text{update weights} = w_i \cdot exp(\alpha I(\hat{y}_i \neq y_i))/Z$$

The algorithm trains the weak classifier at each step with the current sample weights. Weak models are added sequentially. Once completed, you are left with a pool of weak learners each with a α value. The final prediction is obtained by aggregating the predictions.

Pseudocode

Algorithm 1 *AdaBoost (Freund & Schapire 1997)*

1. *Initialize the observation weights* $w_i = 1/n,\ i = 1, 2, \ldots, n.$
2. *For* $m = 1$ *to* M*:*
   (a) *Fit a classifier* $T^{(m)}(\boldsymbol{x})$ *to the training data using weights* w_i.
   (b) *Compute*
   $$err^{(m)} = \sum_{i=1}^{n} w_i \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right) / \sum_{i=1}^{n} w_i.$$
   (c) *Compute*
   $$\alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}}.$$
   (d) *Set*
   $$w_i \leftarrow w_i \cdot \exp\left(\alpha^{(m)} \cdot \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right)\right),\ i = 1, 2, \ldots, n.$$
   (e) *Re-normalize* w_i.
3. *Output*
   $$C(\boldsymbol{x}) = \arg\max_k \sum_{m=1}^{M} \alpha^{(m)} \cdot \mathbb{I}(T^{(m)}(\boldsymbol{x}) = k).$$

Advantages and Disadvantages of AdaBoost

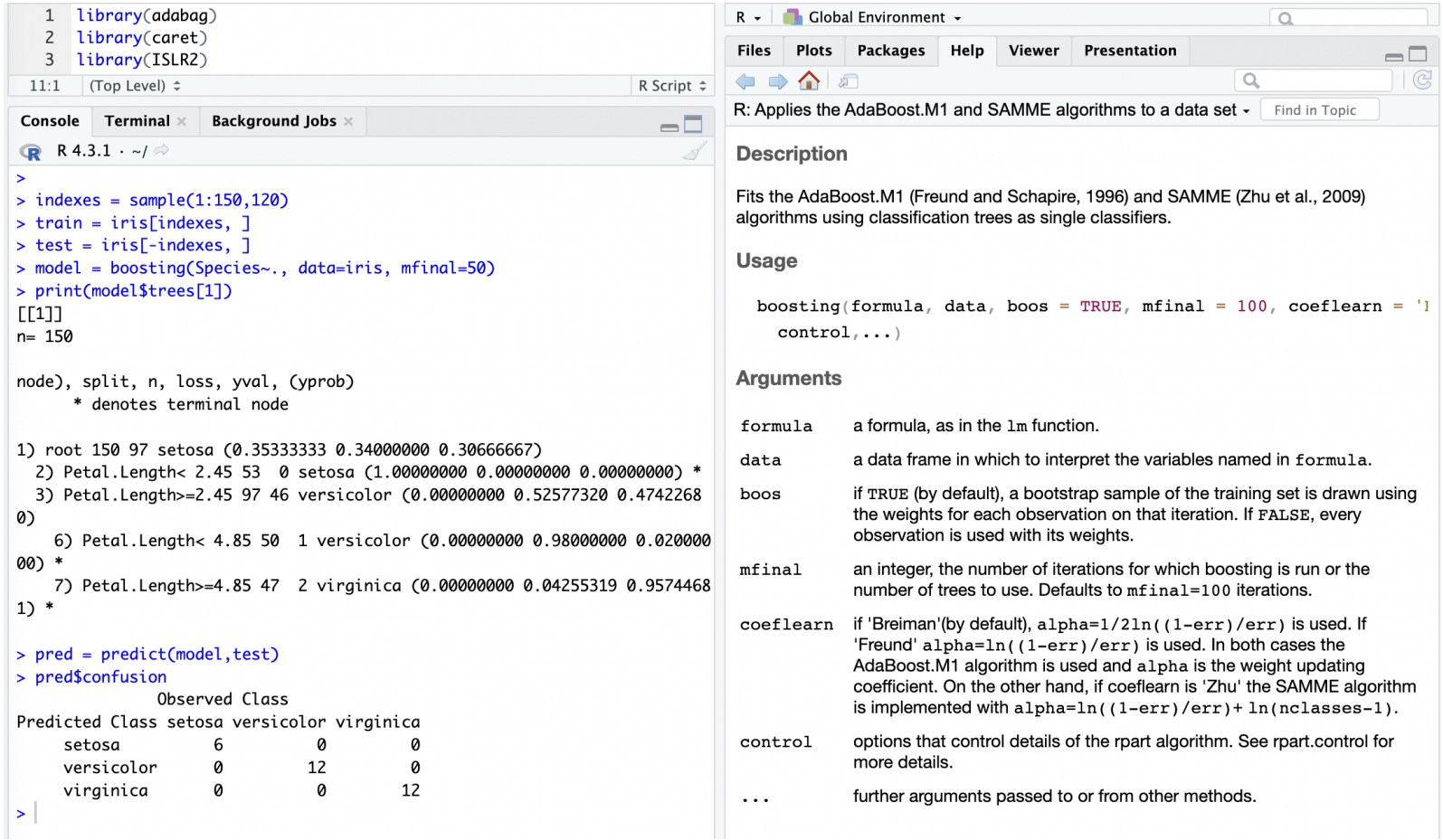
Advantages of AdaBoost:

- Easier to use with less need for tweaking parameters.
- Boosting can be used to improve the accuracy of your weak classifiers hence making it flexible.
- It can be extended beyond binary classification.
- Boosting is an approach that learns slowly, reducing the potentially overfitting issue.

A few Disadvantages of AdaBoost are :

- AdaBoost is also extremely sensitive to Noisy data and outliers.
- AdaBoost has also been proven to be slow.

Example: Iris data



```
1  library(adabag)
2  library(caret)
3  library(ISLR2)
11:1  (Top Level)  R Script

Console  Terminal  Background Jobs
R 4.3.1 · ~/

> indexes = sample(1:150,120)
> train = iris[indexes, ]
> test = iris[-indexes, ]
> model = boosting(Species~., data=iris, mfinal=50)
> print(model$trees[1])
[[1]]
n= 150

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 150 97 setosa (0.35333333 0.34000000 0.30666667)
  2) Petal.Length< 2.45 53  0 setosa (1.00000000 0.00000000 0.00000000) *
  3) Petal.Length>=2.45 97 46 versicolor (0.00000000 0.52577320 0.4742268
0)
    6) Petal.Length< 4.85 50  1 versicolor (0.00000000 0.98000000 0.020000
00) *
    7) Petal.Length>=4.85 47  2 virginica (0.00000000 0.04255319 0.9574468
1) *

> pred = predict(model,test)
> pred$confusion
                Observed Class
Predicted Class setosa versicolor virginica
     setosa          6          0         0
     versicolor      0         12         0
     virginica       0          0        12
>
```

R Global Environment

Files Plots Packages Help Viewer Presentation

R: Applies the AdaBoost.M1 and SAMME algorithms to a data set Find in Topic

Description

Fits the AdaBoost.M1 (Freund and Schapire, 1996) and SAMME (Zhu et al., 2009) algorithms using classification trees as single classifiers.

Usage

```
boosting(formula, data, boos = TRUE, mfinal = 100, coeflearn = 'I
  control,...)
```

Arguments

formula	a formula, as in the lm function.
data	a data frame in which to interpret the variables named in formula.
boos	if TRUE (by default), a bootstrap sample of the training set is drawn using the weights for each observation on that iteration. If FALSE, every observation is used with its weights.
mfinal	an integer, the number of iterations for which boosting is run or the number of trees to use. Defaults to mfinal=100 iterations.
coeflearn	if 'Breiman'(by default), alpha=1/2ln((1-err)/err) is used. If 'Freund' alpha=ln((1-err)/err) is used. In both cases the AdaBoost.M1 algorithm is used and alpha is the weight updating coefficient. On the other hand, if coeflearn is 'Zhu' the SAMME algorithm is implemented with alpha=ln((1-err)/err)+ ln(nclasses-1).
control	options that control details of the rpart algorithm. See rpart.control for more details.
...	further arguments passed to or from other methods.

Boosting

Recall in bagging, each tree is built on a bootstrap data set, independent of the other trees. Boosting works in a similar way, except that the trees are grown sequentially: each tree (Decision Stumps) is grown using information from previously decision Stumps. The algorithm is

- Start with $\hat{f}(x) = 0$ and residuals $r_i = y_i$.
- b=1,2,...,B, Fit a new tree $\hat{f}^b$ with d splits to the data (X, r)
- update $\hat{f}$ by adding a shrunken of the new tree: $\hat{f}(x) + \lambda \hat{f}^b(x)$
- update the residual $r_i \rightarrow r_i - \lambda \hat{f}^b(x_i)$.
- Repeat, and finally output

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

.

Comments

- In boosting we fit a tree using the current residual, rather than the outcome Y.
- Each of these trees can be rather small with d split (Often $d = 1$ works well).
- The shrinkage parameter λ, a small positive number usually 0.01 or 0.001, slows the updating process down more.
- Boosting is a approach that learns slowly, reducing the potentially overfitting issue and usually performs better.
- Boosting is a general approach that can be applied to many statistical learning methods, not restricted to tree.

Example: Boston data

```
library(gbm)
boost.boston=gbm(medv~.,data=Boston[train,],
distribution="gaussian",n.trees=5000, interaction.depth=4)
summary(boost.boston)
yhat.boost = predict(boost.boston,newdata=Boston[-train,])
```

STAT562 Lecture 13 Gradient Boosting

Beidi Qiang

SIUE

Components Gradient Boosting

Gradient boosting is a method that fits complicated models by refitting sub-models typically decision trees to either residuals or pseudo residuals. It involves three elements:

- A loss function to be optimized.
- A weak learner to make predictions.
- An additive model to add weak learners to minimize the loss function.

General Gradient Boosting Algorithm

Initialize: $F_0(x)$
At each iteration m, $m = 1, \cdots M$:

- Optimize:

$$(\beta_m, \boldsymbol{\alpha}_m) = argmin \sum_{i-1}^{n} L(y_i, F_{m-1}(x_i) + \beta h(x_i, \boldsymbol{\alpha}))$$

- Update model:

$$F_m(x) = F_{m-1}(x) + \epsilon \beta_m h(x, \boldsymbol{\alpha}_m)$$

Here,
$L(y, F)$ is a loss function.
$h(x, a)$ is a weak (base) learner with parameters $\boldsymbol{\alpha}$. ϵ is a shrinkage factor that slows the stagewise learning.

The Loss function

- Regression: (squared-error L2 loss)

$$L(y, F) = \frac{1}{n}\sum(y_i - F(x_i))^2$$

- Classification: (logistic loss)

$$L(y, p) = -(y\log(p) + (1-y)\log(1-p)),$$

where $p = Pr(y = 1)$

Gradient Boosting for Regression

The algorithm is

- Start with $\hat{f}(x) = 0$ and residuals $r_i = y_i$.
- b=1,2,...,B, Fit a new tree $\hat{f}^b$ with d splits to the data (X, r)
- update $\hat{f}$ by adding a shrunken of the new tree: $\hat{f}(x) + \epsilon \hat{f}^b(x)$
- update the residual $r_i \rightarrow r_i - \epsilon \hat{f}^b(x_i)$.
- Repeat, and finally output

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

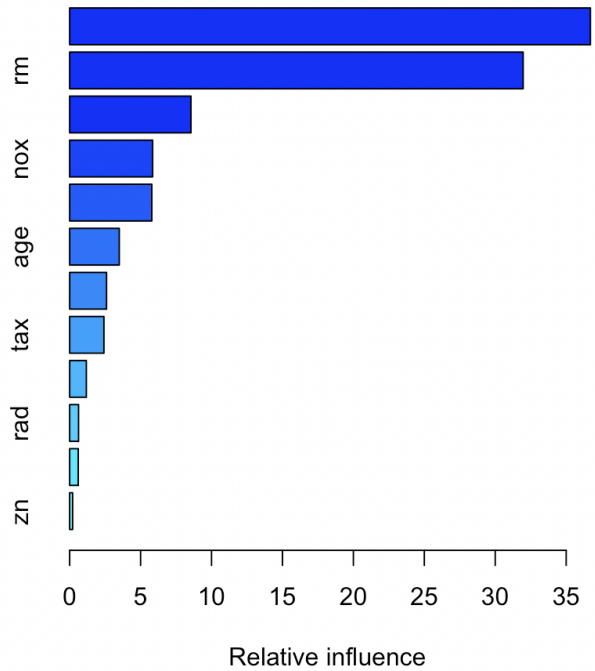
.

Comments

- In boosting we fit a tree using the current residual, rather than the outcome Y.
- Each of these trees can be rather small with d split.
- The shrinkage parameter ϵ, a small positive number usually 0.01 or 0.001, slows the updating process down more.
- Boosting is an approach that learns slowly, reducing the potentially overfitting issue and usually performs better.

Example: Boston data

```
> library(gbm)
Loaded gbm 2.1.8.1
> train = sample(1:506,350)
> boost.boston=gbm(medv~.,data=Boston[train,],
+                  distribution="gaussian",n.trees=5000, interaction.depth=4)
> summary(boost.boston)
            var    rel.inf
lstat     lstat 36.7083465
rm           rm 31.9682625
dis         dis  8.5481945
nox         nox  5.8495002
crim       crim  5.7897636
age         age  3.4983779
ptratio ptratio  2.5997693
tax         tax  2.4168475
chas       chas  1.1764697
rad         rad  0.6287636
indus     indus  0.6061389
zn           zn  0.2095657
> yhat.boost = predict(boost.boston,newdata=Boston[-train,])
Using 5000 trees...

>
```



Stochastic Gradient Boosting

stochastic gradient boosting is a variation of grading boosting that reduces the correlation between the trees in the sequence in gradient boosting models. At each iteration a subsample of the training data is drawn at random (without replacement) from the full training dataset. The randomly selected subsample is then used, instead of the full sample, to fit the base learner.

A few variants of stochastic boosting that can be used:

- Subsample rows before creating each tree.
- Subsample columns before creating each tree
- Subsample columns before considering each split.

STAT562 Lecture 14 Support Vector Machines

Beidi Qiang

SIUE

Introduction

Support vector machine (SVM) is an approach for classification that was developed in the 1990s and that has grown in popularity since then. SVMs have been shown to perform well in a variety of settings, and are often considered one of the best classifiers. We will introduce SVM in the following settings:

- Classes are separable by a linear boundary (hyperplane) in binary setting
- Classes are not separable in binary setting
- Extension of SVM to nonlinear class boundaries in binary setting
- Extensions of SVM to more than two classes.

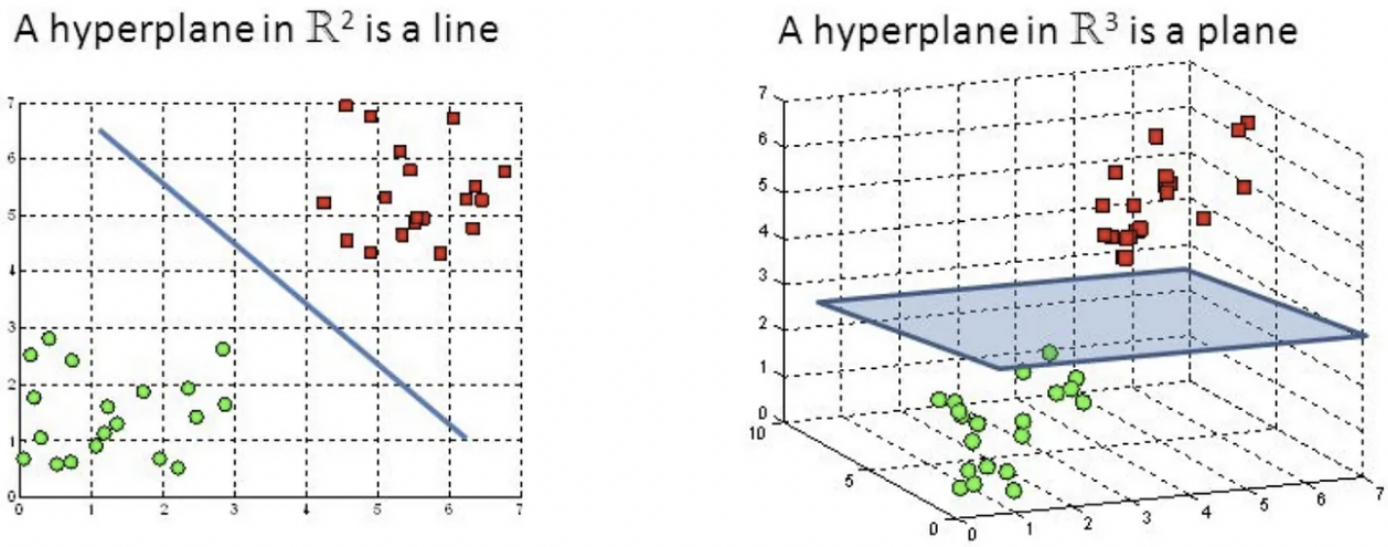
Hyperplane

In a p-dimensional space, hyperplane is a flat (p-1)-dimensional subspace , defined as

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

- In two dimensions, a hyperplane is a line. In three dimensions, a hyperplane is a plane.
- If $X = (X_1 \cdots X_p)^T$ satisfy $\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p = 0$, we say X lies on the hyperplane.
- $\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p > 0$, or $\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p < 0$, tells us that X lies to one or the other side of the hyperplane.

2D and 3D Hyperplane



Hyperplanes in 2D and 3D feature space

Classification Using a Separating Hyperplane

Our goal is to develop a classifier based on the training data that will correctly classify the test observation using the predictors.

- Assuming in binary case, the observations fall into two classes, labeled as $y = -1$ and $y = 1$.
- Suppose that it is possible to construct a hyperplane that separates the training observations perfectly according to their class labels, i.e.,
  $\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} < 0$, for all i when $y_i = -1$
  $\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} > 0$, for all i when $y_i = 1$
- Equivalently, $y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) > 0$ for all observation in the training.
- A test observation is assigned a class depending on which side of the hyperplane it is located.

Separating Hyperplane

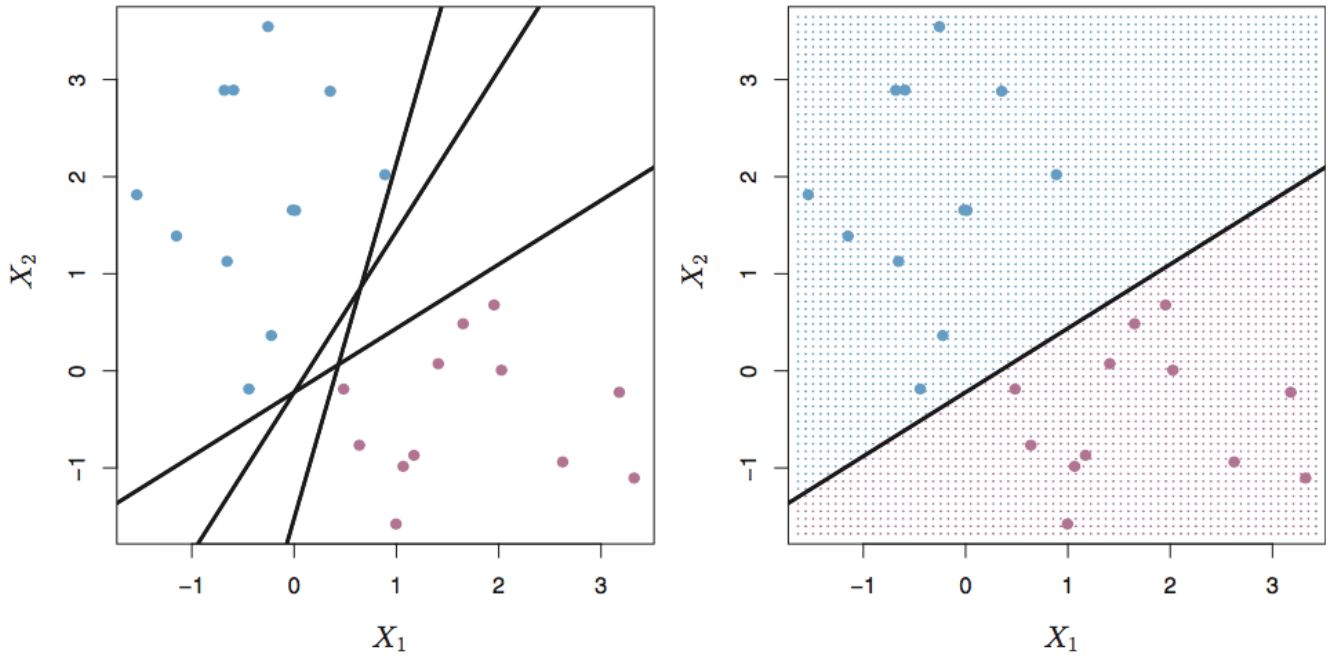


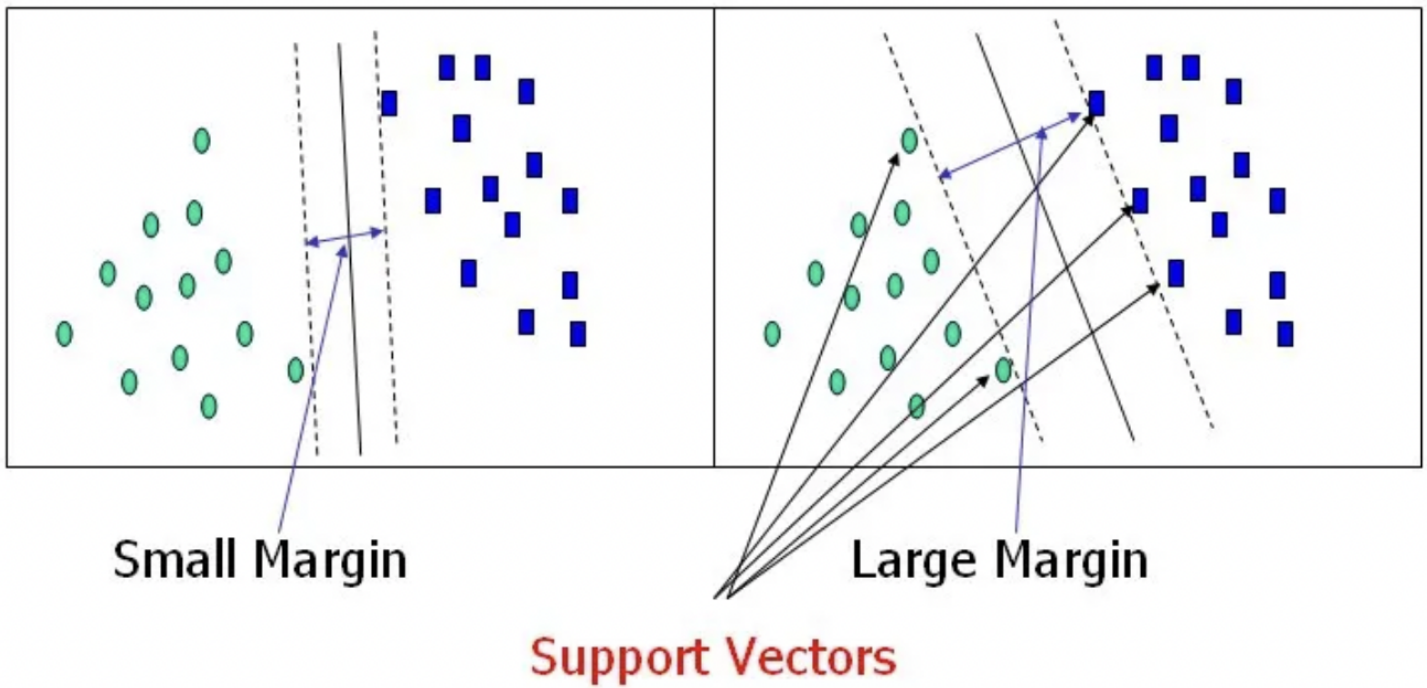
FIGURE 9.2. Left: *There are two classes of observations, shown in blue and in purple, each of which has measurements on two variables. Three separating hyperplanes, out of many possible, are shown in black.* Right: *A separating hyperplane is shown in black. The blue and purple grid indicates the decision rule made by a classifier based on this separating hyperplane: a test observation that falls in the blue portion of the grid will be assigned to the blue class, and a test observation that falls into the purple portion of the grid will be assigned to the purple class.*

Maximal Margin Classifier

If the training data can be perfectly separated using a hyperplane, then there will in fact exist an infinite number of such hyperplanes. We must have a reasonable way to decide on one. A natural choice is the "maximal margin hyperplane".

- We can compute the distance from each training observation to a hyperplane. The smallest such distance is known as the margin.
- The maximal margin hyperplane is the separating hyperplane for which the margin is largest
- The maximal margin hyperplane is the separating hyperplane that is farthest from the training observations.

Maximal Margin



Support Vectors

- Training observations that are on the margin are called "support vectors". They support the maximal margin hyperplane.
- If the support vectors moved slightly then the maximal margin hyperplane would move as well.
- A small movement to any of the other observations would not affect the separating hyperplane.
- the maximal margin hyperplane depends directly on only a small subset of the observations, i.e. the support vectors.

Construction of the Maximal Margin Classifier

The maximal margin hyperplane is the solution to the optimization problem:

- Chooses $\beta_0, \cdots, \beta_p$ to maximize M, subject to

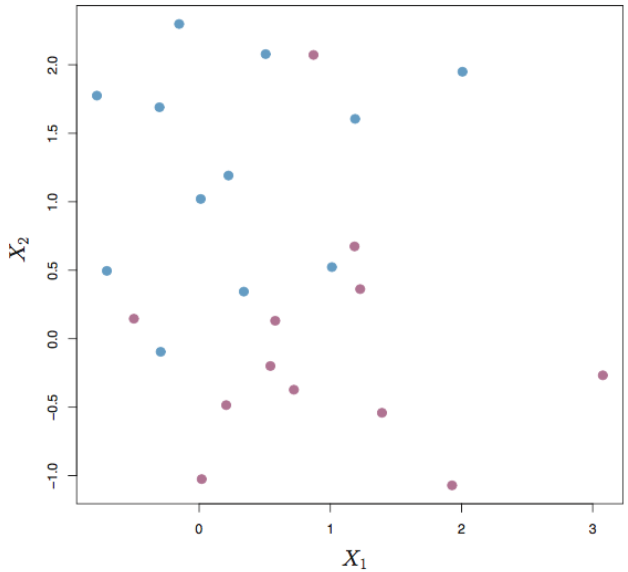
$$\sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq M \text{ for all i.}$$

- Every observation need to be not only on the correct side of the hyperplane but also on the correct side of the margin.
- With the constraints, one can show the perpendicular distance from the ith observation to the hyperplane is given by $y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip})$.
- The optimization problem is usually done numerically.

Non-separable Case

In many cases no separating hyperplane exists.



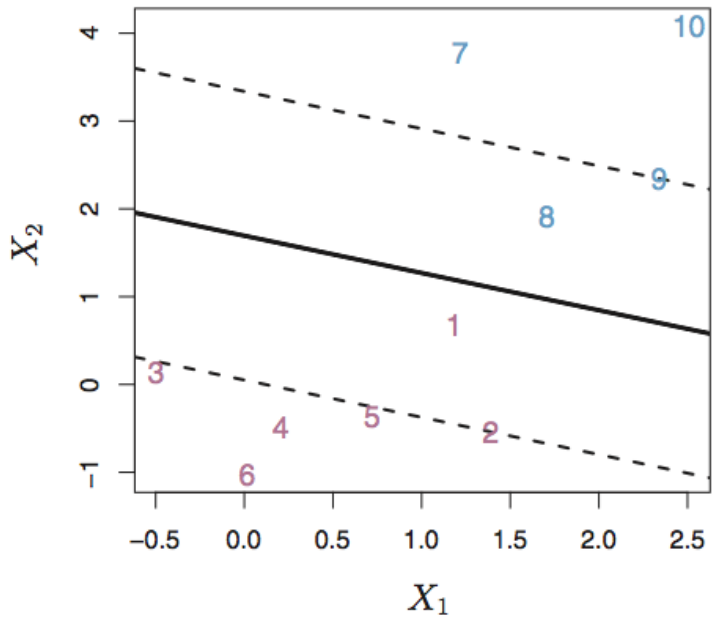
- In this case, the optimization problem on the previous slide has no solution with $M > 0$.
- We will extend the concept of a separating hyperplane in order to develop a hyperplane that almost separates the classes, using a so-called "soft margin".

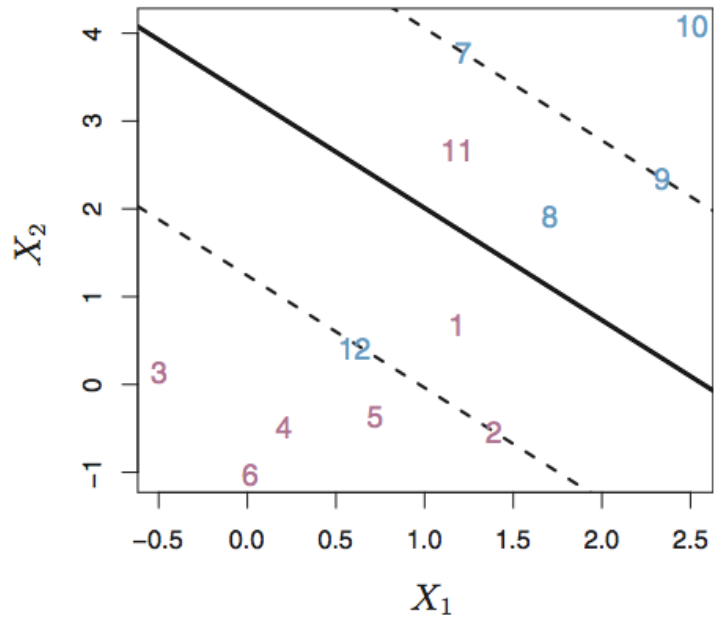
Soft margin classifier

We might need to consider a classifier based on a hyperplane that does not perfectly separate the two classes.

- In the case of non-separable
- Or it could be worthwhile to misclassify a few training observations in order to do a better job in classifying the remaining observations.
- We allow some observations to be on the incorrect side of the margin, or even the incorrect side of the hyperplane.
- The margin is "soft" because it can be violated by some of the training observations.

Soft Margin


10
7
9
8
1
3
4
5
2
6
4
3
2
1
0
−1
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
X_1
X_2


10
7
11
9
8
1
12
3
4
5
2
6
4
3
2
1
0
−1
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
X_1
X_2

Construction of soft margin SVM

The soft margin support vector classifier is the solution to the optimization problem:

- Chooses $\beta_0, \cdots, \beta_p$ to maximize M, subject to

$$y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq M(1 - \epsilon_i) \text{ for } \epsilon_i \geq 0$$

$$\sum_{j=1}^{p} \beta_j^2 = 1, \text{and } \sum_{i=1}^{n} \epsilon_i \leq C$$

- C is a nonnegative tuning parameter.
- ϵ_i are slack variables that allow individual observations to be on the wrong side.
- We classify a test observation as before, by simply determining on which side of the hyperplane it lies.

The tuning parameter C

- C determines the number and severity of the violations to the margin that we will tolerate.
- Note if $\epsilon_i = 0$, the observation is on the correct side of the margin. If $\epsilon_i > 0$, the observation violated the margin. Further, If $\epsilon_i > 1$, the observation is on the wrong side of the hyperplane(misclassified).
- If $C = 0$ then there is no budget for violations,
- For $C > 0$, there can be no more than C observations misclassified.
- In general, a larger C will allow for a wider margin.
- C is generally chosen via cross-validation.

Remarks

- Observations that lie directly on the margin, or on the wrong side of the margin for their class, are known as support vectors.
- Only observations that either lie on the margin or that violate the margin (the support vectors) will affect the hyperplane.
- When the tuning parameter C is large, then we allow many observations violate the margin (more support vectors). In this case, many observations are involved in determining the hyperplane. We have low variance but high bias.
- If C is small, then there will be fewer support vectors and hence the resulting classifier will have low bias but high variance.

Example: Simulated Data

We will use e1071 package in R to train SVM models. (See attached R code) .

The parameterization used in e1071 is slightly different from the one introduced in the text book. The optimization in e1071 goes like

- Chooses $\beta_0, \cdots, \beta_p$ to minimize

$$1/M + C\sum_{i=1}^{n} \epsilon_i$$

$$\text{subject to } \frac{1}{M} y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq (1 - \epsilon_i)$$

$$\sum_{j=1}^{p} \beta_j^2 = 1$$

- C is called the "cost", which controls the trade-off between the slack variable penalty and the margin.

STAT562 Lecture 15 Kernel SVM

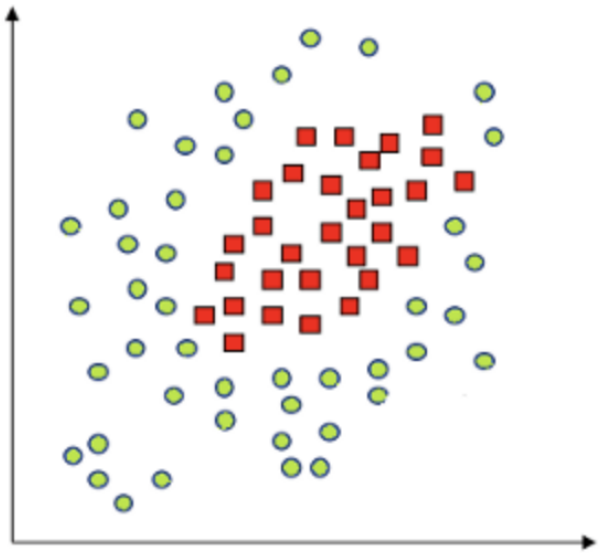
Beidi Qiang

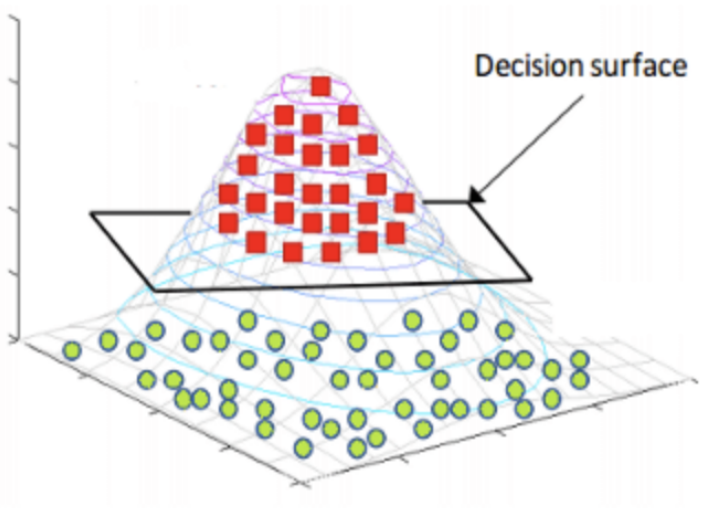
SIUE

Kernel SVM

Kernel SVMs are a class of Support Vector Machines (SVMs) that use kernel functions to classify data.
It is a generalization technique of the linear SVM on nonlinear data.
Kernel SVM offers more flexibility when dealing with a linearly inseparable classification task.

Example: Non-linear Boundaries and Kernel Transformation





The Kernel Trick

Let's consider a simple example. Suppose we have a dataset dimensional features $x_1 = (1, 2)$ and $x_2 = (3, 4)$.

- we want to map our vectors to higher-dimensional (say a 4-d) space by the 2nd order terms, i.e.

$$\phi(x_1) = (1, 2, 2, 4), \phi(x_2) = (9, 12, 12, 16)$$

- The distance between the two new vectors in high dimensions can be calculated by taking the inner product.

$$< \phi(x_1), \phi(x_2) >= \phi(x_1) \cdot \phi(x_2) = 1 \times 9 + 2 \times 12 + 2 \times 12 + 4 \times 16 = 121$$

- The kernel Trick: you get the same answer by

$$(< x_1, x_2 >)^2 = (1 \times 3 + 2 \times 4)^2 = 11^2 = 121$$

Kernel

The support vector machine enlarges the feature space using "kernel trick", in order to accommodate a non-linear boundary.

- The inner product of two vectors is defined as

$$< \mathbf{x_1}, \mathbf{x_2} >= \sum_{j=1}^{p} x_{1j} x_{2j}.$$

- The kernel is a generalization of the inner product, denoted by $K(\mathbf{x_1}, \mathbf{x_2})$.
- For example, a d-degree polynomial kernel has the form

$$K(\mathbf{x_1}, \mathbf{x_2}) = (1 + \sum_{j=1}^{p} x_{1j} x_{2j})^d.$$

Some More insights on using kernels with SVM

Recall in SVM we have the following optimization problem: Chooses $\beta_0, \cdots, \beta_p$ to maximize M, subject to

$$\sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq M \text{ for all i.}$$

- The solution to the SVM optimization problem can be simplified to involves only the inner products of the observations.
- The classifier (hyperplane) can be expressed in terms of the inner product between the new observation and each of the training points, as $f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i < \mathbf{x}, \mathbf{x_i} >$.
- It turns out if a training observation is not a support vector, then its α_i equals zero, i.e. $f(x) = \beta_0 + \sum_{i \in S} \alpha_i < \mathbf{x}, \mathbf{x_i} >$, where S is the collection of indices of the support vectors.

Popular Kernels

- d-degree polynomial kernel :

$$K(\mathbf{x_i}, \mathbf{x_{i'}}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d.$$

- radial basis function (RBF) kernel:

$$K(\mathbf{x_i}, \mathbf{x_{i'}}) = exp[-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2].$$

- Gaussian kernel (Gaussian RBF):

$$K(\mathbf{x_i}, \mathbf{x_{i'}}) = exp[-\frac{1}{2\sigma^2} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2].$$

Example: polynomial and radial kernel

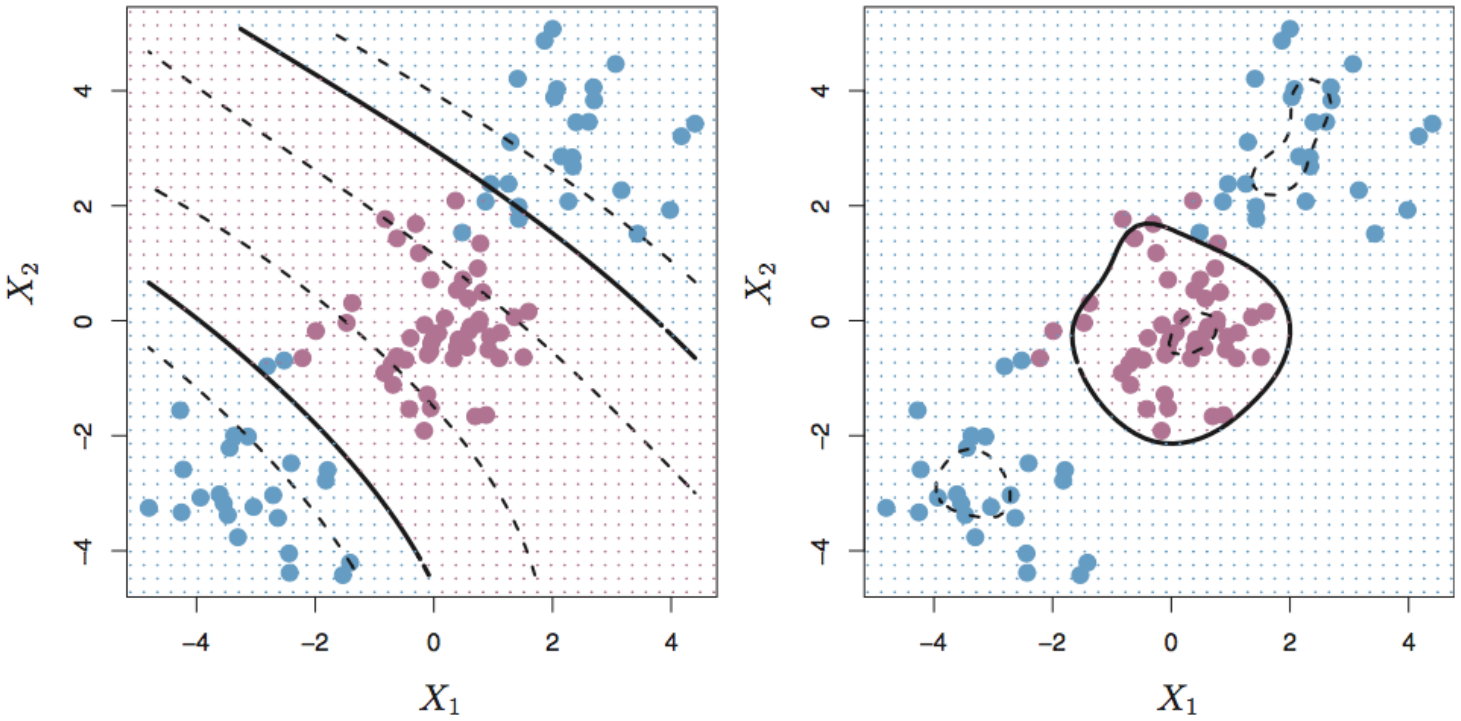


FIGURE 9.9. Left: *An SVM with a polynomial kernel of degree 3 is applied to the non-linear data from Figure 9.8, resulting in a far more appropriate decision rule.* Right: *An SVM with a radial kernel is applied. In this example, either kernel is capable of capturing the decision boundary.*

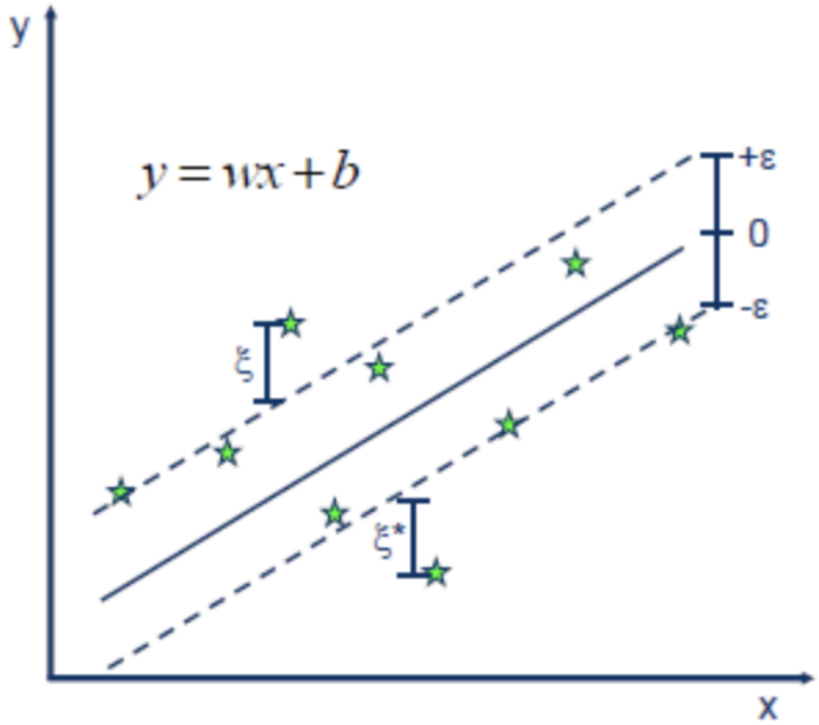
Extending to Multi-classes

We extend SVMs to the more general case where we have some arbitrary number of classes. Suppose we have K classes. There are two popular approaches:

- **One-Versus-One Classification:** Construct $\binom{K}{2}$ SVMs, each compares a pair of classes.The final classification is performed by assigning the test observation to the class to which it was most frequently assigned.
- **One-Versus-All Classification:** We fit K SVMs, each time comparing one of all the K classes to the remaining $K-1$ classes. Let $\beta_{0k}, \cdots, \beta_{pk}$ denote the parameters that result from fitting an SVM comparing the kth class to the others. We assign an observation to the class for which $\beta_{0k} + \beta_{1k}x_1 + \cdots + \beta_{pk}x_p$ is largest.

Support Vector Regression (FYI)

Support Vector Regression uses the same principle as the SVMs. The basic idea behind SVR is to find the best fit hyperplane, that has the maximum number of points within its margin (ϵ).



- Minimize:

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\left(\xi_i + \xi_i^*\right)$$

- Constraints:

$$y_i - wx_i - b \le \varepsilon + \xi_i$$

$$wx_i + b - y_i \le \varepsilon + \xi_i^*$$

$$\xi_i, \xi_i^* \ge 0$$

STAT562 Lecture 17 Neural Network and Deep Learning

Beidi Qiang

SIUE

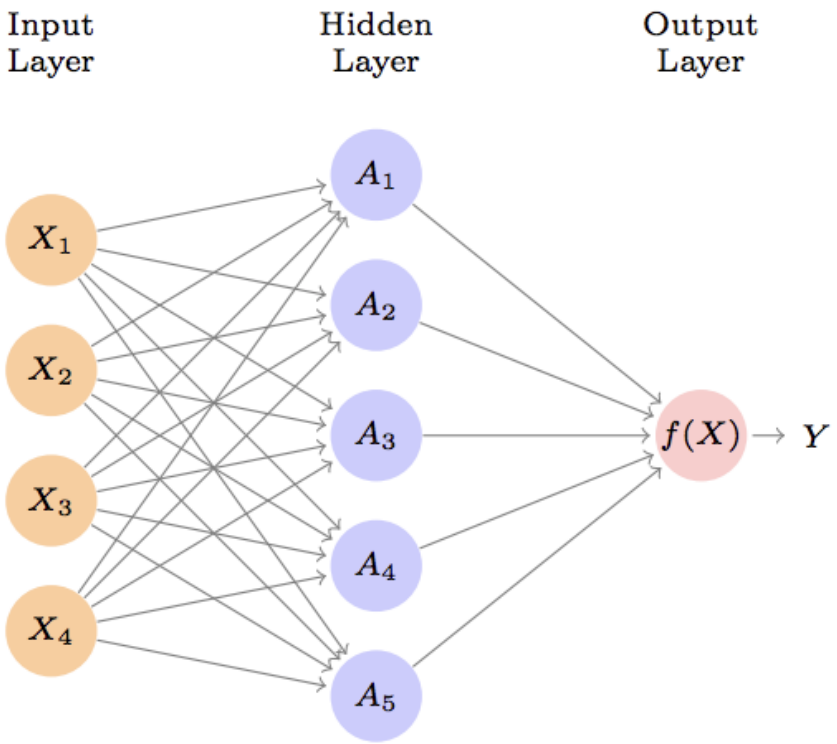
Introduction

Deep learning is a very active area of research in the machine learning and artificial intelligence. The cornerstone of deep learning is the neural network.

- Neural networks rose to fame in the late 1980s.
- Benefit from ever-larger training datasets, Neural networks start to gain popularity after 2010 with the new name deep learning and new architectures
- It is now considered one of the best model in problems such as image and video classification, speech and text modeling.

Single Layer Neural Networks

A neural network takes an input feature vector of $X = (X_1, X_2, ..., X_p)$ and builds a nonlinear function (structure) to predict the response Y. It has a unique structure:



Terminology of neural networks

- $X = (X_1, X_2, ..., X_p)$ forms the input layer.
- The arrows indicate that each of the inputs from the input layer feeds into each of the K hidden units.
- The K activations A_k in hidden layer are computed as functions (transformations) of the input features, i.e. $A_k = h_k(X)$.
- The output layer is a linear model (with quantitative labels) that uses these activations A_k as inputs, resulting in a function $f(X) = \beta_0 + \sum \beta_k A_k$.

Activation Function

At the hidden layer, A_k are computed as functions of X in the form

$$A_k = h_k(X) = g(\omega_{k0} + \omega_{k1}X_1 + \cdots + \omega_{kp}X_p)$$

- $g(\cdot)$ is a **nonlinear** "activation function" that is specified in advance.
- The nonlinearity in the activation function is essential, since without it the model would collapse into a simple linear model!
- All the parameters $\omega_{10}, \cdots, \omega_{Kp}$ in the activation function need to be estimated from training data.
- The $\beta_0 \cdots \beta_K$ in the output function also need to be estimated from training.

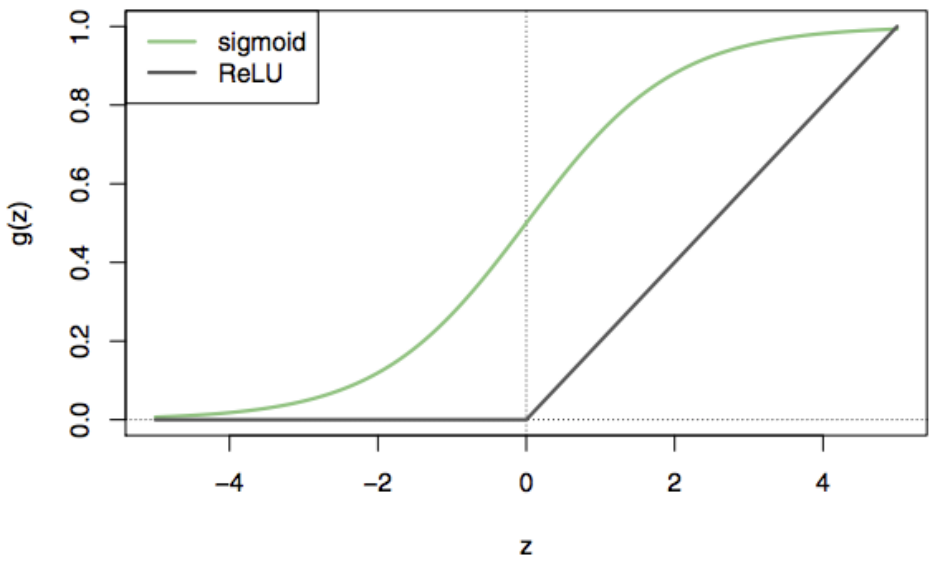
Common Choice of Activation Function

- Sigmoid:

$$g(z) = \frac{e^z}{1 + e^z}$$

- ReLU (rectified linear unit) [more preferred these days]:

$$g(z) = z * I(z > 0)$$



FYI...

The name neural network originally derived from thinking of these hidden units as analogous to neurons in the brain. Values of the activations A_k close to one are "firing", while those close to zero are "silent".

Estimating Coefficients

- Squared-error loss is used. The parameters are chosen to minimize

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

where

$$f(x_i) = \beta_0 + \sum_{k=1}^{K}\beta_k g(\omega_{k0} + \sum_{j=1}^{p}\omega_{kj}x_{ij})$$

- The optimization problem is not simple given the nested parameters. Usually it is done using gradient descent and back-propagation. (Textbook section 10.7)

Weight Decay

Weight decay is a regularization technique by adding a small penalty, usually the L2 norm of the coefficients (weights), to the loss function.

- The parameters are chosen to minimize

$$\frac{1}{2}\sum_{i=1}^{n}[y_i - f(x_i)]^2 + \lambda\omega^T\omega$$

  where ω is the vector of all weights (coefficients) in the model.
- The regularization is added to prevent overfitting and to keep the weights small to avoid exploding gradient.

Multilayer Neural Networks

Modern neural networks typically have more than one hidden layer, and often many units per layer.

- $X = (X_1, X_2, \cdots, X_p)$ is the input layer.
- The activations $A_1^{(1)}, \cdots, A_{K_1}^{(1)}$ in first hidden layer is are computed base of the input layer, i.e.

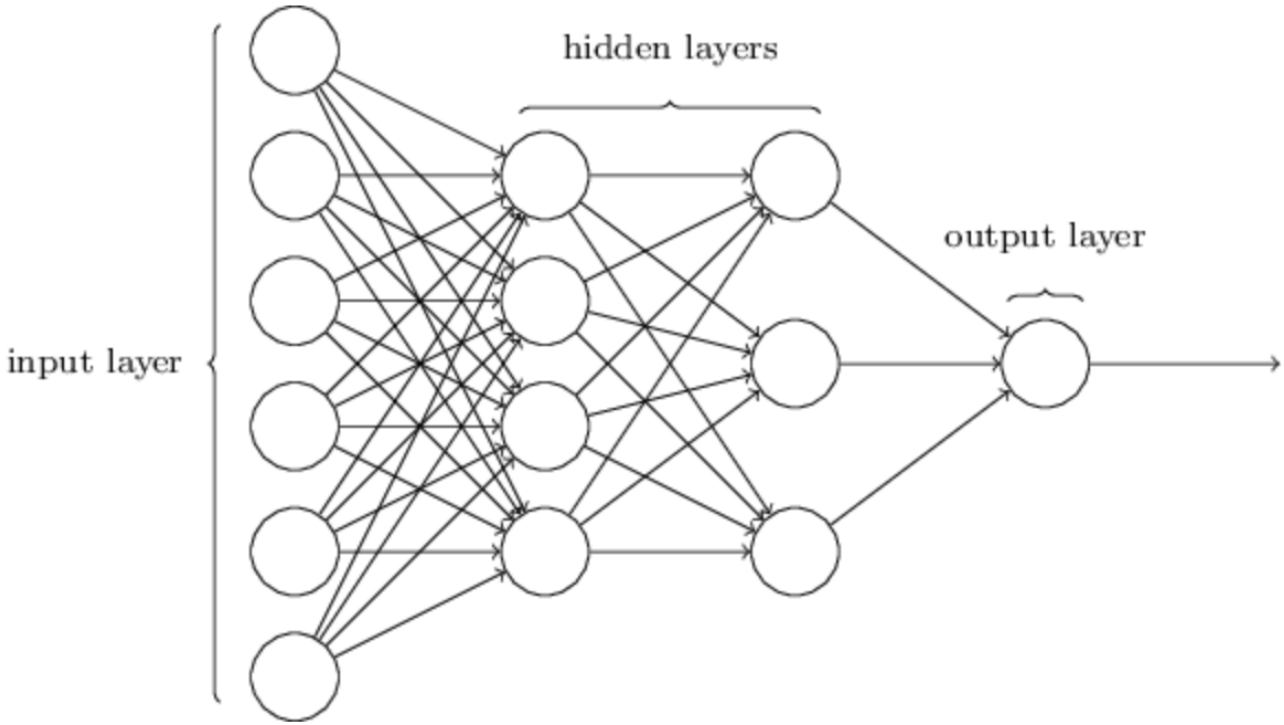
$$A_k^{(1)} = g(\omega_{k0}^{(1)} + \omega_{k1}^{(1)} X_1 + \cdots + \omega_{kp}^{(1)} X_p).$$

- The second hidden layer treats the activations $A_k^{(1)}$ of the first hidden layer as inputs and computes new activations

$$A_l^{(2)} = g(\omega_{l0}^{(2)} + \omega_{l1}^{(2)} A_1^{(1)} + \cdots + \omega_{lK_1}^{(2)} A_{K_1}^{(1)}).$$

- And so on...

2-Layer Neural Networks



Comments

- Each of the activations in the each of the hidden layers is a function of the input vector X.
- This is the case because they are function of previous layers and these go all way back to the input layer of X.
- Through a chain of transformations, the network is able to build up fairly complex transformations of X that ultimately feed into the output layer.

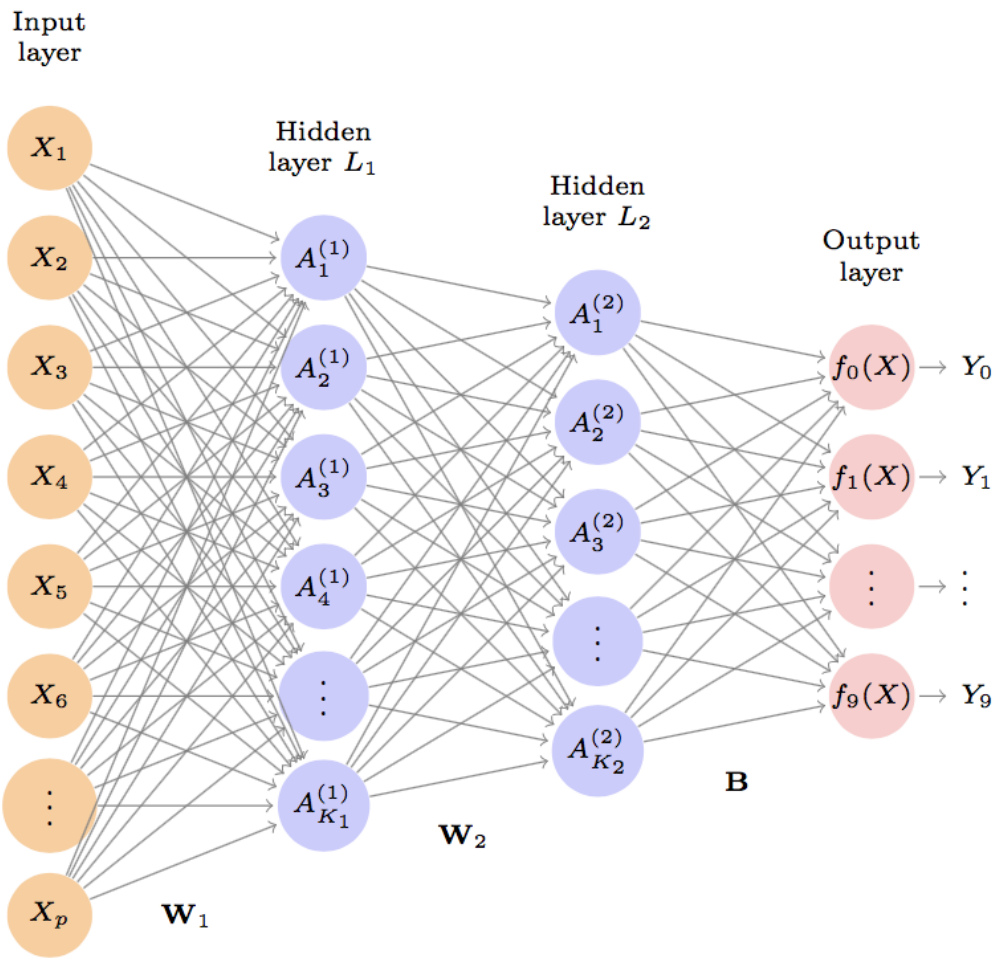
Activation function on the Output layer for Categorical Response

- When we have quantitative response, we simply compute $f(X) = \beta_0 + \sum \beta_l A_l^{(L)}$ at the output layer, where $A_l^{(L)}$ are the activations at the last hidden layer.
- In classification problems we would like our estimates to represent class probabilities. we use the special softmax activation function: Calculate $Z_m = \beta_0 + \sum \beta_l A_l^{(L)}$, for each class, $m = 0, 1, 2, \cdots, C$

$$f_m(X) = Pr(Y = m|X) = \frac{e^{Z_m}}{\sum_{i=0}^{C} e^{Z_i}}$$

- The classifier then assigns the class with the highest probability.

2-Layer Neural Networks in Classification



Example: Single Layer Nnet

Console | Terminal | Background Jobs

R 4.3.1 · ~/

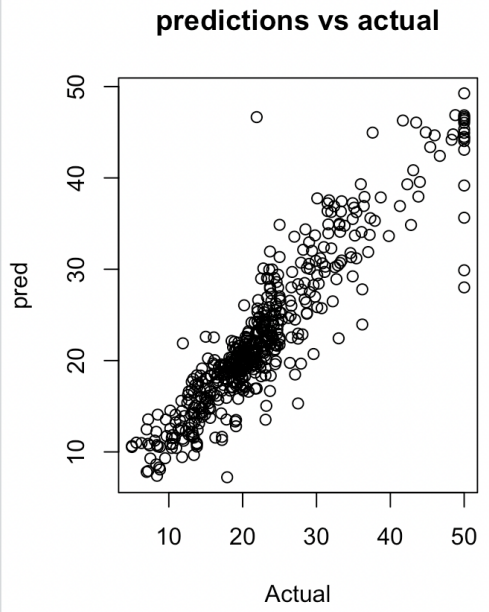
```
> library (nnet)
> nnet.fit=nnet(medv/50~ ., data=Boston, size=3,decay=0.1,linout=T)
# weights:  46
initial  value 17.069815
iter  10 value 15.559714
iter  20 value 12.277624
iter  30 value 9.202323
iter  40 value 8.112481
iter  50 value 5.864114
iter  60 value 3.451666
iter  70 value 3.125704
iter  80 value 3.026654
iter  90 value 2.974369
iter 100 value 2.933375
final  value 2.933375
stopped after 100 iterations
> pred=predict(nnet.fit)*50
> mse=mean((pred-Boston$medv)^2)
> plot(Boston$medv, pred, main="predictions vs actual", xlab="Actual")
> #Tunning
> library(caret)
> mygrid=expand.grid(.decay=c(0.1),.size=c(3,4,5,6,7,8))
> nnetfit=train(medv/50~., data=Boston, method="nnet", maxit=1000, tuneGrid=mygrid, trace=F)
> print(nnetfit)
Neural Network

506 samples
 13 predictor

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 506, 506, 506, 506, 506, 506, ...
Resampling results across tuning parameters:

  size  RMSE        Rsquared   MAE
  3     0.08581603  0.7894123  0.06002583
  4     0.08445377  0.7947649  0.05842578
  5     0.08173036  0.8072009  0.05701671
  6     0.08189119  0.8071613  0.05685111
  7     0.08193556  0.8072438  0.05767213
  8     0.07942243  0.8189901  0.05604136
```

New Connection

Connection	Status

Files | Plots | Packages | Help | Viewer | Presen

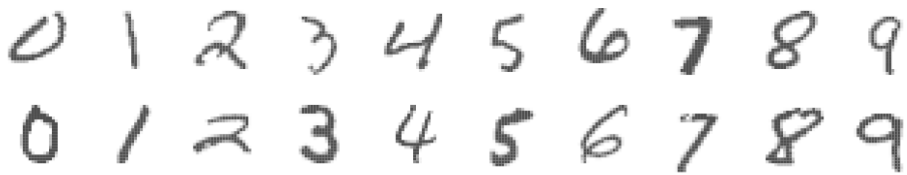
Zoom | Export

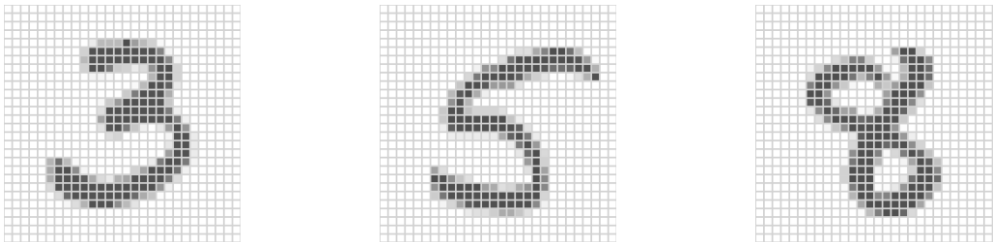


Example: MINST Data

We will illustrate a large feed-forward neural network on the famous MNIST handwritten digit dataset (available at http://yann.lecun.com/exdb/mnist).

- The goal is to build a model to classify the images into their correct digit class 0 to 9.
- Every image has $28X28 = 784$ pixels, each of which is an eight-bit grayscale value between 0 and 255 representing the relative amount of the written digit in that tiny square.





MINST Data Cont.

- There are 60,000 images in the training data and 10,000 in the test data.
- The first layer goes from 784 input units to a hidden layer of 256 units, which uses the ReLU activation function.
- The second hidden layer comes with 128 hidden units, also uses the ReLU activation function.
- The first hidden layer involves $(784 + 1) * 256 = 200960$ parameters and the second hidden layer involves $(256 + 1) * 128 = 32896$ parameters.
- The feed-forward ANN can be trained with Keras packages. The package is based on the TensorFlow backend engine and will require a python environment.
  see: https://www.python.org/downloads/
- Also worth noting are other Deep Learning packages in R, such as H2O, mxnet, deepnet and darch.

For implementation: see the Keras sample code here:
https://cran.r-project.org/web/packages/keras/vignettes/

STAT562 Lecture 18 Convolutional Neural Network

Beidi Qiang

SIUE

Convolutional Neural Networks

Convolutional neural network (CNN) is an advanced architectures of artificial neural network (ANN) and it is specifically used in image classification.

- Similar to ordinary Neural Networks: they are made up of hidden layers and units that have learnable weights.
- Convolutional neural network has two specialized types of hidden layers, called convolution layers and pooling layers.
- Convolution layers search for instances of small patterns in the image
- Pooling layers downsample these instances of patterns to select a prominent subset.

Convolution layer

A convolution layer is made up of a large number of convolution filters (activation units).

- A filter is a l_1 x l_2 array of weights that is operated to the original image data.
- The operation is called convolution, which basically repeatedly multiplying matrix elements to every l_1 x l_2 submatrix of the original image and then adding up.

$$\text{Original Image} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{bmatrix}.$$

Now consider a 2×2 filter of the form

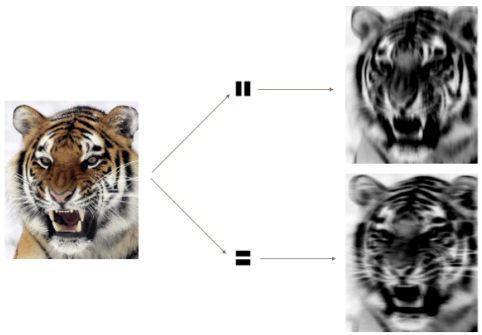
$$\text{Convolution Filter} = \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}.$$

When we *convolve* the image with the filter, we get the result[8]

$$\text{Convolved Image} = \begin{bmatrix} a\alpha + b\beta + d\gamma + e\delta & b\alpha + c\beta + e\gamma + f\delta \\ d\alpha + e\beta + g\gamma + h\delta & e\alpha + f\beta + h\gamma + i\delta \\ g\alpha + h\beta + j\gamma + k\delta & h\alpha + i\beta + k\gamma + l\delta \end{bmatrix}.$$

Convolution layer, Cont.

- If a sub-matrix of the original image resembles the convolution filter, then it will have a large value in the convolved image; otherwise, it will have a small value.
- Thus, the convolved image highlights regions of the original image that resemble the convolution filter.
- We can think of the original image as the input layer in a convolutional neural network, and the convolved images as the units in the first hidden layer.



More details on colored Image Data

- Format of input data: W x H colored (RGB) pixels per image. Each color channel (red, green, and blue), is a 2-d (W x H) array (called feature map). The dimension of input image is W x H x 3.
- A single convolution filter will also have three channels, one per color, with potentially different filter weights.
- The results of the three convolutions are summed to form a 2-d output feature map (color is not passed on to subsequent layers).
- The dimension of output activations in the 1st convolution layer is W_1 x H_1 x K, where K is the number of filters.

Depth, stride and zero-padding

Three hyperparameters control the size of the output in each convolution layer: depth, stride and zero-padding.

- The depth (K) of the output volume is the number of filters we would like to use.
- The stride (usually $S = 1$ or 2) defined how we slide the filter. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 then the filters jump 2 pixels at a time.
- Zero-padding (P) is the amount of 0 added to the outside of the feature map, which allows us to control or preserve the spatial size of the input volume.

Detector layer

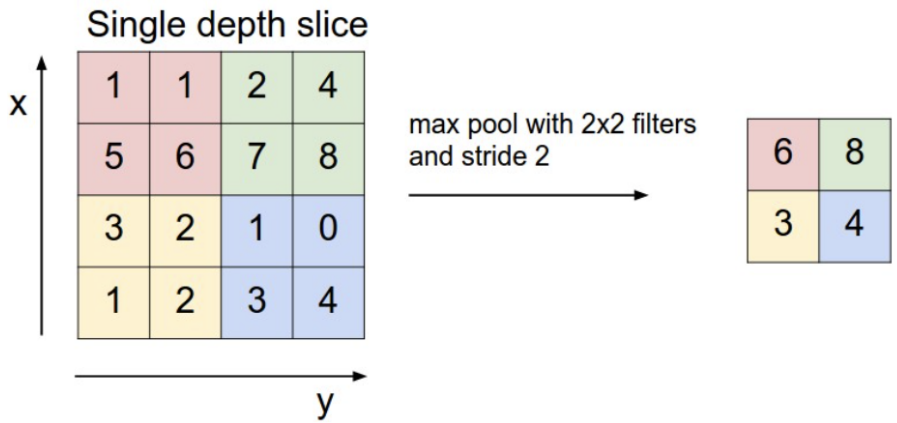
We apply the ReLU activation function to the convolved image from the convolution layer.

- This step is often viewed as a separate layer in CNN, called detector layer or ReLU layer.
- This layer simply threshold at zero and dimension of the information is unchanged.

Pooling Layers

A pooling layer condense a large image into a smaller summary image. The layer reduce the amount of parameters and computation in the network, and hence to also control overfitting.

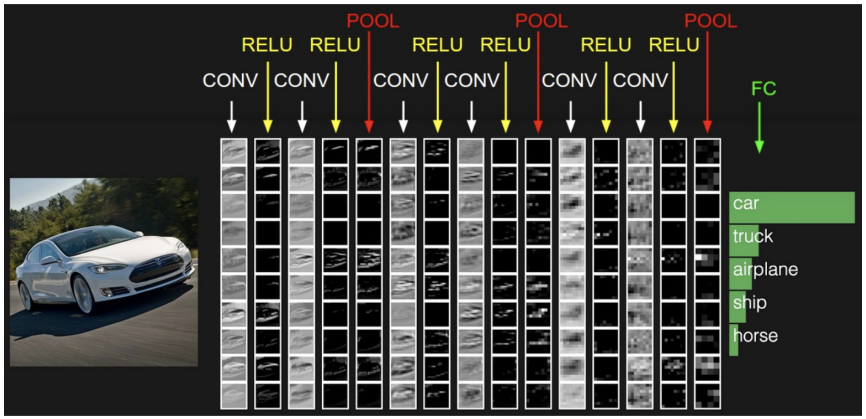
- The Pooling layer operates independently on every slice of the input, using the MAX operation.
- The most common form is a max pooling of size 2 x 2, taking a max over 4 numbers (little 2 x 2 region).
- This reduces the size of the image by a factor of two in each direction, discarding 75%.



Architecture of a Convolutional Neural Network

In a convolutional Neural Network, the convolve-then-pool sequence is often repeated several times

- Each subsequent convolutional layer is similar to the first, with number of channels same as the number of filters from previous convolution layer.
- Since the feature maps are reduced in size after each pool layer, we usually increase the number of filters in the next convolve layer to compensate.
- Sometimes we repeat several convolve layers before a pool layer.



Fully-connected layer

After a set of convolve-then-pool, each channel feature map down to just a few pixels in each dimension.

- We "flattened" the feature maps by treating each of the pixels as separate units.
- We then feed those into a fully connected layer just as one in a regular ANN.
- The output is fed into the output layer, which is has softmax activation for image classes.

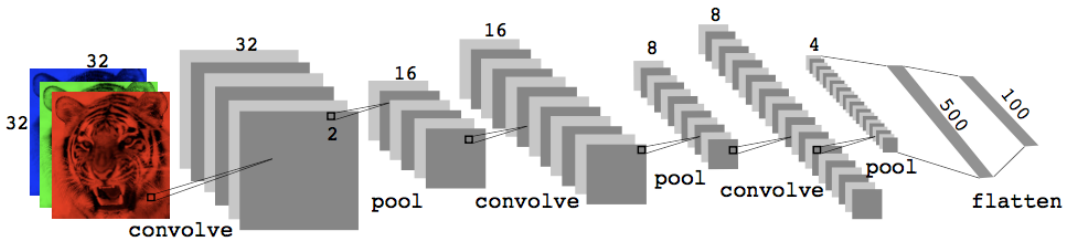


FIGURE 10.8. *Architecture of a deep CNN for the* CIFAR100 *classification task. Convolution layers are interspersed with* 2×2 *max-pool layers, which reduce the size by a factor of 2 in both dimensions.*

Sequential Input data and RNN

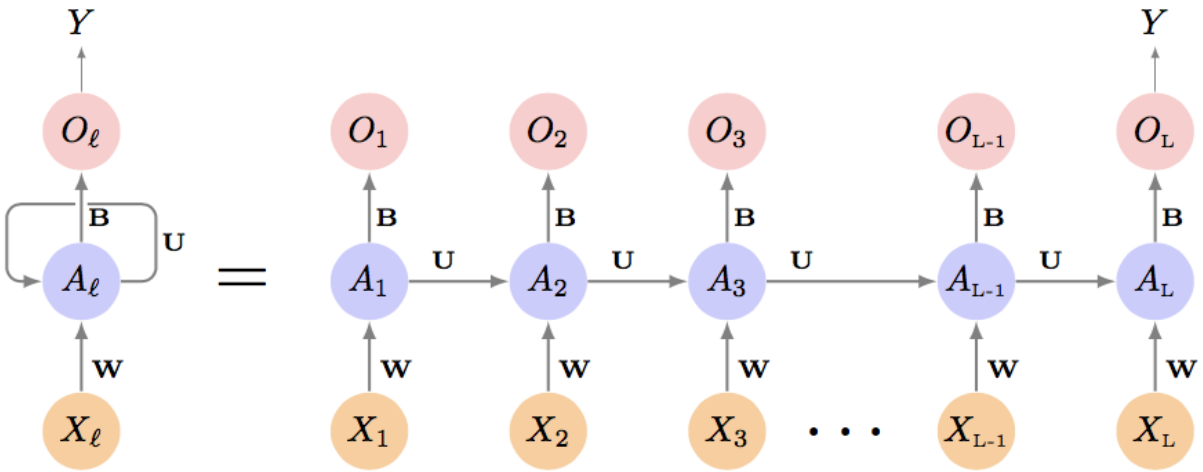
Many data sources are sequential in nature:

- Documents such as book, articles, and tweets. The sequence of words in a document needs to be exploited in tasks such as topic classification, and language translation.
- Time series of temperature, rainfall, wind speed, air quality, and so on.
- Financial time series, where we track market indices, trading volumes, stock and bond prices
- Recorded speech, musical recordings, and other sound recordings.

Structure of Simple Recurrent Neural Networks

- Input: a sequence $X = \{X_1 \cdots X_L\}$
- Sequence of hidden layers: $A = \{A_1 \cdots A_L\}$
- The network processes the input sequence X sequentially.
- At each step, the network updates the activations A_l in the hidden layer, taking as input the vector X_l and the activation vector A_{l-1} from the previous step.
- Then A_l feeds into the output layer and produces a prediction O_l.

Note here O_L is the final output. Intermediate outputs O_l are not used.



STAT562 Lecture 19: Unsupervised Learning

Beidi Qiang

SIUE

Introduction

So far in the semester, we mostly concerns supervised learning methods such as regression and classification.

- In the supervised learning setting, we typically have access to a set of features $X_1, \cdots, X_p$, measured on n observations, and a response Y also measured on the same observations. The goal is then to predict Y using $X_1, \cdots, X_p$.
- In unsupervised learning, we have only a set of features $X_1, \cdots, X_p$ measured on n observations.
  The goal is to discover interesting things about the measurements on $X_1, \cdots, X_p$, such as patterns and clusters.

The Challenge of Unsupervised Learning

Supervised learning has very well developed set of tools at your disposal and a clear way to assess the quality of the results obtained
However, unsupervised learning is often more subjective, and hard to assess.

- Unsupervised learning is often performed as part of an exploratory data analysis
- There is no simple goal for the analysis, such as prediction of a label
- There is no universally accepted mechanism for performing validation

Application of Unsupervised Learning

Unsupervised learning are of growing importance in a number of fields:

- Targeted Ads: Identify groups of shoppers with similar browsing and purchase histories
- Recommendation Systems: identify users with similar preferences
- Anomaly detection: Identify rare items, events or observations which differ significantly from the majority of the data

Clustering Methods

Clustering refers to a very broad set of techniques for finding subgroups, or clusters, in a data set.

- We seek to partition data into distinct groups.
- Observations within each group should be quite similar to each other
- Observations in different groups should be quite different from each other.
- We must first answer the question: what it means for two or more observations to be similar or different.

Applications in Clustering

- **Marketing:** We may have a large number of measurements (e.g.household income, occupation, and so forth) for a large number of people. Our goal is to perform market segmentation by identifying subgroups of people who might be more likely to purchase a particular product.

Two best-known Clustering Approaches

- K-means clustering: partition the observations into a pre-specified K distinct clusters.
- Hierarchical clustering: No pre-specified number of clusters. Create a tree-like visual representation of the observations, that allows us to view clusterings obtained for each possible number of clusters.

K-Means Clustering

A simple approach for partitioning a data set into K distinct, non-overlapping clusters.

- K, the number of clusters, is pre-defined.
- Each observation belongs to exactly one of the K clusters.
- The clusters are non- overlapping
- The idea behind K-means clustering is that a good clustering is one for which the within-cluster variation is as small as possible.

Deriving K-Means Clustering

- Let C_k be the indices of the observations in cluster k.
- We seek to minimize the total within-cluster variation, $W(C_k)$, i.e,

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

- A common choice is of measuring how observations are differ from each other is the squared Euclidean distance:

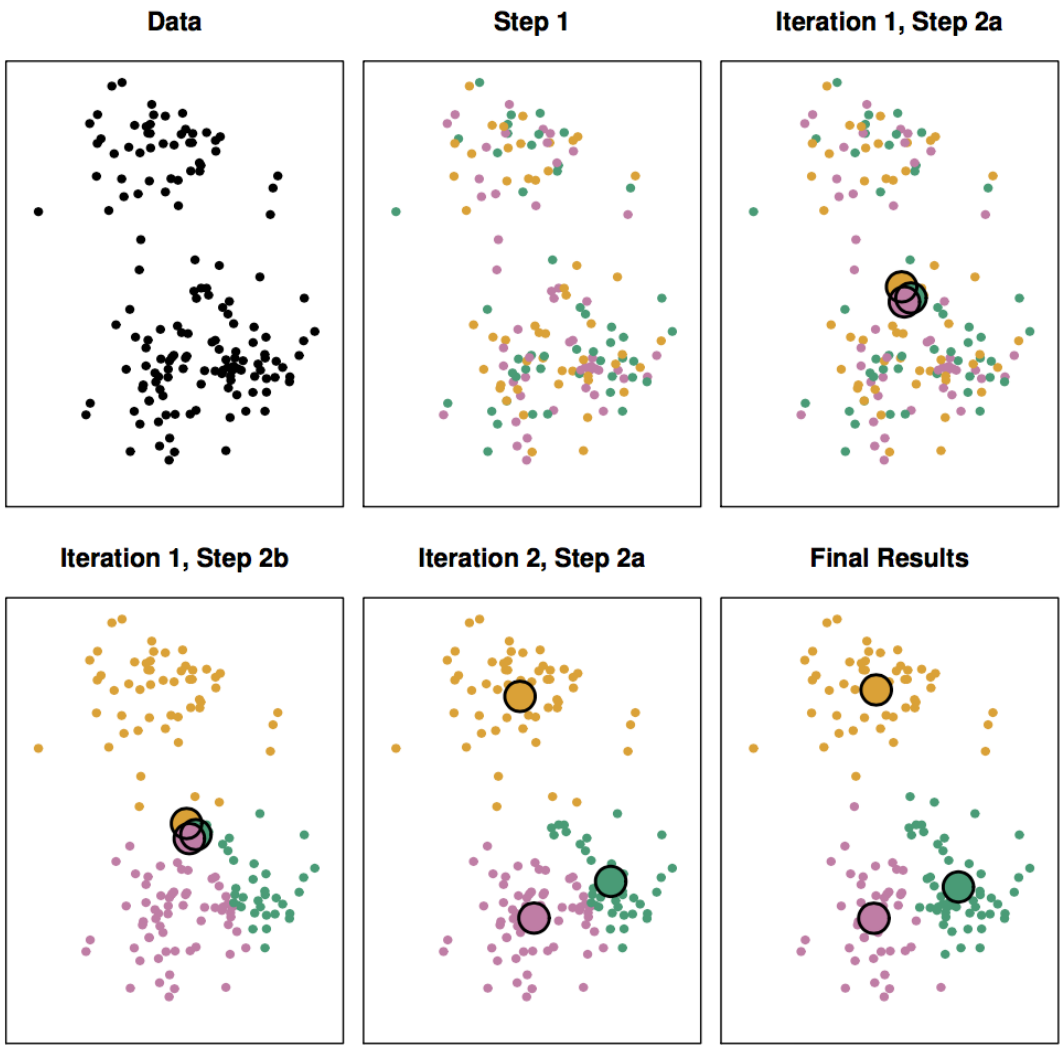
$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

- This is in fact a very difficult optimization problem to solve precisely, since there are almost K^n ways to partition. We would like to find an algorithm find a solution.

Algorithm: K-Means Clustering

- Randomly assign an initial cluster, from 1 to K, to each of the observations.
- Iterate the following until the cluster assignments stop changing:
  (1) For each of the K clusters, compute the cluster centroid, which is the vector of the p feature means for the observations in the cluster.
  (2) Assign each observation to the cluster whose centroid is closest (using Euclidean distance).

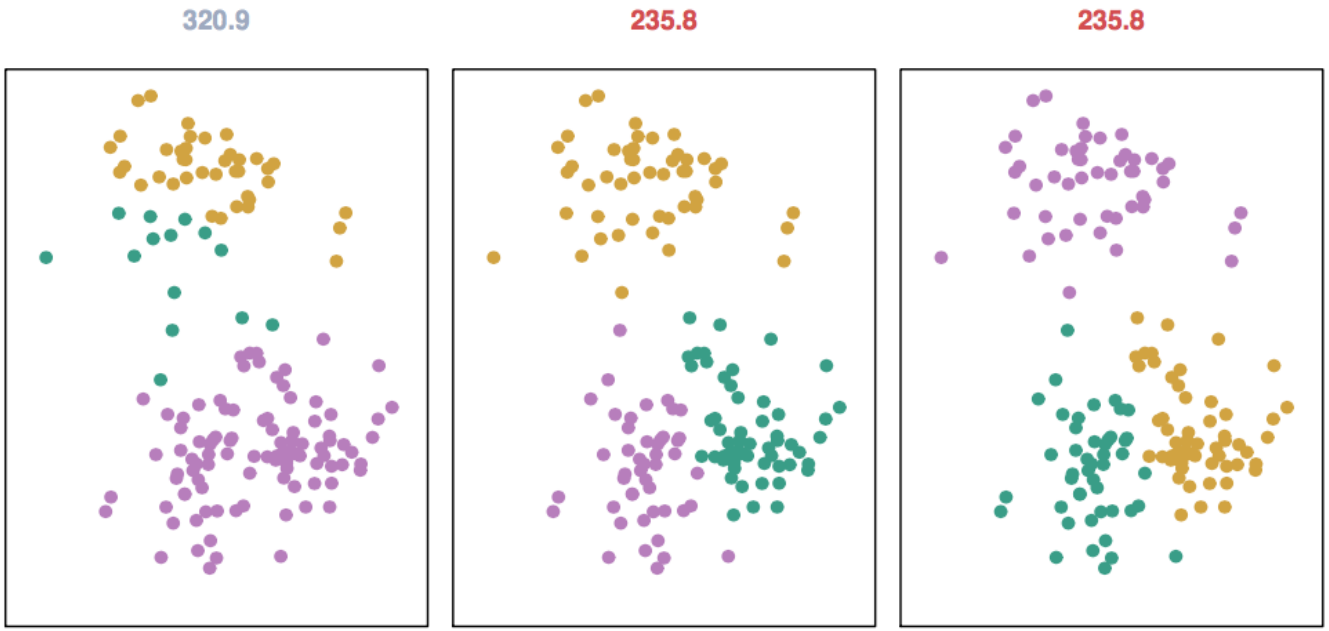
K-means


Data
Step 1
Iteration 1, Step 2a
Iteration 1, Step 2b
Iteration 2, Step 2a
Final Results

Comments

- The algorithm of K-means is guaranteed to decrease the total within-cluster variation at each step.
- As the algorithm is run, the clustering obtained will continually improve until the result no longer changes.
- K-means algorithm finds a local rather than a global optimum. So the results obtained will depend on the initial (random) cluster.
- It is important to run the algorithm multiple times from different random initial configurations.

K-means with Different Initial Clusters



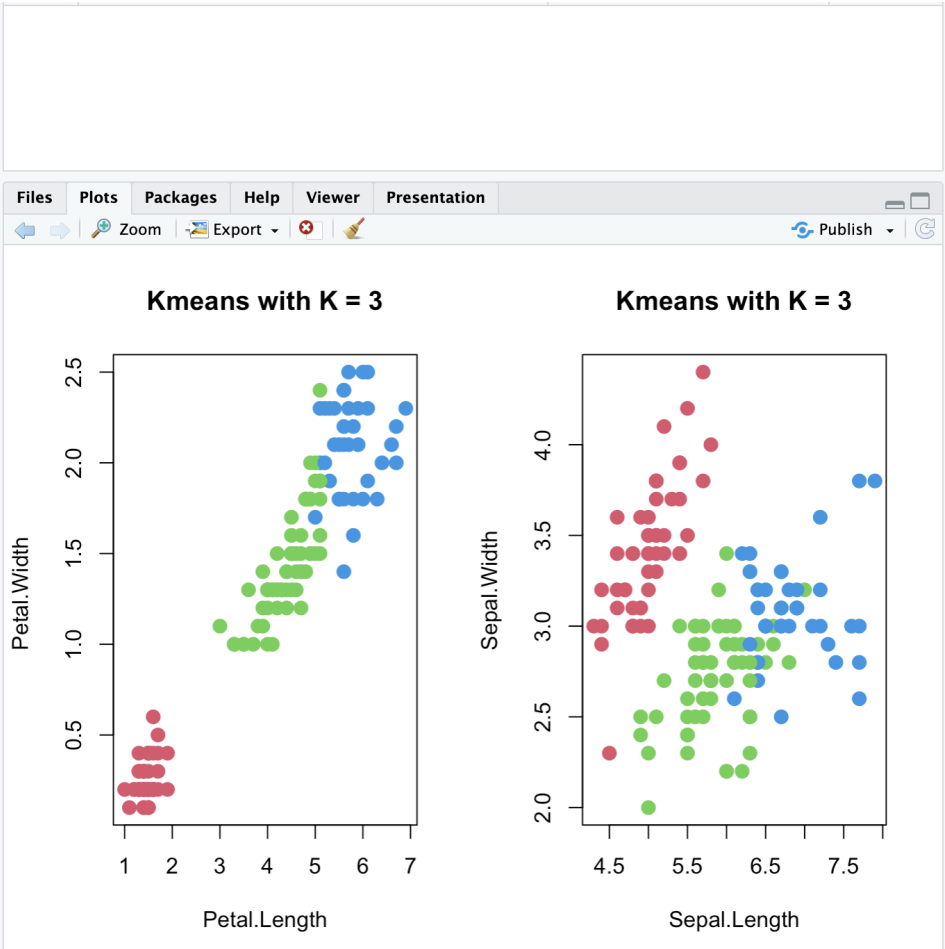
Working Example

R 4.3.1 · ~/

```
> x=c(1,2,3,5,8,9,10)
> A0=c(1,2,8,9)
> B0=c(3,5,10)
> centroid0=c(mean(A0),mean(B0))
> centroid0
[1] 5 6
> A1=c(1,2,3,5)
> B1=c(8,9,10)
> centroid1=c(mean(A1),mean(B1))
> centroid1
[1] 2.75 9.00
>
> kmeans(x,2)
K-means clustering with 2 clusters of sizes 4, 3

Cluster means:
  [,1]
1 2.75
2 9.00

Clustering vector:
[1] 1 1 1 1 2 2 2

Within cluster sum of squares by cluster:
[1] 8.75 2.00
 (between_SS / total_SS =  86.2 %)
```

Implement in R

```
R 4.3.1 · ~/
> x=iris[,1:4]
> km.iris=kmeans(x, 3)
> par(mfrow = c(1, 2))
> plot(x[,3:4], col = (km.iris$cluster + 1), main = "Kmeans with K = 3",
xlab = "Petal.Length", ylab = "Petal.Width ", pch = 20, cex = 2)
> plot(x[,1:2], col = (km.iris$cluster + 1), main = "Kmeans with K = 3",
xlab = "Sepal.Length", ylab = "Sepal.Width", pch = 20, cex = 2)
>
> km.iris
K-means clustering with 3 clusters of sizes 50, 62, 38

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.006000    3.428000     1.462000    0.246000
2     5.901613    2.748387     4.393548    1.433871
3     6.850000    3.073684     5.742105    2.071053

Clustering vector:
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [26] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [51] 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [76] 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[101] 3 2 3 3 3 3 2 3 3 3 3 3 2 2 3 3 3 2 3 2 3 2 3
[126] 3 2 2 3 3 3 3 2 3 3 3 3 2 3 3 3 2 3 3 3 2 3 3 2

Within cluster sum of squares by cluster:
[1] 15.15100 39.82097 23.87947
 (between_SS / total_SS =  88.4 %)

Available components:

[1] "cluster"      "centers"      "totss"
[4] "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
>
```

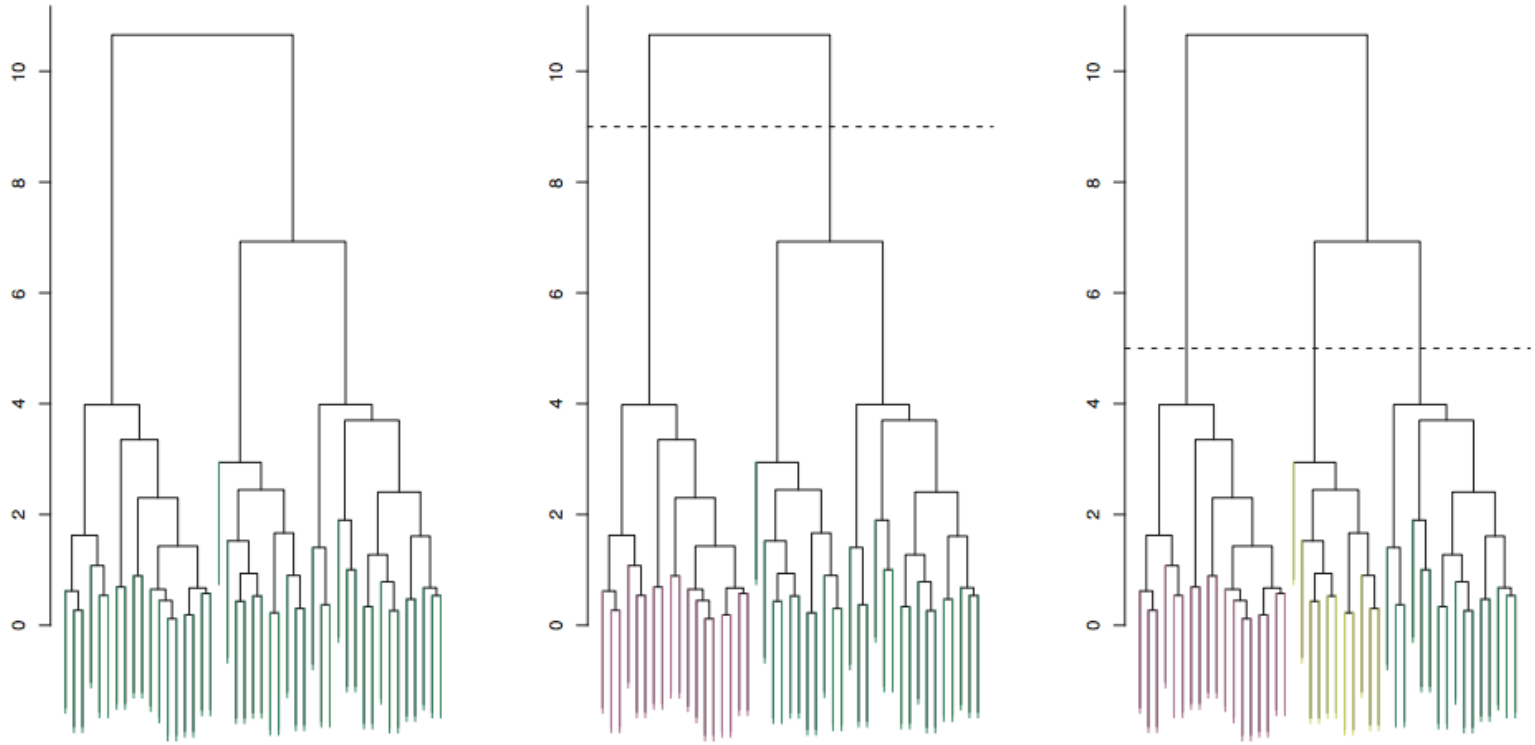


Hierarchical Clustering

Hierarchical clustering is an alternative clustering approach which does not require that we commit to a particular choice of K.

- Hierarchical Clustering results in an attractive tree-based representation, called a dendrogram.
- Most common type of hierarchical clustering: bottom-up. The dendrogram is built starting from the leaves and combining clusters up to the trunk.

Dendrogram



Interpreting Dendrogram

- Each leaf of the dendrogram represents one of the observations.
- As we move up the tree, some leaves begin to fuse into branches. and as we move higher up the tree, branches themselves fuse, either with leaves or other branches.
- For any two observations, we can look for the point in the tree where branches containing those two observations are first fused.
  The height (vertical axis) of this fusion indicates how different the two observations are.
- We make a horizontal cut across the dendrogram to identify clusters.
  Cutting at different height allow us to obtain different number of clusters.

The Hierarchical Clustering Algorithm

- Start with each observation as its own cluster. Define a measure (such as Euclidean distance) of dissimilarities.
- In each iteration,
  a. Examine all pairwise inter-cluster dissimilarities among the clusters and identify the pair of clusters that are least dissimilar. Fuse these two clusters. The dissimilarity between these two clusters indicates the height at which the fusion should be placed.
  b. Compute the new pairwise inter-cluster dissimilarities among the remaining clusters.

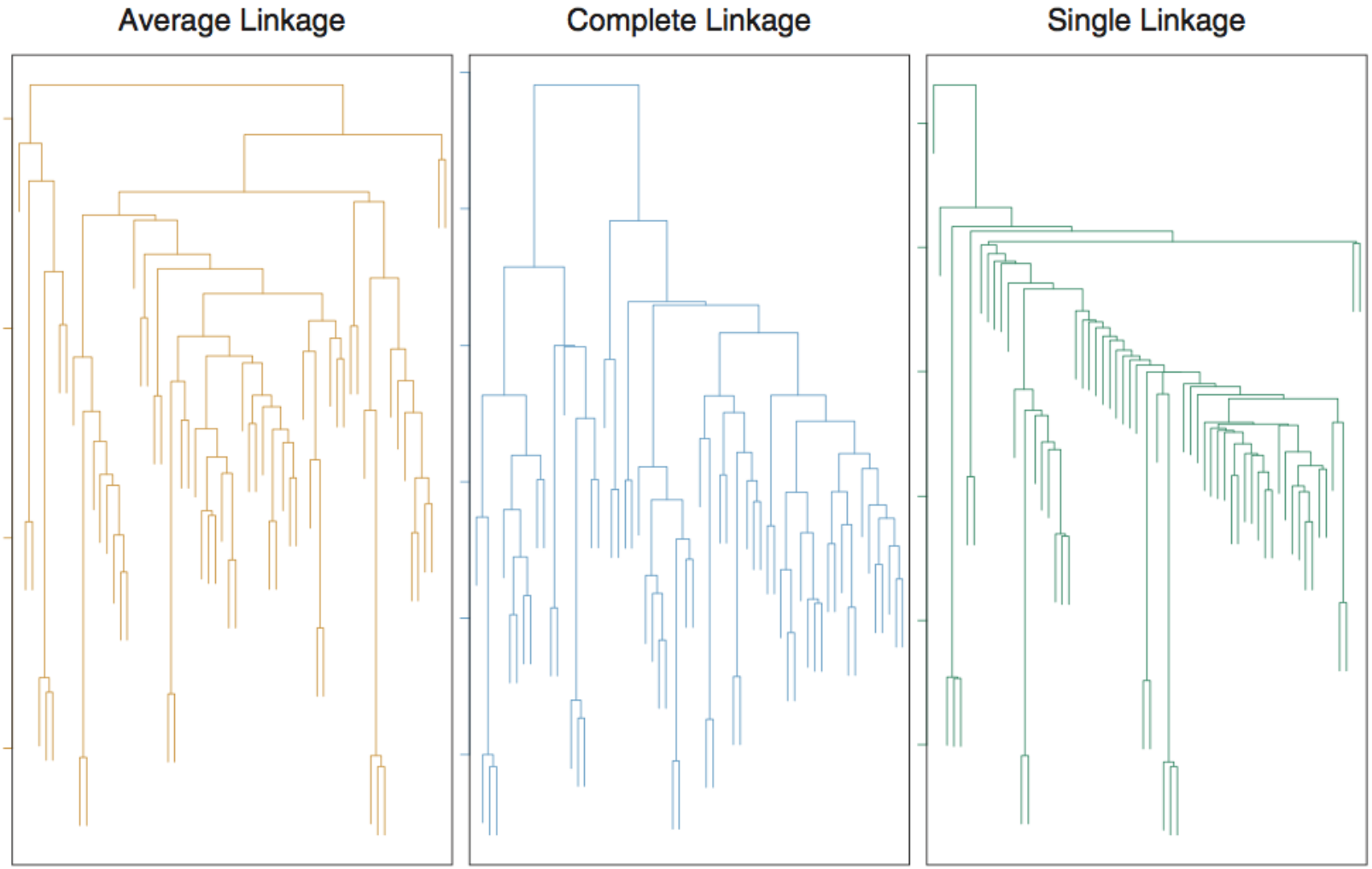
Inter-cluster dissimilarities

We use linkage to defines the dissimilarity between two groups of observations. The common choices of linkage are:

- **Complete**: Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the largest of these dissimilarities.
- **Single**: Compute all pairwise dissimilarities, and record the smallest of these dissimilarities.
- **Average**: Compute all pairwise dissimilarities, and record the average of these dissimilarities.
- **Centroid**: Compute dissimilarity between the centroid(mean) for cluster A and the centroid for cluster B.

Linkage

Dendrogram typically depends quite strongly on the type of linkage used.



The choice of dissimilarity measure

The choice of dissimilarity measure also has a strong effect on the resulting dendrogram.

- Euclidean distance as a common dissimilarity measure, but we can also use correlation-based distance.
- Two observations to be similar if their features are highly correlated.
- Correlation-based distance focuses on the shapes of observation profiles rather than their magnitudes.
- In addition to selecting the dissimilarity measure, one must also consider whether or not the variables should be scaled: the choice of units can will greatly affect the dissimilarity measure obtained.

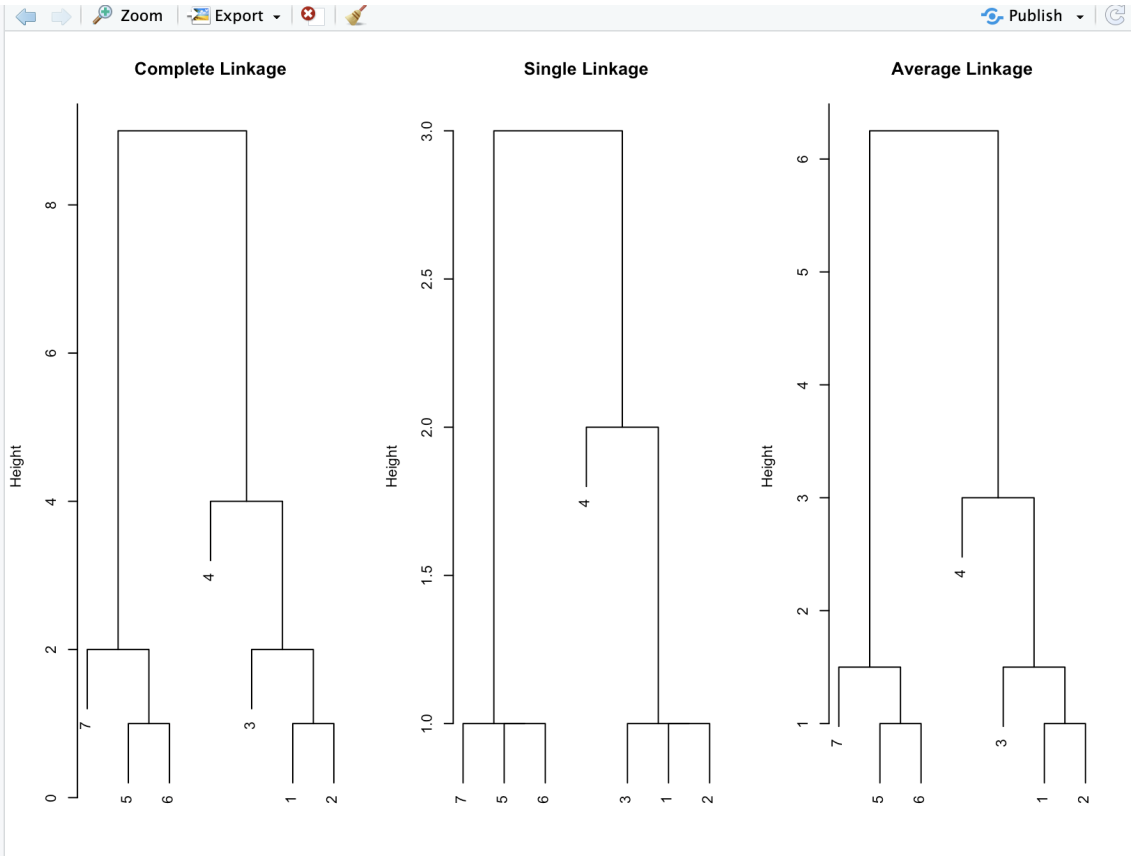
Additional Comments

- The term hierarchical refers to the fact that clusters obtained using hierarchical clustering are nested.
- This means if we'd like to obtain K clusters by cutting the dendrogram, the result will be nested in clusters obtained by cutting at a higher level (resulting K-1 clusters).
- If this is not a desired assumption, K means will likely give better results, since the clusters obtained are not nested.

Example

R 4.3.1 · ~/

```
> x=c(1,2,3,5,8,9,10)
>
> hc.complete=hclust(dist(x), method="complete")
> hc.single=hclust(dist(x), method="single")
> hc.average=hclust(dist(x), method="average")
> par(mfrow=c(1,3 ))
> plot(hc.complete,main="Complete Linkage")
> plot(hc.single,main="Single Linkage")
> plot(hc.average,main="Average Linkage")
>
```



Implement in R

```
R 4.3.1 · ~/
> x=iris[,1:4]
> hc.iris=hclust(dist(x), method="complete")
> cls=cutree(hc.iris,3)
> par(mfrow = c(1, 2))
> plot(x[,3:4], col = (cls + 1), main = "hierarchical", xlab = "Petal.Length", yla
b = "Petal.Width ", pch = 20, cex = 2)
> plot(x[,1:2], col = (cls + 1), main = "hierarchical", xlab = "Sepal.Length", yla
b = "Sepal.Width", pch = 20, cex = 2)
>
> #You may scale the data first
> x.scaled=scale(x)
>
> #Using corrlation as measure of dissimilarity
> dd=as.dist(1-cor(t(x)))
> hc.iris=hclust(dd, method="complete")
> cls=cutree(hc.iris,3)
> plot(x[,3:4], col = (cls + 1), main = "hierarchical", xlab = "Petal.Length", yla
b = "Petal.Width ", pch = 20, cex = 2)
> plot(x[,1:2], col = (cls + 1), main = "hierarchical", xlab = "Sepal.Length", yla
b = "Sepal.Width", pch = 20, cex = 2)
>
```