

STA 695: Statistical Machine Learning and Predictive Modeling

Zeya Wang

University of Kentucky

Fall 2024

Outline of the Course

- ① Introduction to Machine Learning
- ② Supervised Learning & Classification
- ③ Model Assessment
- ④ Model Selection
- ⑤ Data Preprocessing
- ⑥ Ensemble Methods
- ⑦ Neural Networks/Deep Learning
- ⑧ Unsupervised Learning
- ⑨ Intro to Extra Topics (e.g., Graphical Models)

Introduction to Machine Learning

- Machine learning: building statistical models and algorithms that allow computers to perform specific tasks on data

| CS | STAT | Applied Math |
|-----------|--------------------------|--------------------|
| Algorithm | Probability | Optimization |
| Database | Stat inference | Numerical analysis |
| | Sample size & complexity | |

- Predictive modeling: the process of developing a mathematical tool or model that generates an accurate prediction

Introduction to Machine Learning

- Machine learning: building statistical models and algorithms that allow computers to perform specific tasks on data

| CS | STAT | Common Areas |
|------------------------|------------------------|--|
| Deep Learning | Confidence Sets | Prediction (Regression & Classification) |
| Reinforcement Learning | Large Sample Theory | Probability Bounds |
| Efficient Computation | Statistical Optimality | Clustering |
| Large Language Model | Causality | Graphical Models |

- Predictive modeling: the process of developing a mathematical tool or model that generates an accurate prediction

Introduction to Machine Learning

- Example 1: Email Spam. X_i : an email;

$$Y_i = \begin{cases} 0, & \text{email} \\ 1, & \text{spam} \end{cases}$$

- Feature extraction: e.g. frequency of certain key words
- A classification problem: 0-1
- Metric: misclassification rate

Introduction to Machine Learning

- Example 2. Predict the log of prostate-specific antigen (PSA) from a number of measurements
- A regression problem.
- Example 3. Handwritten digit recognition.
- X_i : a 16×16 eight-bit grayscale maps; $Y_i \in \{0, 1, \dots, 9\}$
- Feature transformation: vectorized X_i or summary statistics, e.g., marginal histograms or numbers of crossing changes

Introduction to Machine Learning

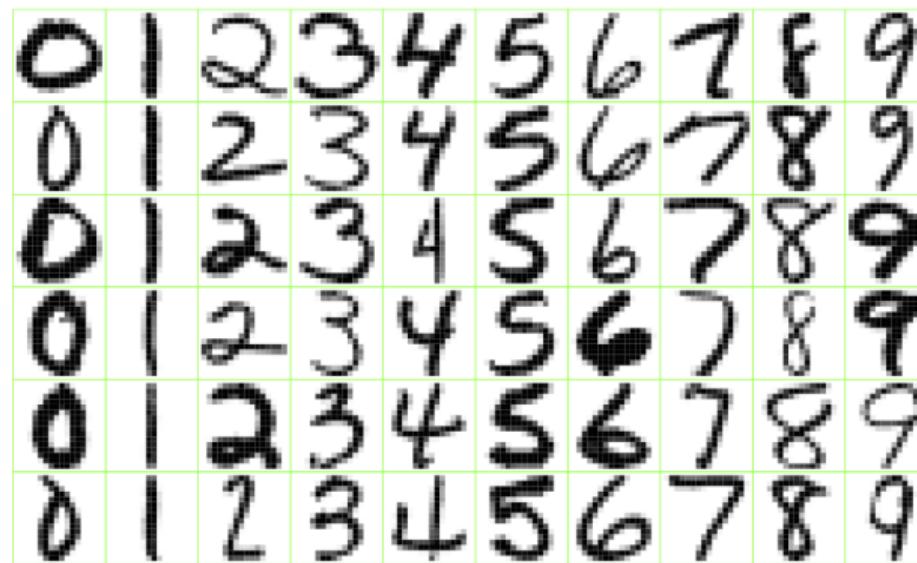
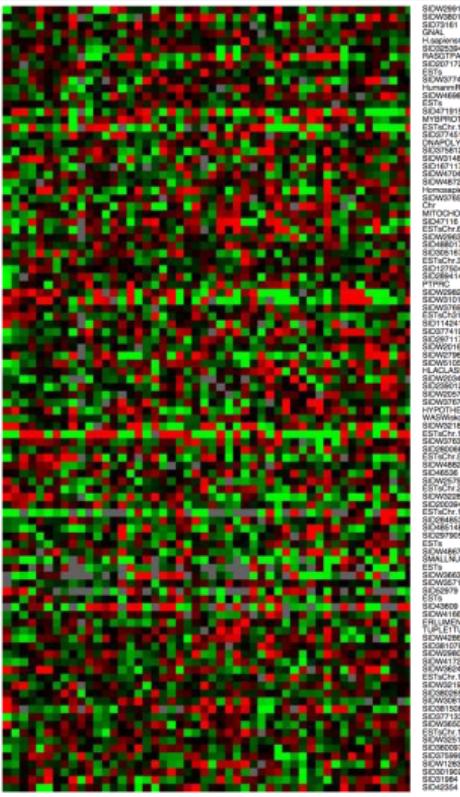


FIGURE 1.2. Examples of handwritten digits from U.S. postal envelopes.

Introduction to Machine Learning

- Example 4. Microarray gene expression data.
- X_i : gene expression levels
- Supervised learning: a classification problem when we have a label. Y_i : tumor types. (a typical small n , large p problem)
- Unsupervised learning: find subtypes of cancer when we don't have a label (clustering analysis)
- Semi-supervised learning: we have a label but possibly novel tumor subtypes in prediction

Introduction to Machine Learning



Introduction to Machine Learning

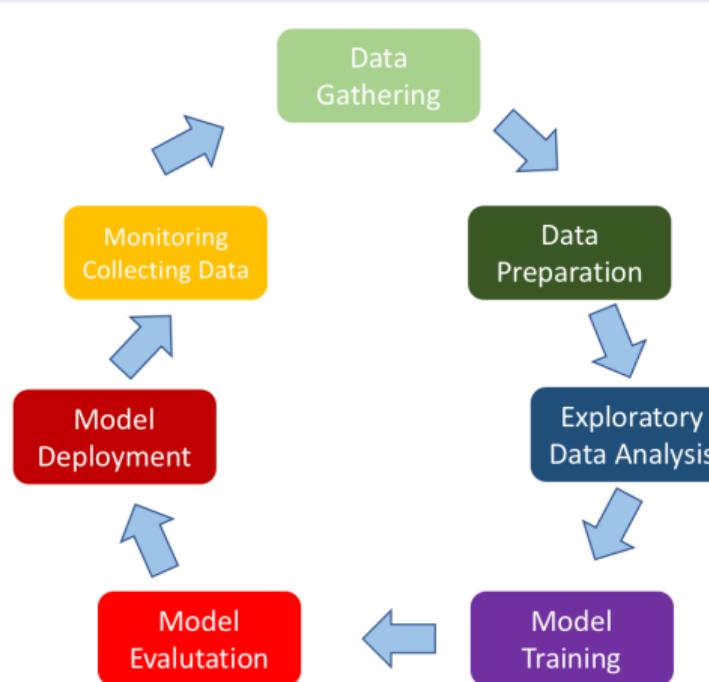


Figure: Life cycle of machine learning

Supervised Learning vs. Unsupervised Learning

- Supervised Learning: have labels and outcomes $\{Y_{n \times 1}, X_{n \times p}\}$
 - $X_{n \times p}$ Data Matrix (given or fixed)
 - $Y_{n \times 1}$ Outcomes: ordinal/quantitative (regression)/binary (classification)/categorical (classification)
- Unsupervised Learning
 - Clustering groups of samples/features
 - Pattern Recognition
 - Association between features

Training vs. Testing

- Training: fit a model
 $\{X^{train}, Y^{train}\}$
- Testing: assess the performance
 $\{X^{test}, Y^{test}\}$
- Overfitting: fit training data well, but terribly predict test data

Least Squares & Linear Regression

- Linear model

$$\begin{aligned}\hat{Y} &= \hat{\beta}_0 + \sum_{j=1}^p X_j \hat{\beta}_j \\ &= X \hat{\beta}\end{aligned}$$

$$X = [1, X_1, \dots, X_p], \hat{\beta} = [\hat{\beta}_0, \dots, \hat{\beta}_p]^T$$

- Residual sum of squares (RSS)

$$\begin{aligned}RSS(\beta) &= \sum_{i=1}^N (y_i - x_i^T \beta)^2 \\ &= (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta) \\ &= \|\mathbf{y} - \mathbf{X}\beta\|_2^2\end{aligned}$$

Least Squares & Linear Regression

- Minimize RSS: $\min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2$

$$\begin{aligned}\frac{\partial \text{RSS}}{\partial \beta} &= 0 \\ -\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) &= 0 \\ \hat{\beta} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}\end{aligned}$$

- Hat matrix H

$$\begin{aligned}\hat{Y} &= \mathbf{X}\hat{\beta} \\ &= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \\ &= Hy\end{aligned}$$

Least Squares & Linear Regression

- Property of $\hat{\beta}$ (BLUE)

$$\begin{aligned} E[\hat{\beta}] &= \beta \\ Var(\hat{\beta}) &= (X^T X)^{-1} \sigma^2 \end{aligned}$$

- Gauss-Markov theorem: $\hat{\beta}$ has minimum variance among all linear unbiased estimators
- Estimate of the variance σ^2

$$\hat{\sigma}^2 = \frac{1}{n-p-1} (y - \hat{y})^T (y - \hat{y})$$

- Property of $\hat{\sigma}$

$$(N - p - 1) \hat{\sigma}^2 \sim \sigma^2 \chi_{N-p-1}^2$$

Least Squares & Linear Regression

- T-test $\beta_j = 0$

$$z_j = \frac{\hat{\beta}_j}{\hat{\sigma} \sqrt{(X^T X)^{-1}_{j \times j}}} \sim t_{N-p-1}$$

- F-test $\beta_j = \beta_{j+1} = \dots = \beta_{j+k} = 0$

$$F = \frac{(RSS_0 - RSS_1) / (p_1 - p_0)}{RSS_1 / (N - p_1 - 1)} \sim F_{p_1 - p_0, N - p_1 - 1}$$

Least Squares & Linear Regression

- Multiple outputs

$$\begin{aligned}Y_k &= \beta_{0k} + \sum_{j=1}^p X_j \beta_{jk} + \varepsilon_k \\ Y &= XB + E\end{aligned}$$

$Y = [Y_1, Y_2, \dots, Y_K]$, $B = [\beta_1, \beta_2, \dots, \beta_K]$, where $\beta_k = [\beta_{0k}, \dots, \beta_{pK}]^T$

- Loss function (when error is uncorrelated):

$$\begin{aligned}RSS(B) &= \sum_{k=1}^K \sum_{i=1}^N (Y - \beta_{0k} - \sum_{j=1}^p X_j \beta_{jk})^2 \\ &= \text{tr}[(Y - XB)^T (Y - XB)]\end{aligned}$$

Issues for Least Square

- Colinearity: correlated predictors
 - $p > n \Rightarrow$ overfitting!
 - $X^T X$ (nearly) singular, not invertible!
- Low bias but high variance
- Interpretation
- Feature selection/Shrinkage method
 - Subset selection
 - Shrinkage/regularization method

Subset Selection

- Best-subset selection: leaps and bounds *leaps()*
- Forward-stepwise selection: QR Decomposition (ESLII Exercise 3.9)
- Backward-stepwise selection: Z-score (ESLII Exercise 3.10)
- Forward stagewise regression
 1. Start with $\mathbf{r} = \mathbf{y} - \bar{\mathbf{y}}$, $\beta_1, \beta_2, \dots, \beta_p = 0$.
 2. Find the predictor \mathbf{x}_j most correlated with \mathbf{r} .
 3. Update $\beta_j \leftarrow \beta_j + \delta_j$, where $\delta_j = \frac{\langle \mathbf{r}, \mathbf{x}_j \rangle}{\langle \mathbf{x}_j, \mathbf{x}_j \rangle}$
 4. Update $\mathbf{r} \leftarrow \mathbf{r} - \delta_j \mathbf{x}_j$, and repeat steps 2 and 3 until no predictor has any correlation with \mathbf{r} .

Shrinkage or regularization methods

- Regularized or penalized RSS:

$$\text{PRSS}(\beta) = \text{RSS}(\beta) + \lambda J(\beta).$$

- λ : penalization parameter (tuning/hyperparameter) to be determined
- $J()$: constraint prior; log prior density (a Bayesian interpretation)

Ridge Regression

- $J(\beta) = \sum_{j=1}^p \beta_j^2$ (Tikhonov Regularization)

$$\hat{\beta}^{\text{ridge}} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^N \left(y_i - \beta_0 - \sum_{j=1}^p x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^p \beta_j^2 \right\}.$$

$\lambda \geq 0$ controls the amount of shrinkage

- Equivalent problem:

$$\hat{\beta}^{\text{ridge}} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^N \left(y_i - \beta_0 - \sum_{j=1}^p x_{ij} \beta_j \right)^2,$$

$$\text{subject to } \sum_{j=1}^p \beta_j^2 \leq t,$$

- Closed-form solution

$$\hat{\beta}^{\text{ridge}} = (X^T X + \lambda I)^{-1} X^T Y.$$

Ridge Regression

- Nonsingular even if $X^T X$ is not full rank

- Biased but small variances,

$$E(\hat{\beta}^R) = (X^T X + \lambda I)^{-1} X^T X \beta,$$

$$\text{Var}(\hat{\beta}^R) = \sigma^2 (X^T X + \lambda I)^{-1} X^T X (X^T X + \lambda I)^{-1} \leq \text{Var}(\hat{\beta}),$$

- Prior of β : $\beta_j \sim N(0, \tau^2)$. (ESLII Exercise 3.6)

Ridge Regression

- Special case: orthonormal inputs $\hat{\beta}^{\text{ridge}} = \frac{\hat{\beta}}{1+\lambda}$
- (Thin) SVD decomposition (when $N > p$)

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$$

\mathbf{U} : $N \times p$ matrix; \mathbf{V} : $p \times p$ matrix; \mathbf{D} : $p \times p$ diagonal matrix;

- \mathbf{U} : column vectors u_1, u_2, \dots, u_p form a orthonormal set, i.e., $\mathbf{U}^T \mathbf{U} = I$
- \mathbf{V} : column vectors v_1, v_2, \dots, v_p form a orthonormal set, i.e., $\mathbf{V}^T \mathbf{V} = I$

Ridge Regression

- (Thin) SVD decomposition (when $N > p$)

-

$$\begin{aligned}\mathbf{X} \hat{\beta}^{1s} &= \mathbf{X} (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \\ &= \mathbf{U} \mathbf{U}^T \mathbf{y}\end{aligned}$$

-

$$\begin{aligned}\mathbf{X} \hat{\beta}^{\text{ridge}} &= \mathbf{X} (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y} \\ &= \mathbf{U} \mathbf{D} (\mathbf{D}^2 + \lambda \mathbf{I})^{-1} \mathbf{D} \mathbf{U}^T \mathbf{y} \\ &= \sum_{j=1}^p \mathbf{u}_j \frac{d_j^2}{d_j^2 + \lambda} \mathbf{u}_j^T \mathbf{y}\end{aligned}$$

Ridge Regression

- Effective degree of freedom

$$df(\lambda) = \text{tr} \left[X (X^T X + \lambda I)^{-1} X^T \right] \leq df(0) = \text{tr} \left(X (X^T X)^{-1} X^T \right) = \\ \text{tr} \left((X^T X)^{-1} X^T X \right) = p$$

- X is orthonormal

$$df(\lambda) = \sum_{j=1}^p \frac{d_j^2}{d_j^2 + \lambda}$$

Lasso Regression

- $J(\beta) = \sum_{j=1}^p |\beta_j|.$
- Equivalent problem:

$$\hat{\beta}^{\text{lasso}} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^N \left(y_i - \beta_0 - \sum_{j=1}^p x_{ij} \beta_j \right)^2,$$

$$\text{subject to } \sum_{j=1}^p |\beta_j| \leq t,$$

- No closed-form solution

Lasso Regression

- Prior of β : β_j Laplace or DE $(0, \frac{1}{\lambda})$;
- Biased but small variances
- $\hat{\beta}_j^{Lasso} = 0$ for large λ (but not $\hat{\beta}_j^{Ridge}$)
- $df(\hat{\beta}^{Lasso}) = \# \text{ of non-zero } \hat{\beta}_j^{Lasso} \text{ 's}$ (Zou et al).
- Special case: orthonormal inputs:
 - $\hat{\beta}_j^{Lasso} = ST(\hat{\beta}_j, \lambda) = \text{sign}(\hat{\beta}_j)(|\hat{\beta}_j| - \lambda)_+$
 - $\hat{\beta}_j^{Ridge} = \hat{\beta}_j / (1 + \lambda)$
 - $\hat{\beta}_j^{Best} = HT(\hat{\beta}_j, M) = \hat{\beta}_j I(\text{rank}(\hat{\beta}_j) \leq M)$

Lasso Regression

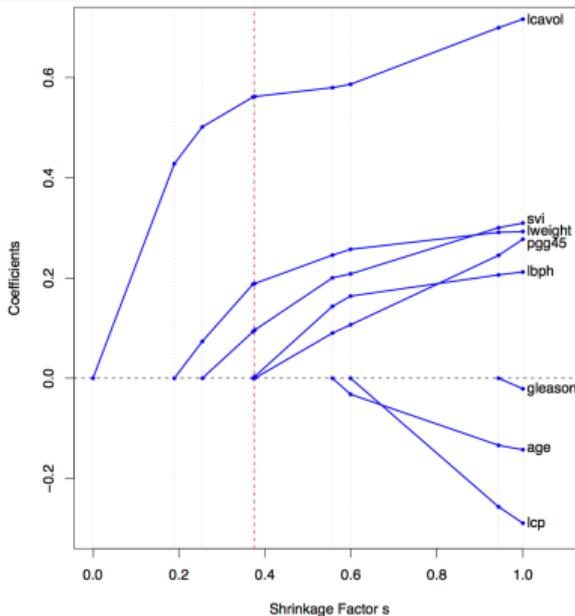


FIGURE 3.10. Profiles of lasso coefficients, as the tuning parameter t is varied. Coefficients are plotted versus $s = t / \sum_1^p |\hat{\beta}_j|$. A vertical line is drawn at $s = 0.36$, the value chosen by cross-validation. Compare Figure 3.8 on page 65; the lasso profiles hit zero, while those for ridge do not. The profiles are piece-wise linear, and so are computed only at the points displayed; see Section 3.4.4 for details.

Lasso vs. Ridge

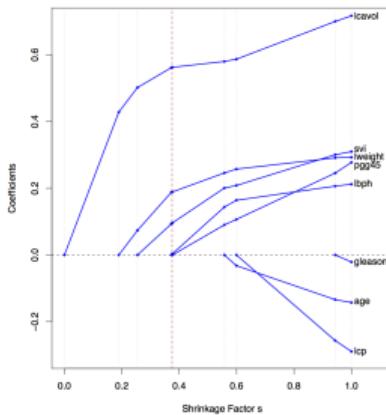


FIGURE 3.10. Profiles of lasso coefficients, as the tuning parameter t is varied. Coefficients are plotted versus $s = t / \sum_i |\hat{\beta}_i|$. A vertical line is drawn at $s = 0.36$, the value chosen by cross-validation. Compare Figure 3.8 on page 65; the lasso profiles hit zero, while those for ridge do not. The profiles are piece-wise linear, and so are computed only at the points displayed; see Section 3.4.4 for details.

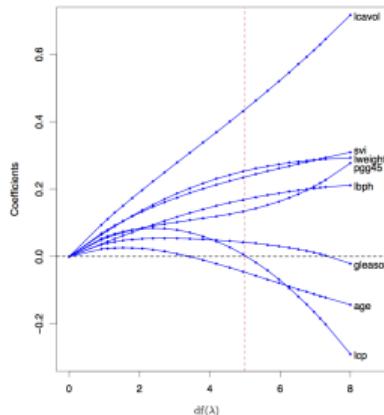
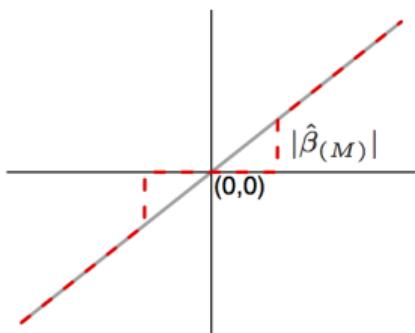


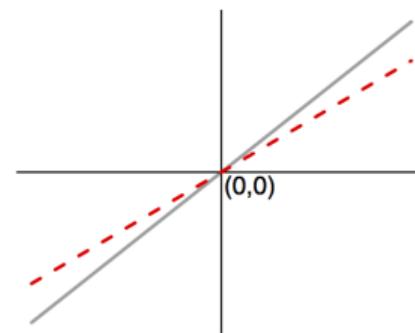
FIGURE 3.8. Profiles of ridge coefficients for the prostate cancer example, as the tuning parameter λ is varied. Coefficients are plotted versus $df(\lambda)$, the effective degrees of freedom. A vertical line is drawn at $df = 5.0$, the value chosen by cross-validation.

Lasso vs. Ridge

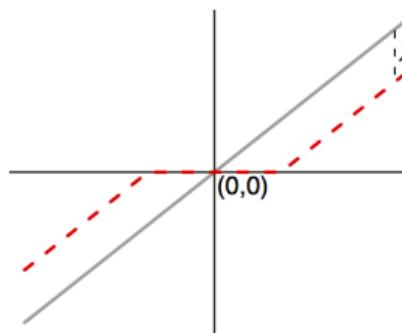
Best Subset



Ridge



Lasso



Lasso vs. Ridge

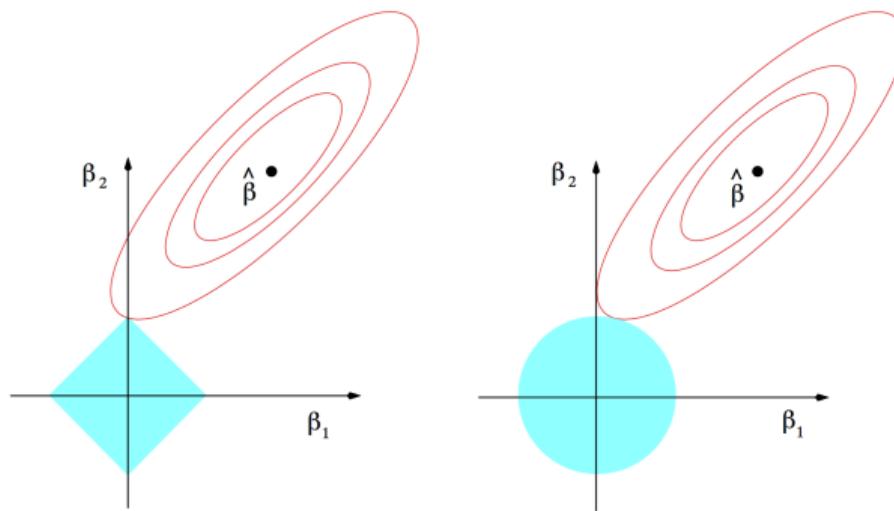


FIGURE 3.11. Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions $|\beta_1| + |\beta_2| \leq t$ and $\beta_1^2 + \beta_2^2 \leq t^2$, respectively, while the red ellipses are the contours of the least squares error function.

Regularization methods

- Generalized ridge and lasso

$$\hat{\beta}^{\text{ridge}} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^N \left(y_i - \beta_0 - \sum_{j=1}^p x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^p |\beta_j|^q \right\}.$$

$$q \geq 0$$

- Elastic-net (Zou and Hastie (2005))

$$\lambda \sum_{j=1}^p (\alpha \beta_j^2 + (1 - \alpha) |\beta_j|)$$

$$0 < \alpha < 1$$

Least Angle Regression (LAR)

Algorithm 3.2 Least Angle Regression.

1. Standardize the predictors to have mean zero and unit norm. Start with the residual $\mathbf{r} = \mathbf{y} - \bar{\mathbf{y}}$, $\beta_1, \beta_2, \dots, \beta_p = 0$.
 2. Find the predictor \mathbf{x}_j most correlated with \mathbf{r} .
 3. Move β_j from 0 towards its least-squares coefficient $\langle \mathbf{x}_j, \mathbf{r} \rangle$, until some other competitor \mathbf{x}_k has as much correlation with the current residual as does \mathbf{x}_j .
 4. Move β_j and β_k in the direction defined by their joint least squares coefficient of the current residual on $(\mathbf{x}_j, \mathbf{x}_k)$, until some other competitor \mathbf{x}_l has as much correlation with the current residual.
 5. Continue in this way until all p predictors have been entered. After $\min(N - 1, p)$ steps, we arrive at the full least-squares solution.
-

LAR vs. Lasso

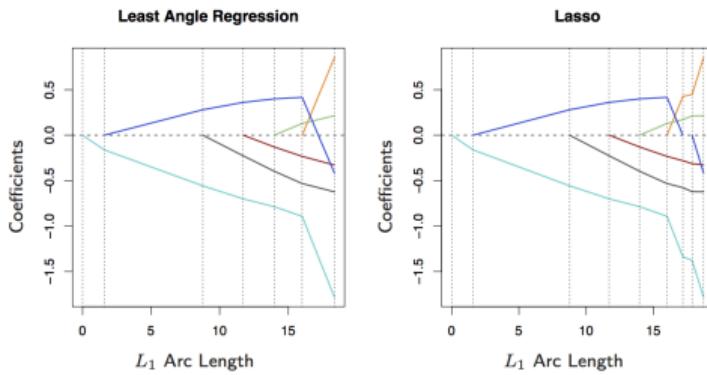


FIGURE 3.15. Left panel shows the LAR coefficient profiles on the simulated data, as a function of the L_1 arc length. The right panel shows the Lasso profile. They are identical until the dark-blue coefficient crosses zero at an arc length of about 18.

LAR vs. Lasso

Algorithm 3.2a Least Angle Regression: Lasso Modification.

- 4a. If a non-zero coefficient hits zero, drop its variable from the active set of variables and recompute the current joint least squares direction.
-

Lasso Algorithms

- Quadratic programming
- LARS (2001)
- Coordinate descent/shooting (2007) in ESL
- Proximal Gradient (2009) (Iterate shrinkage and thresholding algorithm)

Lasso Algorithms

| Aspect | LARS | Coordinate Descent | Gradient Descent | Quadratic Programming |
|---------------|---------------------------|---------------------------|-----------------------------|---------------------------------|
| Solution Path | Full path | Specific λ | Specific λ | Specific λ |
| Speed | Fast, esp. for large sets | Efficient for sparse data | Slower for large sets | Slower, high-dimensional data |
| Scalability | High-dimensional data | Large, sparse datasets | Needs tuning | Not scalable to very large data |
| Complexity | Low complexity | Simple iterative process | Iterative, needs adaptation | Convex optimization |

Lasso Regression

- Shrink non-zero coefficient \Rightarrow Not consistent
- Smoothly clipped absolute deviation (SCAD) penalty (Fan and Li, 2005)

$$J(\beta) = \lambda \{ I(\beta \leq \lambda) + \frac{(\alpha\lambda - \beta)_+}{(\alpha - 1)\lambda} I(\beta > \lambda) \}$$

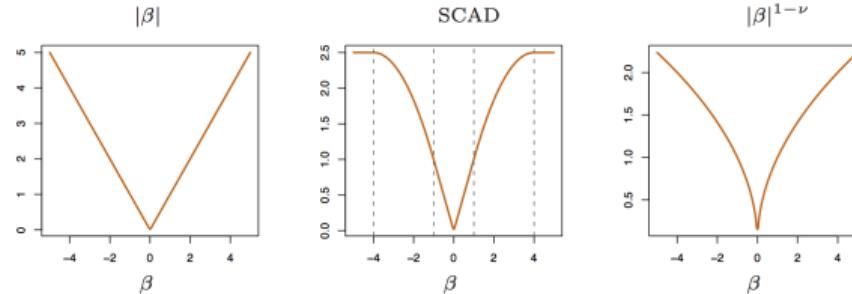


FIGURE 3.20. The lasso and two alternative non-convex penalties designed to penalize large coefficients less. For SCAD we use $\lambda = 1$ and $a = 4$, and $\nu = \frac{1}{2}$ in the last panel.

Lasso Regression

- Adaptive Lasso:

$$J(\beta) = \lambda \sum_i \hat{\omega}_j |\beta_j|$$

β_j increases, ω_j decreases such that less shrinkage. One choice of weights can be:
 $\hat{\omega}_j = 1/|\hat{\beta}_j^{LS}|^\nu$

- MC+: Smooth link between hard-threshold and least square; a minimax concave penalty (MCP) (Zhang, 2010)

R-implementation for Lasso Regression

- Many R-routines to compute a Lasso solution of path, a function of tuning parameter λ .
- Lasso and elastic-net regularized generalized linear models *glmnet*:
<http://cran.r-project.org/web/packages/glmnet/index.html>
- Improved *glmnet*:
https://www.csie.ntu.edu.tw/~cjlin/papers/l1_glmnet/long-glmnet.pdf.
Highly recommended faster than (5-10 times).

R-implementation for Lasso Regression

```
# Fit and predict
install.packages("glmnet")
library(glmnet)
x=matrix(rnorm(100*20),100,20)
y=rnorm(100)
fit.lasso=glmnet(x,y)
coef(fit.lasso,s=0.01)
coef(fit.lasso,s=0.1)
predict(fit.lasso,newx=x[1:10,],s=c(0.01,0.1))
```

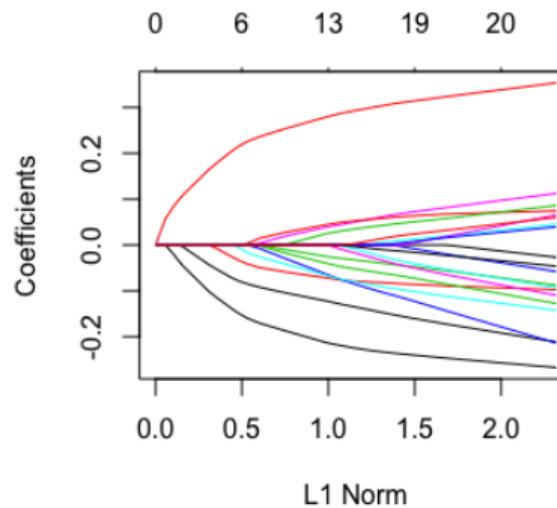
Python-implementation for Lasso Regression

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_regression
from sklearn.linear_model import Lasso
import numpy as np

n_samples, n_features = 1000, 20
rng = np.random.RandomState(0)
X, y = make_regression(n_samples, n_features, random_state=rng)
sample_weight = rng.rand(n_samples)
X_train, X_test, y_train, y_test, sw_train, sw_test = train_test_split(
    X, y, sample_weight, random_state=rng)
reg = Lasso()
reg.fit(X_train, y_train, sample_weight=sw_train)
print(reg.score(X_test, y_test, sw_test))
```

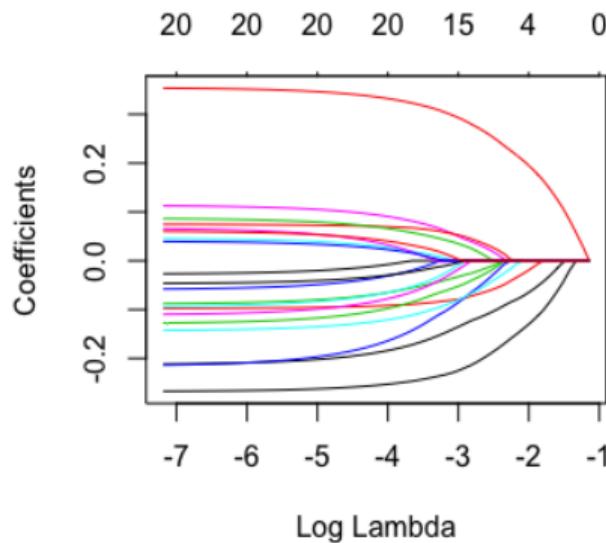
R-implementation for Lasso Regression

```
# Regularization path  
plot(fit.lasso, xvar="norm" )
```



R-implementation for Lasso Regression

```
# Regularization path  
plot(fit.lasso, xvar="lambda")  
# For python, please refer to sklearn.linear_model.lasso_path
```



Implementation

```
# LARS - R
library("lars")
lars.fit <- lars(x,y, type="lasso")
# LARS - Python
from sklearn.linear_model import lars
reg = Lars(n_nonzero_coefs=1)
reg.fit(X_train, y_train)
```

Lasso Regression

- Predictors belong to pre-defined L groups
- Grouped Lasso:

$$\hat{\beta}^{\text{glasso}} = \underset{\beta}{\operatorname{argmin}} \|y_i - \beta_0 \mathbf{1} - \sum_{\ell=1}^L \mathbf{X}_\ell \beta_\ell\|_2^2 + \lambda \sum_{\ell=1}^L \sqrt{p_\ell} \|\beta_\ell\|_2$$

$\sqrt{p_\ell}$: group size

\mathbf{X}_ℓ : Predictors for the ℓ -th group

β_ℓ : coefficient vector for the ℓ -th group

R-implementation

```
library(grpgreg)
library("lars")
group <- c(rep(1,4), rep(2,6))
fit <- grpreg(x,y,group,penalty="grLasso")
plot(fit,main = "Group Lasso")
fit <- grpreg(x,y,group,penalty="grMCP")
plot(fit, main = "Group MCP")
fit <- grpreg(x,y,group,penalty="grSCAD")
plot(fit, main = "Group SCAD")
```

Using Derived Input Directions

- Principle Components Regression (variation in X is useful for predicting Y)

- ① Standardize X
- ② Take SVD of X and find truncated $Z = UD$ (truncated $k < r$)
- ③ regression on Z

$$\hat{\beta}^{PCR} = (Z^T Z)^{-1} Z^T y = (DU^T UD)^{-1} DU^T y = D^{-1} U^T y$$

- Partial Least Square Regression

- PCA maximizes $\text{Cov}(X)$ while PLS maximizes $\text{Cov}(X, y)$
- Find direction with variation in X related to Y

Classification Methods

- $Y_{n \times 1}$: class labels (binary classification: $Y_i = 0$ or 1)
- Decision boundary: a surface that partitions the underlying vector space into two sets, one for each class
- Linear: monotone transformation of discriminant function $\delta_k(x)$ or $Pr(Y = k|X = x)$ is linear
- Linear regression: $E(Y_i | X_i) = \beta_0 + X_i^T \beta$
 - Use LS to estimate β^T 's $\Rightarrow \hat{Y}_i$; $\tilde{Y}_i = I(\hat{Y}_i \geq 0.5)$
 - Decision boundary: $\hat{Y}(x) = \hat{\beta}_0 + x^T \hat{\beta} = 0.5$, linear

Classification Methods

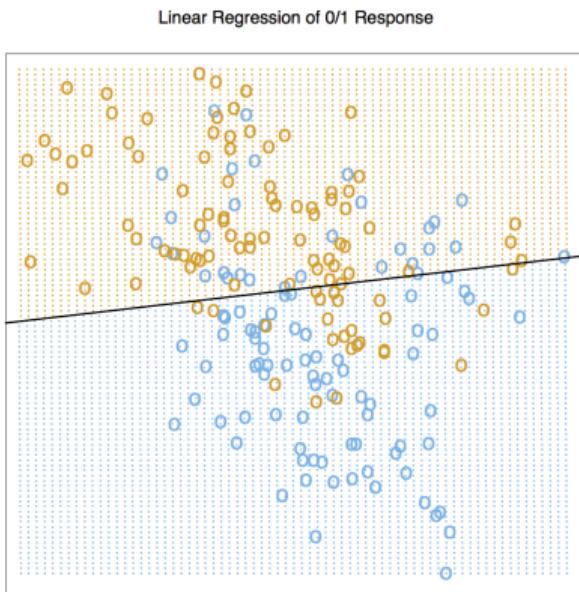


FIGURE 2.1. A classification example in two dimensions. The classes are coded as a binary variable (**BLUE** = 0, **ORANGE** = 1), and then fit by linear regression. The line is the decision boundary defined by $x^T \hat{\beta} = 0.5$. The orange shaded region denotes that part of input space classified as **ORANGE**, while the blue region is classified as **BLUE**.

Nearest Neighbor Methods

- KNN: $N_{k(x)}$ is the k nearest training data points that are closest to x (Euclidean distance)

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} Y_i.$$

- Majority vote: $\hat{Y}(x) > 0.5$
- Key: choice of k , or how much "smoothness" is to be assumed
- Modeling assumption: larger k , higher or lower model complexity?
- Try a few values of k , then ...

Nearest Neighbor Methods

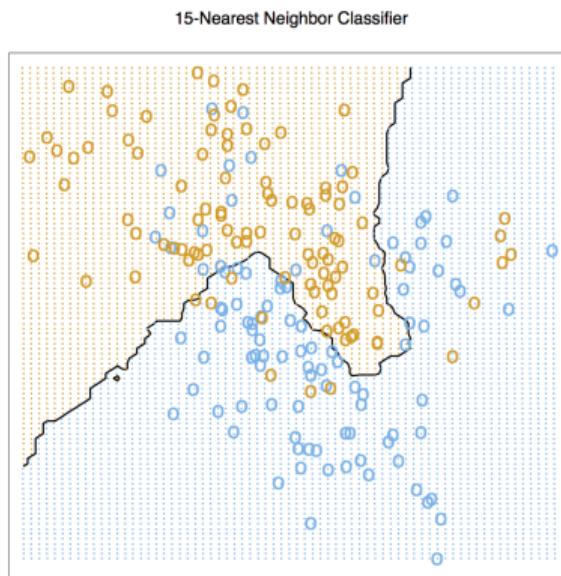


FIGURE 2.2. The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable (BLUE = 0, ORANGE = 1) and then fit by 15-nearest-neighbor averaging as in (2.8). The predicted class is hence chosen by majority vote amongst the 15-nearest neighbors.

Nearest Neighbor Methods

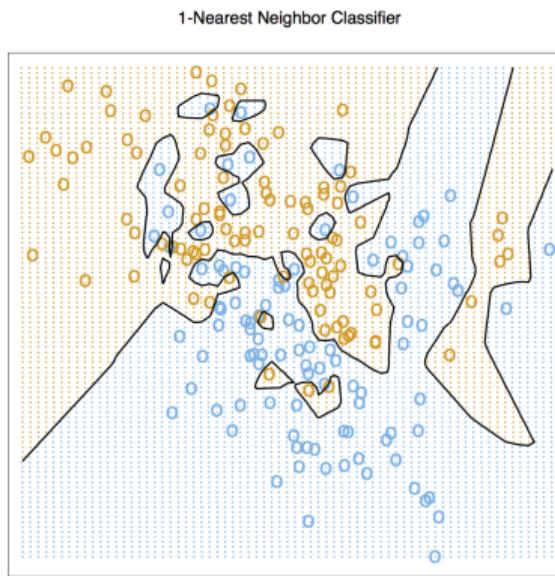


FIGURE 2.3. The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable (BLUE = 0, ORANGE = 1), and then predicted by 1-nearest-neighbor classification.

Nearest Centroid (Prototype) Classifier

- Assign the label of the class of training samples whose mean (centroid) is closest
- Training: compute the per-class centroids $\vec{\mu}_\ell = \frac{1}{|C_\ell|} \sum_{i \in C_\ell} \vec{x}_i$ where C_ℓ is the index set of samples belonging to class $\ell \in \mathbb{Y}$.
- Prediction: $\hat{y} = \arg \min_{\ell \in \mathbb{Y}} \|\vec{\mu}_\ell - \vec{x}\|$

Naive Bayes Classifier

- Bayes theorem (choose the class with the highest posterior density)

$$P(Y = k | X) = \frac{\pi_k P(X | Y = k)}{\sum_I \pi_I P(X | Y = I)}$$

where $\pi_k = P(Y \in C(k))$ is the class prior. (usually, $\hat{\pi}_k = \frac{n_k}{n}$)

- Gaussian Bayes Classifier:

$$x | Y = k \sim N(\mu_k, \sigma_k^2 \mathbf{I}_{p \times p})$$

- Parameter estimation:

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i \in C(k)} X_i$$

Linear Regression of an Indicator Matrix

- Categorical Y with K classes; $K \geq 2$
- One-hot encoding: Indicator variable $Y_k = 1$ if $Y = k$ for $k = 1, \dots, K$ (similar to dummy encoding)
- LM: for each class k , $f_k(x) = E(y_k | x) = \Pr(Y = k | x) = \beta_{0k} + \beta_k^T x$
- $\hat{Y}(x) = \arg \max_k \hat{f}_k(x)$
- Multivariate response (Y_1, \dots, Y_K)
- Decision boundary: $f_k(x) = f_l(x)$, linear
- Classes can be masked by others

Least Squares & Linear Regression

- Multiple outputs

$$\begin{aligned}Y_k &= \beta_{0k} + \sum_{j=1}^p X_j \beta_{jk} + \varepsilon_k \\ Y &= XB + E\end{aligned}$$

$Y = [Y_1, Y_2, \dots, Y_K]$, $B = [\beta_1, \beta_2, \dots, \beta_K]$, where $\beta_k = [\beta_{0k}, \dots, \beta_{pK}]^T$

- Loss function (when error is uncorrelated):

$$\begin{aligned}RSS(B) &= \sum_{k=1}^K \sum_{i=1}^N (Y - \beta_{0k} - \sum_{j=1}^p X_j \beta_{jk})^2 \\ &= \text{tr}[(Y - XB)^T (Y - XB)]\end{aligned}$$

Linear Regression of an Indicator Matrix

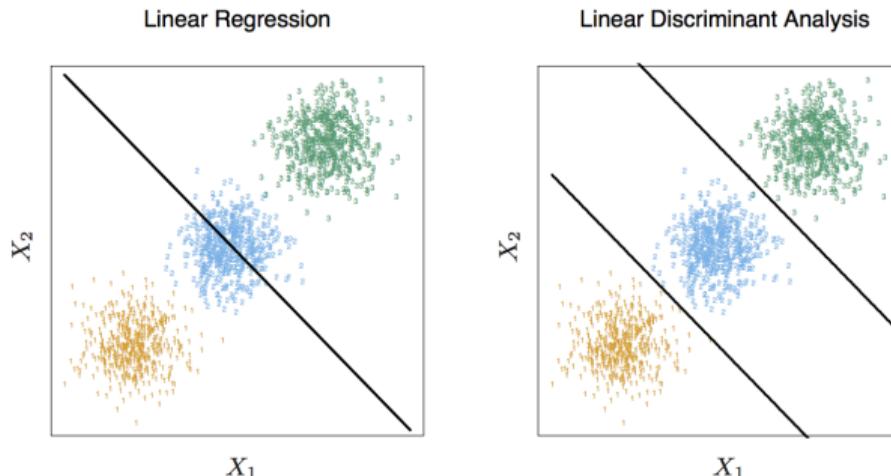


FIGURE 4.2. The data come from three classes in \mathbb{R}^2 and are easily separated by linear decision boundaries. The right plot shows the boundaries found by linear discriminant analysis. The left plot shows the boundaries found by linear regression of the indicator response variables. The middle class is completely masked (never dominates).

Linear Regression of an Indicator Matrix

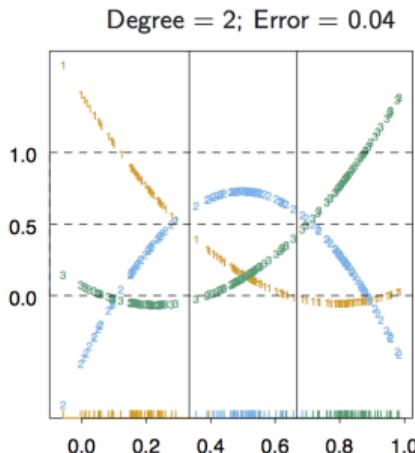
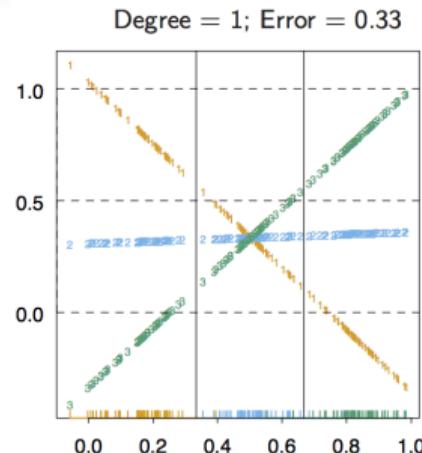


FIGURE 4.3. The effects of masking on linear regression in IR for a three-class problem. The rug plot at the base indicates the positions and class membership of each observation. The three curves in each panel are the fitted regressions to the three-class indicator variables; for example, for the blue class, y_{blue} is 1 for the blue observations, and 0 for the green and orange. The fits are linear and quadratic polynomials. Above each plot is the training error rate. The Bayes error rate is 0.025 for this problem, as is the LDA error rate.

Discriminant Analysis

- Optimal Bayes rule: $\arg \max_k \Pr(Y = k | X)$.

$$P(Y = k | X) = \frac{\pi_k P(X | Y = k)}{\sum_I \pi_I P(X | Y = I)} = \frac{\pi_k f_k(x)}{\sum_{I=1}^K \pi_I f_I(x)}$$

- Assume $f_k = MVN(\mu_k, \Sigma) \Rightarrow$ LDA.
- Assume $f_k = MVN(\mu_k, \Sigma_k) \Rightarrow$ QDA.
- FDA, assume homoscedasticity \Rightarrow LDA
- Estimate f_k nonparametrically, e.g. by kernel density estimation \Rightarrow KDA.
General, but not working well for large p - "curse of dim."
- Naive Bayes: assuming independence among the predictors, $f_k(x) = \prod_{j=1}^p f_{kj}(x_j)$.

Linear Discriminant Analysis

- Assume: $x | k \sim N(\mu_k, \Sigma)$.

$$\delta_k(x) \propto \log(\pi_k f_k(x)) \propto \log \pi_k + x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k$$

- $\hat{y} = \arg \max_k \Pr(k | x) = \arg \max_k \delta_k(x)$.
- In practice, estimate

$$\hat{\pi}_k = n_k / n,$$

$$\hat{\mu}_k = \sum_{G_i=k} x_i / n_k,$$

$$\hat{\Sigma} = \sum_{k=1}^K \sum_{G_i=k} (x_i - \hat{\mu}_k)^T (x_i - \hat{\mu}_k) / (N - K),$$

Linear Discriminant Analysis

- Log-ratio between two classes k and l

$$\begin{aligned} \log \frac{\Pr(k | x)}{\Pr(l | x)} &= \delta_k(x) - \delta_l(x) \\ &= \log \frac{\pi_k}{\pi_l} - \frac{1}{2} (\mu_k + \mu_l)^T \Sigma^{-1} (\mu_k - \mu_l) + x^T \Sigma^{-1} (\mu_k - \mu_l) \\ &= \beta_{0,kl} + x^T \beta_{kl}, \end{aligned}$$

- Linear decision boundary
- when $K = 2$,

- LDA rule:

$$x^T \hat{\Sigma}^{-1} (\hat{\mu}_2 - \hat{\mu}_1) > \frac{1}{2} \hat{\mu}_2^T \hat{\Sigma}^{-1} \hat{\mu}_2 - \frac{1}{2} \hat{\mu}_1^T \hat{\Sigma}^{-1} \hat{\mu}_1 + \log(N_1/N) - \log(N_2/N)$$

- Code $y = -N/N1, N/N2$, LM ($E(y) = b_0 + x^T b$) rule: $\hat{b} \propto \hat{\Sigma}^{-1} (\hat{\mu}_2 - \hat{\mu}_1)$ (Ex. 4.2)

Quadratic Discriminant Analysis

- Assume: $x | k \sim N(\mu_k, \Sigma_k)$.

$$\delta_k(x) \propto \log \pi_k - \frac{1}{2} \log |\Sigma_k| - \frac{1}{2} (x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k).$$

quadratic, or

$$\log \frac{\Pr(k | x)}{\Pr(I | x)} = \beta_{0,kI} + x^T \beta_{1,kI} + x^T B_{2,kI} x.$$

quadratic logit model.

- LDA vs. QDA: much larger number of parameters; $(K - 1) \times (p + 1)$ vs. $(K - 1) \times \{p(p + 3)/2 + 1\}$

Quadratic Discriminant Analysis

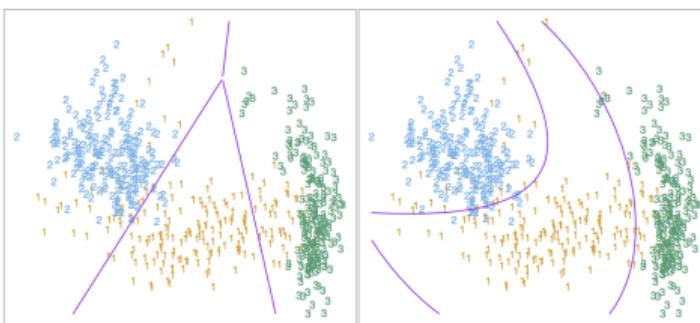


FIGURE 4.1. The left plot shows some data from three classes, with linear decision boundaries found by linear discriminant analysis. The right plot shows quadratic decision boundaries. These were obtained by finding linear boundaries in the five-dimensional space $X_1, X_2, X_1X_2, X_1^2, X_2^2$. Linear inequalities in this space are quadratic inequalities in the original space.

Quadratic Discriminant Analysis

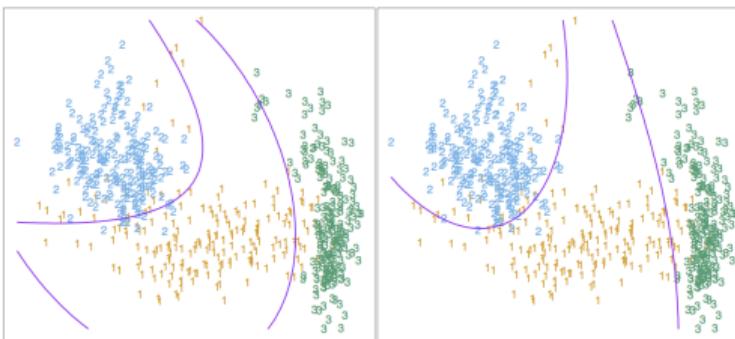


FIGURE 4.6. Two methods for fitting quadratic boundaries. The left plot shows the quadratic decision boundaries for the data in Figure 4.1 (obtained using LDA in the five-dimensional space $X_1, X_2, X_1X_2, X_1^2, X_2^2$). The right plot shows the quadratic decision boundaries found by QDA. The differences are small, as is usually the case.

Regularized Discriminant Analysis

- A compromise between LDA and QDA: use $\tilde{\Sigma}_k(\alpha)$ in QDA,

$$\tilde{\Sigma}_k(\alpha) = \alpha \hat{\Sigma}_k + (1 - \alpha) \hat{\Sigma}$$

where $\alpha \in [0, 1]$

- Shrinkage: Replace $\hat{\Sigma}$ with $\hat{\Sigma}(\gamma) = \gamma \hat{\Sigma} + (1 - \gamma) \hat{\sigma}^2 I$,

Logistic Regression

- Assume $y_i \sim \text{Bernoulli}(p)$

$$f(y_i) = p_i^{y_i} (1 - p_i)^{1-y_i} \text{ where } y_i \in \{0, 1\}$$

$$E[y_i] = p_i \stackrel{\text{set}}{\in} [0, 1] \neq X_i^T \beta \in (-\infty, \infty)$$

- A link function: logit function $g : [0, 1] \rightarrow \mathbb{R}$:

$$\text{logit}(p_i) = \log \left(\frac{p_i}{1 - p_i} \right) \stackrel{\text{set}}{=} X_i^T \beta \text{ where } p_i \in (0, 1)$$

- Logistic function

$$p_i = g^{-1}(X_i^T) = \frac{e^{X_i^T \beta}}{1 + e^{X_i^T \beta}}$$

Logistic Regression

- Likelihood function:

$$\begin{aligned} L(\beta) &= \prod_{i=1}^n p_i^{y_i} (1 - p_i)^{1-y_i} \\ &= \prod_{i=1}^n \left[\frac{e^{X_i^T \beta}}{1 + e^{X_i^T \beta}} \right]^{y_i} \left[1 - \frac{e^{X_i^T \beta}}{1 + e^{X_i^T \beta}} \right]^{(1-y_i)} \end{aligned}$$

- Log-likelihood

$$\ell(\beta) = \sum_i y_i X_i^T \beta - \log(1 + \exp(X_i \beta))$$

- Penalized logistic regression:

$$-\ell(\beta) + \lambda P(\beta)$$

Logistic Regression

- The derivative of log-likelihood (score function)

$$\frac{\partial I}{\partial \beta} = \sum_i x_i \left(y_i - \frac{e^{x_i^T \beta}}{1 + e^{x_i^T \beta}} \right) \stackrel{\text{set}}{=} 0$$

- Newton method

$$\beta^{\text{new}} = \beta^{\text{old}} - \left[\frac{\partial^2 I}{\partial \beta^2} (\beta^{\text{old}}) \right]^{-1} \frac{\partial I}{\partial \beta} (\beta^{\text{old}})$$

- Estimation of penalized logistic regression: proximal gradient descent

Logistic Regression

- Wald test $H_0 : \beta_j = 0$

$$z = \frac{\hat{\beta}_j}{SE(\hat{\beta}_j)} \xrightarrow{D} N(0, 1)$$

- Likelihood ratio test $H_0 : \beta_q = \beta_{q+1} = \dots = \beta_p = 0$. Test the goodness of fit between the reduced model and the full model.

$$\lambda = \frac{L(R)}{L(F)}$$

$$-2 \log \lambda \xrightarrow{D} \chi^2_{df_{full} - df_{reduced}}$$

Multi-class Logistic Regression

- Multiple classes: $y_i \in \{1, \dots, K\}$, choose a basis class K (can be an arbitrary class)

$$\log \frac{P(Y = k | X = x)}{P(Y = K | X = x)} = \beta_k^T X, k = 1, \dots, K - 1$$

- As a set of independent binary regressions

$$P(Y = k | X = x) = \frac{e^{\beta_k^T X}}{1 + \sum_{i=1}^{k-1} e^{\beta_i^T X}}, k = 1, \dots, K - 1$$

$$P(Y = K | X = x) = \frac{1}{1 + \sum_{i=1}^{k-1} e^{\beta_i^T X}}$$

- Log linear model / softmax function

$$\Pr(Y_i = k) = \frac{e^{\beta_k \cdot \mathbf{x}_i}}{\sum_{j=1}^K e^{\beta_j \cdot \mathbf{x}_i}}$$

Support Vector Machines

Here we approach the two-class classification problem in a direct way:

We try and find a plane that separates the classes in feature space.

If we cannot, we get creative in two ways:

- We soften what we mean by “separates”, and
- We enrich and enlarge the feature space so that separation is possible.

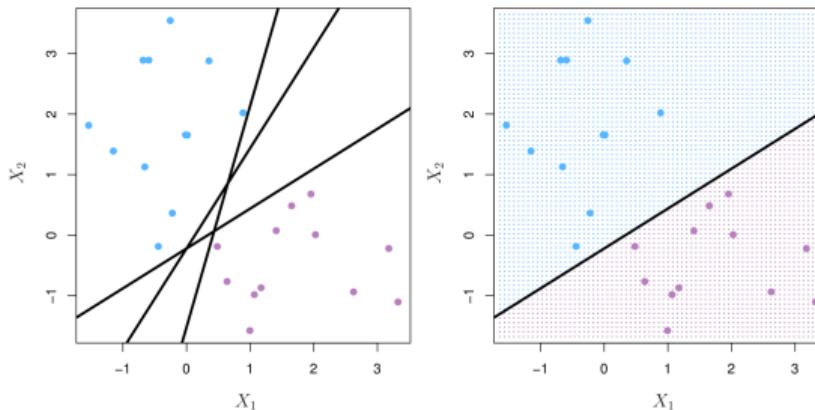
What is a Hyperplane?

- A hyperplane in p dimensions is a flat subspace of dimension $p - 1$.
- In general the equation for a hyperplane has the form

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

- In $p = 2$ dimensions a hyperplane is a line.
- If $\beta_0 = 0$, the hyperplane goes through the origin, otherwise not.
- The vector $\beta = (\beta_1, \beta_2, \dots, \beta_p)$ is called the normal vector — it points in a direction orthogonal to the surface of a hyperplane.

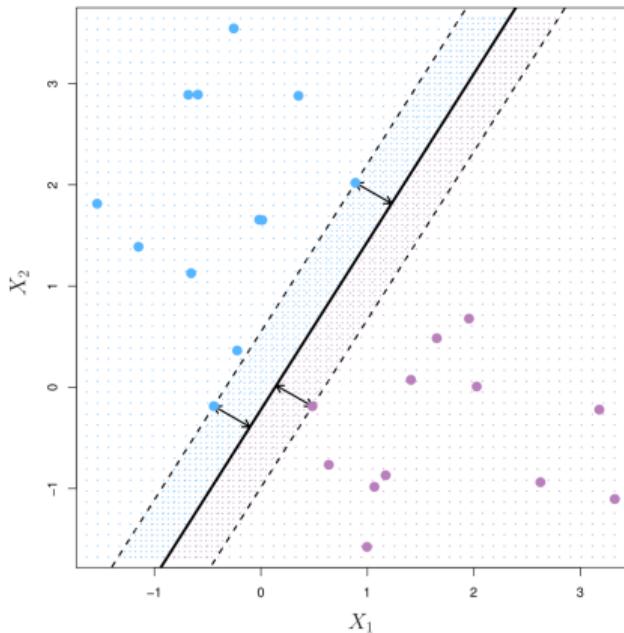
Separating Hyperplanes



- If $f(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$, then $f(X) > 0$ for points on one side of the hyperplane, and $f(X) < 0$ for points on the other.
- If we code the colored points as $Y_i = +1$ for blue, say, and $Y_i = -1$ for purple, then if $Y_i \cdot f(X_i) > 0$ for all i , $f(X) = 0$ defines a **separating hyperplane**.

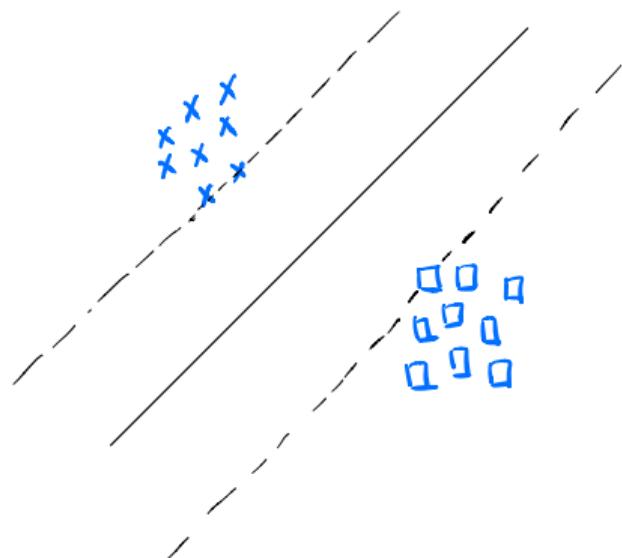
Maximal Margin Classifier

Among all separating hyperplanes, find the one that makes the biggest gap or **margin** between the two classes.



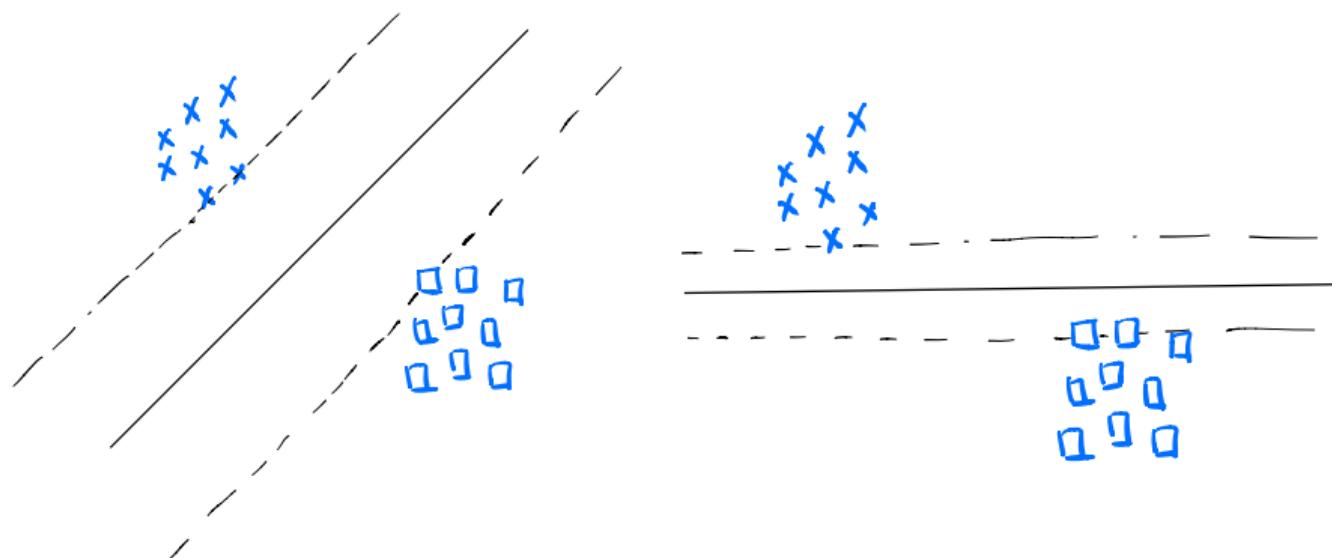
Why biggest margin?

Training



Why biggest margin?

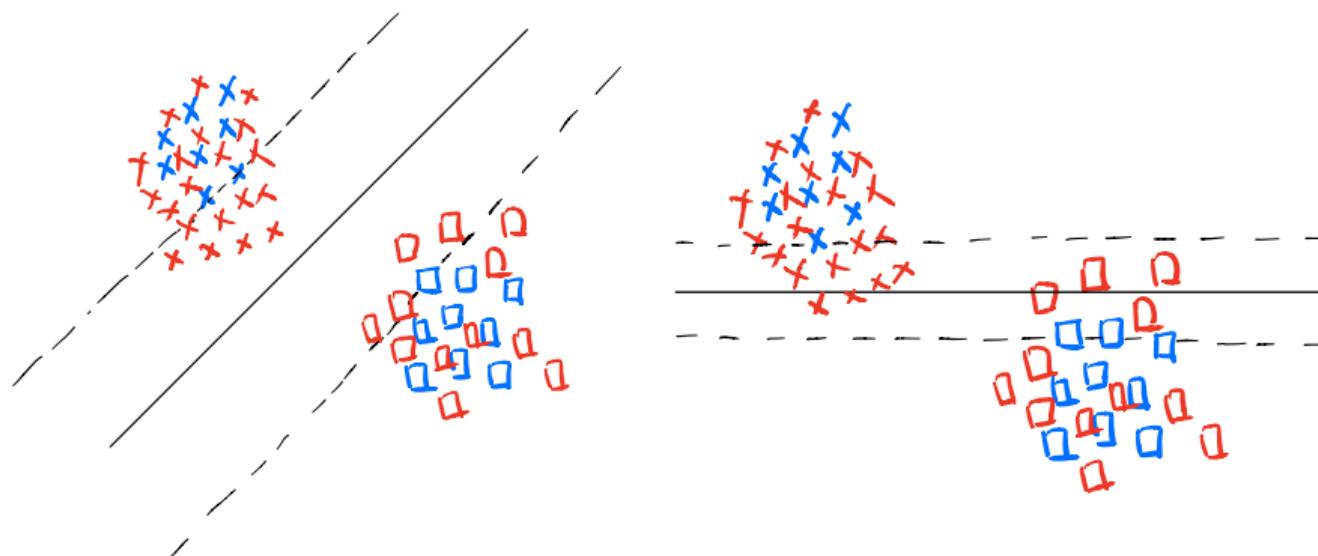
Training



makes no obvious difference.

Why biggest margin?

Testing



Wide margin is clearly better.

Constrained optimization problem

$$\text{maximize}_{\beta_0, \beta_1, \dots, \beta_p} M$$

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

$$y_i(\beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip}) \geq M \quad \text{for all } i = 1, \dots, N.$$

This can be rephrased as a convex quadratic program, and solved efficiently. The function `svm()` in package `e1071` solves this problem efficiently.

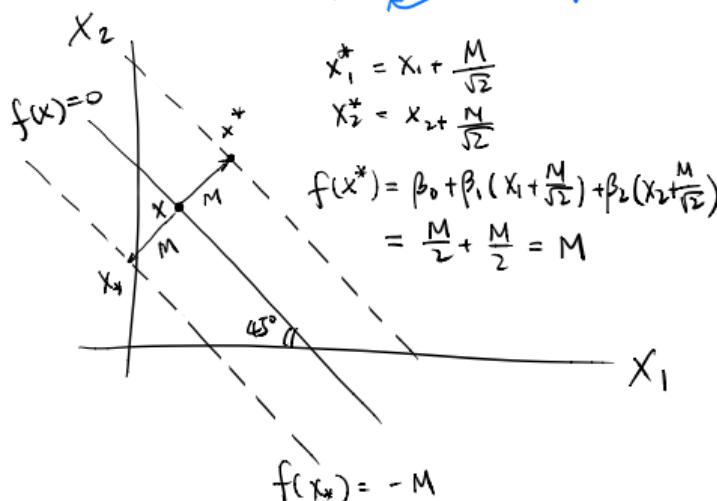
Constraint explained

E.g. $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$

$$\beta_0 = -\frac{\sqrt{2}}{2} \quad \beta_1 = \beta_2 = \frac{\sqrt{2}}{2}$$

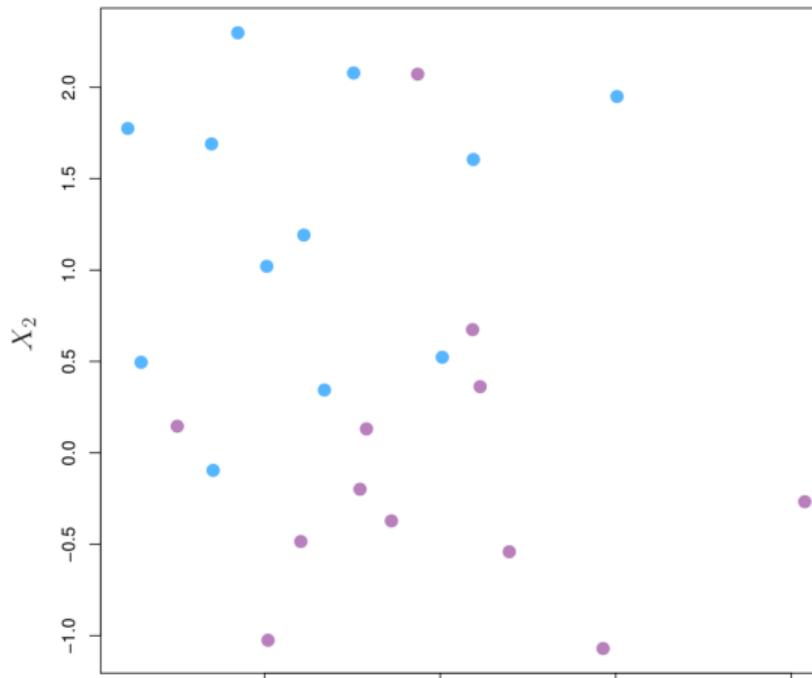
$$\beta_1^2 + \beta_2^2 = 1$$

unit length

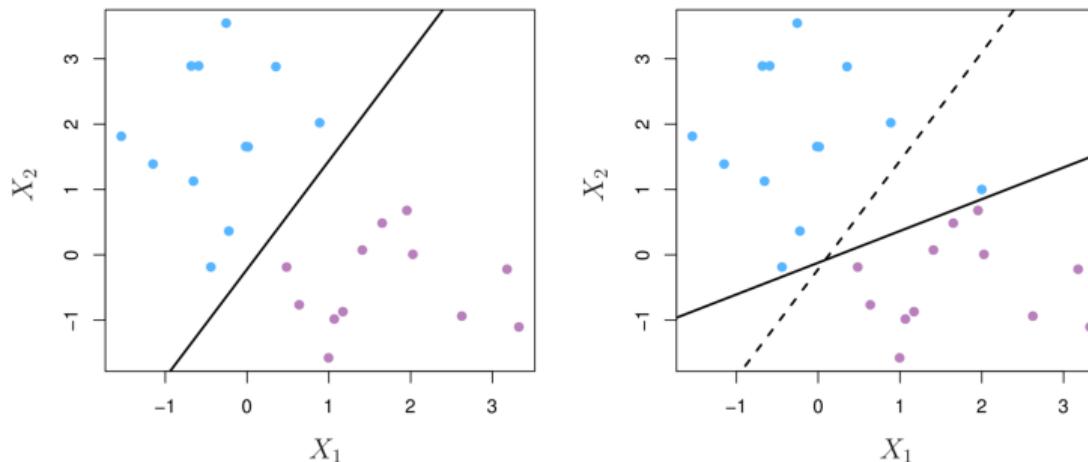


Non-separable Data

The data on the left are not separable by a linear boundary.
This is often the case when $n > p$.



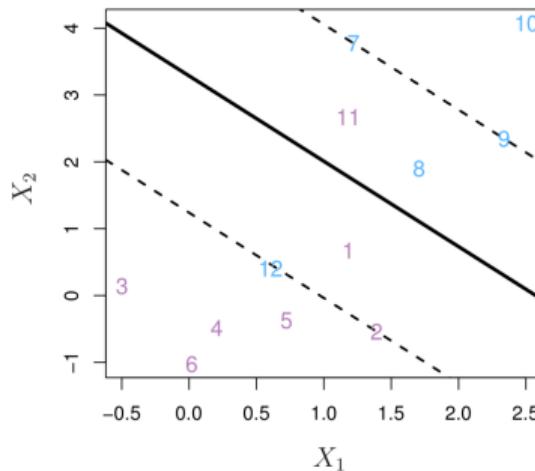
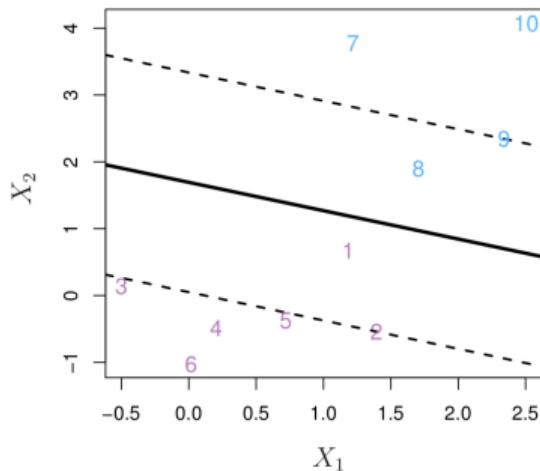
Noisy Data



Sometimes the data are separable, but noisy. This can lead to a poor solution for the maximal-margin classifier.

The **support vector classifier** maximizes a **soft** margin.

Support Vector Classifier



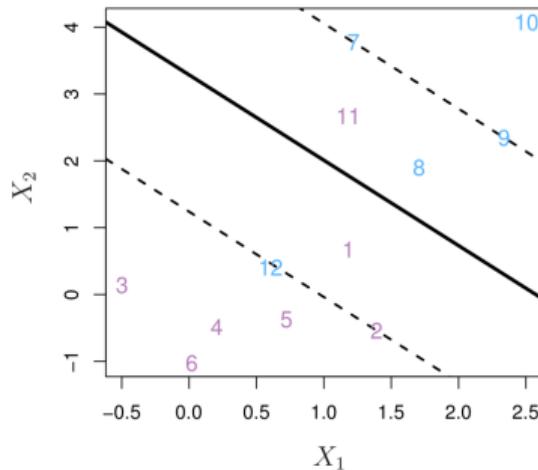
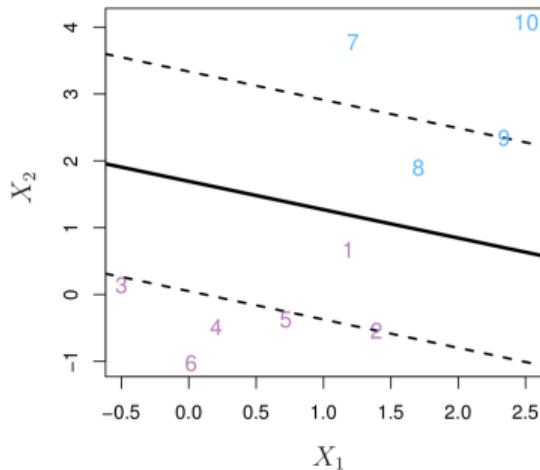
For each point on the wrong side of the dashed line, we pay a “price”. And we have a total budget of C to spend. The price is proportional to the distance from the point to the dashed line.

For each point on the wrong side of the dashed line, we pay a “price”. And we have a total budget of C to spend. The price is proportional to the distance from the point to the dashed line.

$$\begin{aligned} & \text{maximize}_{\beta_0, \beta_1, \dots, \beta_p, \epsilon_1, \dots, \epsilon_n} M \quad \text{subject to} \sum_{j=1}^p \beta_j^2 = 1, \\ & y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip}) \geq M(1 - \epsilon_i), \end{aligned}$$

$$\epsilon_i \geq 0, \sum_{i=1}^n \epsilon_i \leq C.$$

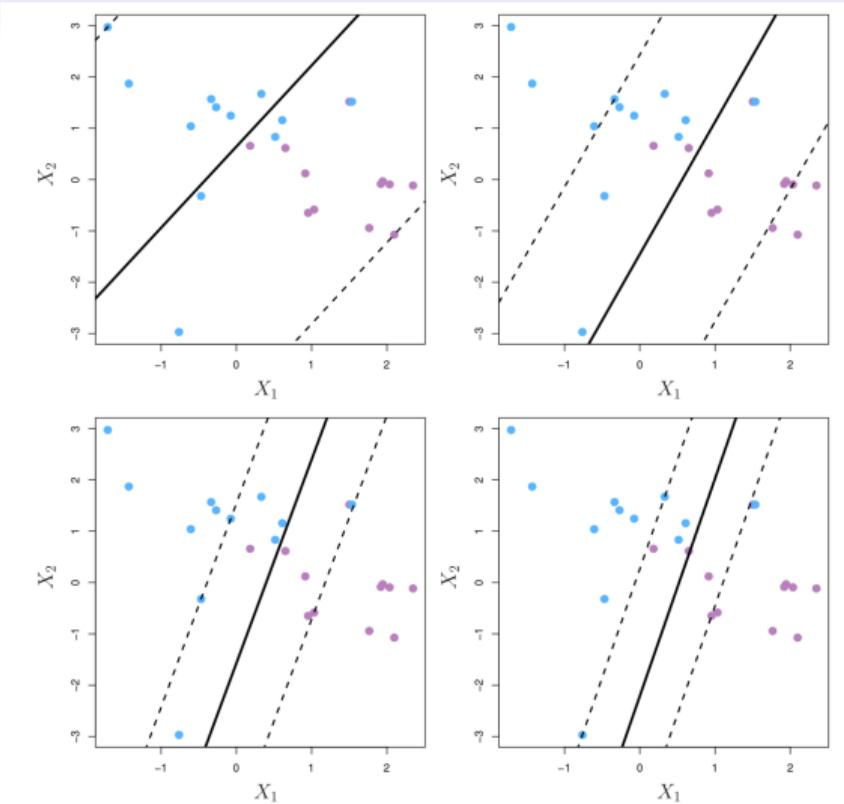
Support Vector Classifier



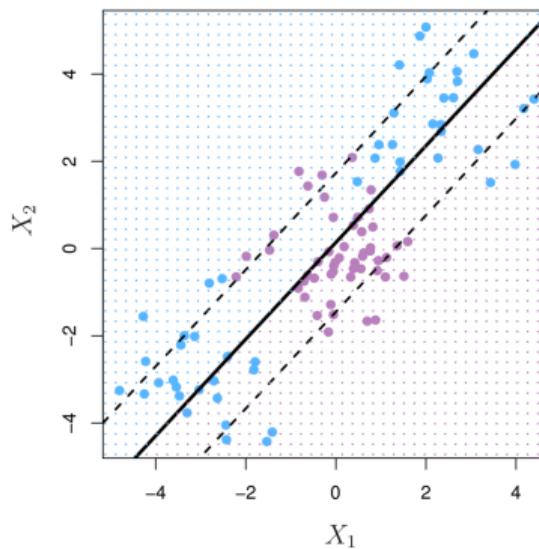
Observations that lie directly on the margin, or on the wrong side of the margin for their class, are known as **support vectors**.

Left: 1,2,8,9 are support vectors. Right: 1,2,7,8,9,11,12 are support vectors

C is a regularization parameter

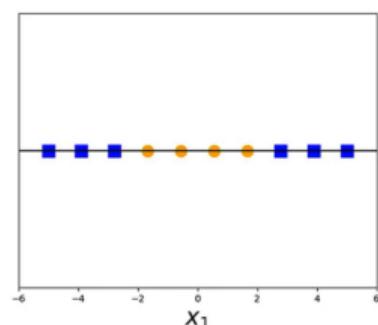


Linear boundary can fail

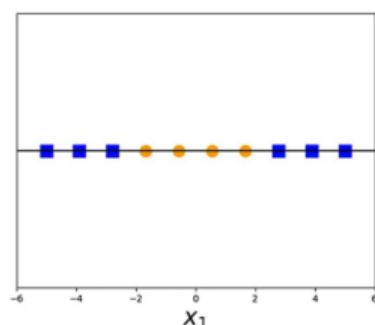


Sometimes a linear boundary simply won't work, no matter what value of C .
What to do?

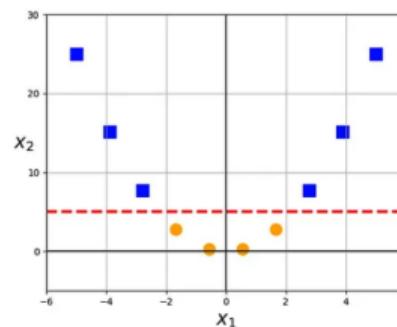
One-dimensional Example



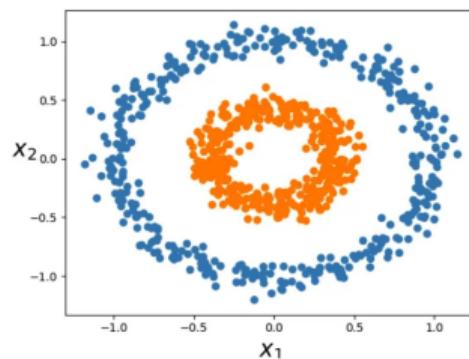
One-dimensional Example



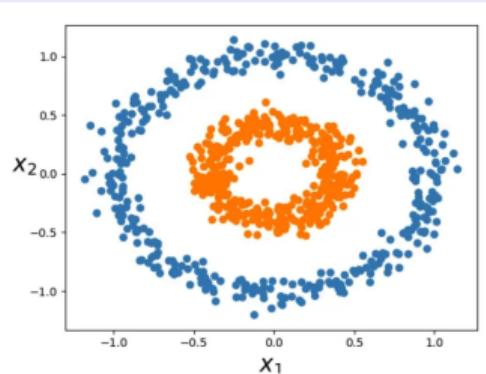
$$x \rightarrow (x, x^2)$$



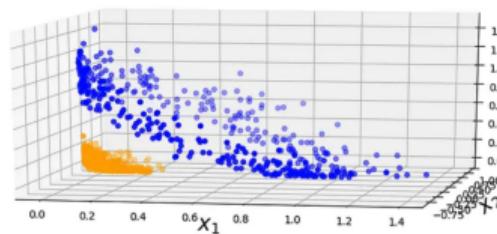
Two-dimensional Example



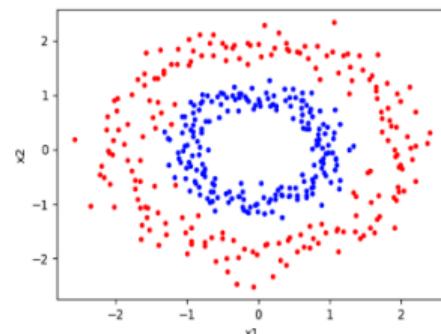
Two-dimensional Example



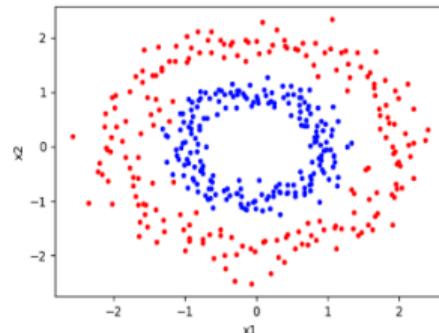
$$x_1, x_2 \rightarrow (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$



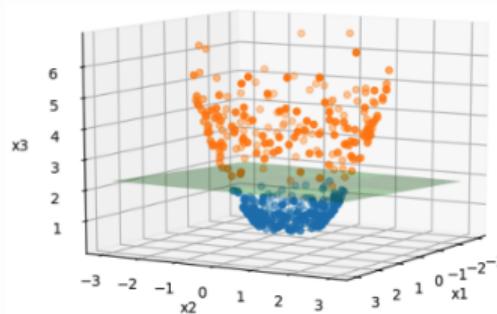
Two-dimensional Example



Two-dimensional Example



$$x_1, x_2 \rightarrow (x_1, x_2, x_1^2 + x_2^2)$$



Feature Expansion

- Enlarge the space of features by including transformations; e.g. $X_1^2, X_1^3, X_1X_2, X_1X_2^2, \dots$. Hence go from a p -dimensional space to a $q > p$ dimensional space.
- Fit a support-vector classifier in the enlarged space.
- This results in non-linear decision boundaries in the original space.

Example: Suppose we use $(X_1, X_2, X_1^2, X_2^2, X_1X_2)$ instead of just (X_1, X_2) . Then the decision boundary would be of the form

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + \beta_4 X_2^2 + \beta_5 X_1 X_2 = 0.$$

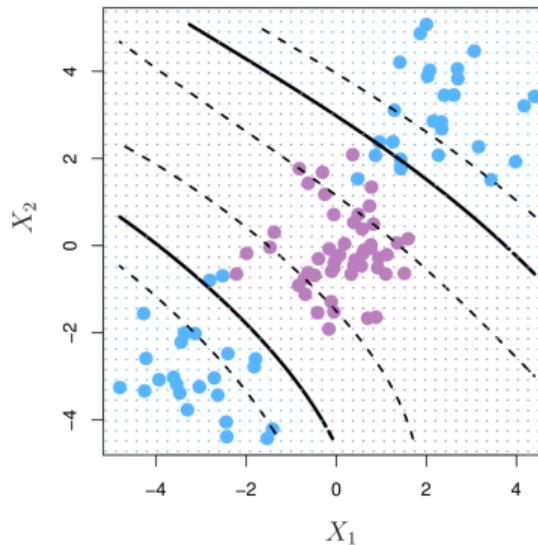
This leads to nonlinear decision boundaries in the original space.

Cubic Polynomials

Here we use a basis expansion of cubic polynomials

From 2 variables to 9

The support-vector classifier in the enlarged space solves the problem in the lower-dimensional space.



Nonlinearities and Kernels

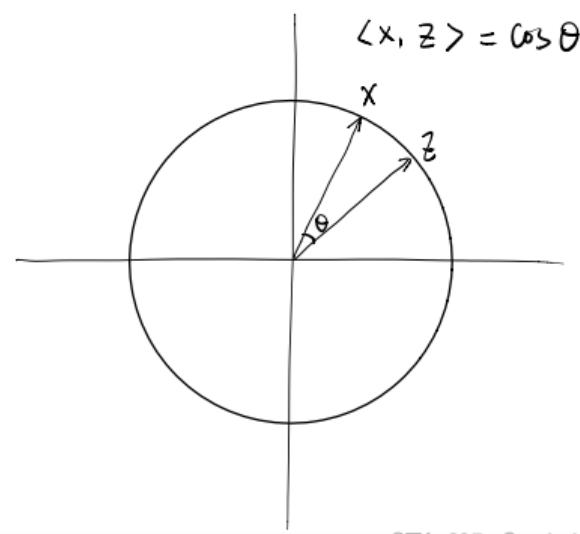
- Polynomials are hard to compute even for moderate degree when p is large.
- There is a more elegant and controlled way to introduce nonlinearities in support-vector classifiers — through the use of **kernels**.
 - Computationally, only need to compute $\binom{n}{2}$ inner products of p -dimensional vectors. Details later.
 - Choosing kernel is not an easy task.
- Before we discuss these, we must understand the role of **inner products** in support-vector classifiers.

Inner products

Inner (dot) product between vectors

$$\langle x, z \rangle = \sum_{j=1}^p x_j z_j$$

Inner product measures the similarity of two vectors.



Inner products and support vectors

- The solution to the linear support vector classifier can be written as

$$\hat{f}(x) = \hat{\beta}_0 + \sum_{i=1}^n \hat{\alpha}_i \langle x, x_i \rangle$$

- To estimate the parameters $\alpha_1, \dots, \alpha_n$ and β_0 , all we need are the $\binom{n}{2}$ inner products $\langle x_i, x_j \rangle$ for all pairs of training observations.
- It turns out that α_i is nonzero only for the support vectors.

$$\hat{f}(x) = \hat{\beta}_0 + \sum_{i \in S} \hat{\alpha}_i \langle x, x_i \rangle$$

S is the collection of indices of these support vectors.

Kernels and Support Vector Machines

- We can replace the inner product by other similarity measures.
- Kernel functions can do this for us. E.g. polynomial kernel

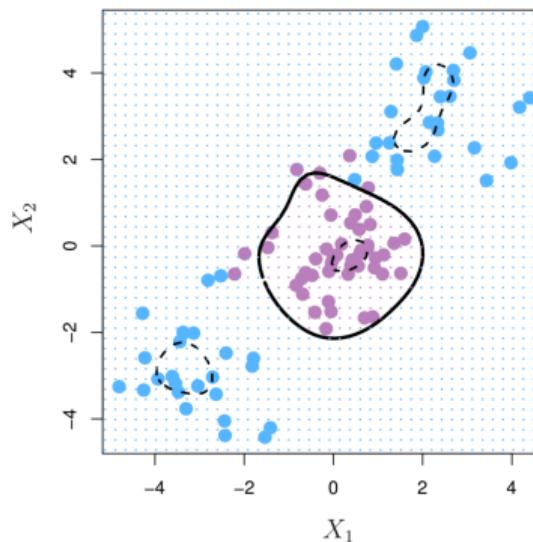
$$K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^p x_{ij}x_{i'j}\right)^d$$

- The solution has the form

$$\hat{f}(x) = \hat{\beta}_0 + \sum_{i \in S} \hat{\alpha}_i K(x, x_i).$$

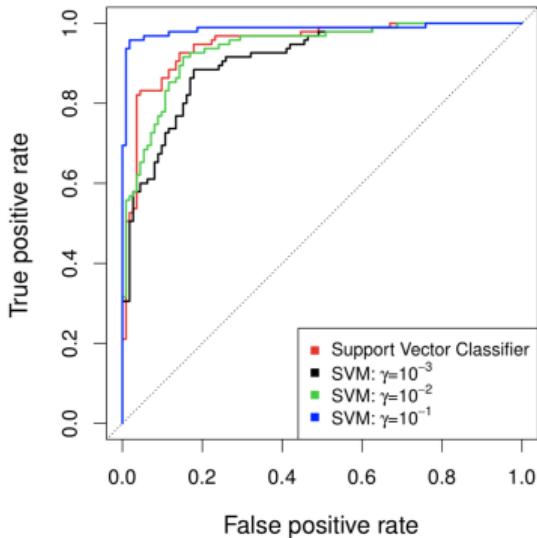
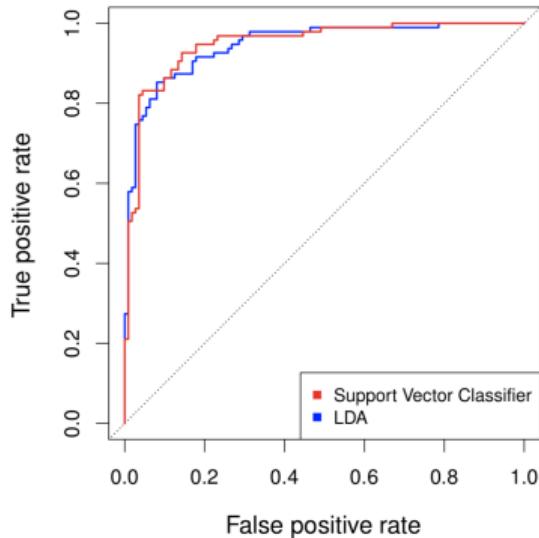
Radial Kernel

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



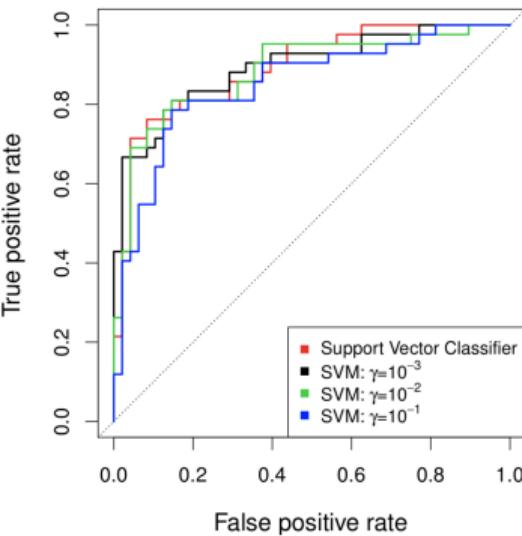
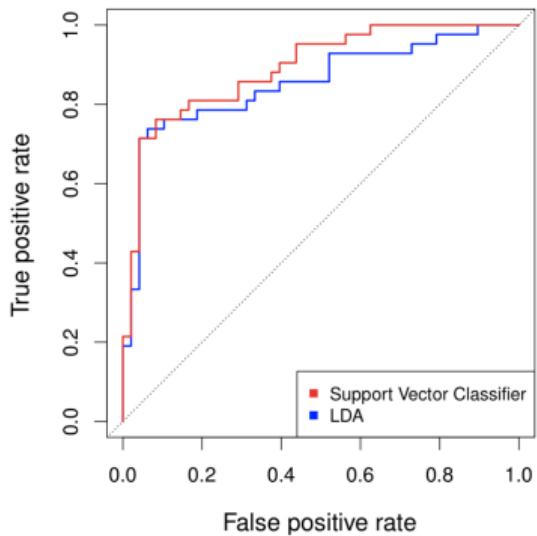
Larger γ leads to more flexible decision boundary.

Example: Heart Data



ROC curve is obtained by changing the threshold 0 to threshold t in $\hat{f}(X) > t$, and recording **false positive** and **true positive** rates as t varies. Here we see ROC curves on training data.

Example continued: Heart Test Data



SVMs: more than 2 classes?

The SVM as defined works for $K = 2$ classes. What do we do if we have $K > 2$ classes?

- **OVA** One versus All. Fit $K <$ different 2-class SVM classifiers $\hat{f}_k(x)$, $k = 1, \dots, K$; each class versus the rest. Classify x^* to the class for which $\hat{f}_k(x^*)$ is largest.
- **OVO** One versus One. Fit all $\binom{K}{2}$ pairwise classifiers $\hat{f}_k(x)$. Classify x^* to the class that wins the most pairwise competitions.

Which to choose? If K is not too large, use OVO.

Which to use: SVM or Logistic Regression or LDA

- When classes are (nearly) separable, SVM and LDA are better than LR.
- When not, LR, LDA and SVM very similar.
- If you wish to estimate probabilities, use LR or LDA.
- For nonlinear boundaries, kernel SVMs are popular. Can use kernels with LR and LDA as well, but computations are more expensive.
- When sample size is large, kernel SVMs can be very slow

Tree-based Methods

- Here we describe tree-based methods for regression and classification.
- These involve **stratifying** or **segmenting** the predictor space into a number of simple regions.
- Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as **tree-based** or **decision-tree** methods.

Pros and Cons

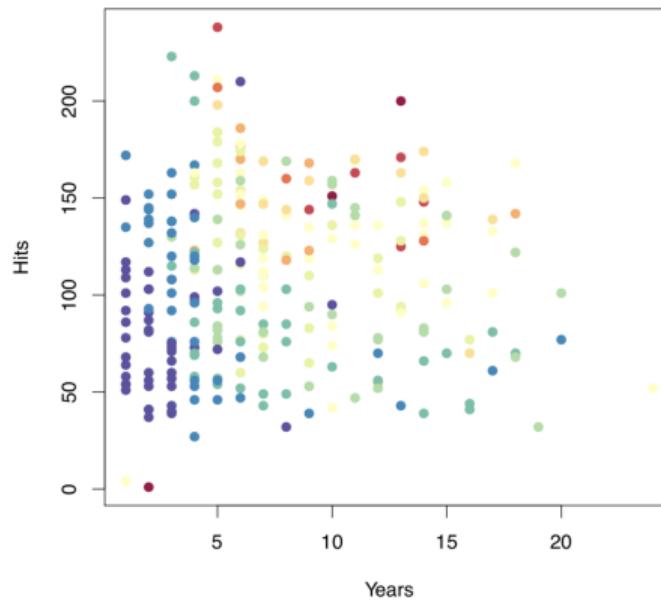
- Tree-based methods are simple and useful for interpretation.
- However they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy.
- Hence we will also discuss **bagging**, **random forests**, and **boosting** in the topic of *Ensemble Learning*. These methods grow multiple trees which are then combined to yield a single consensus prediction.
- Combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss of interpretation.

The Basics of Decision Trees

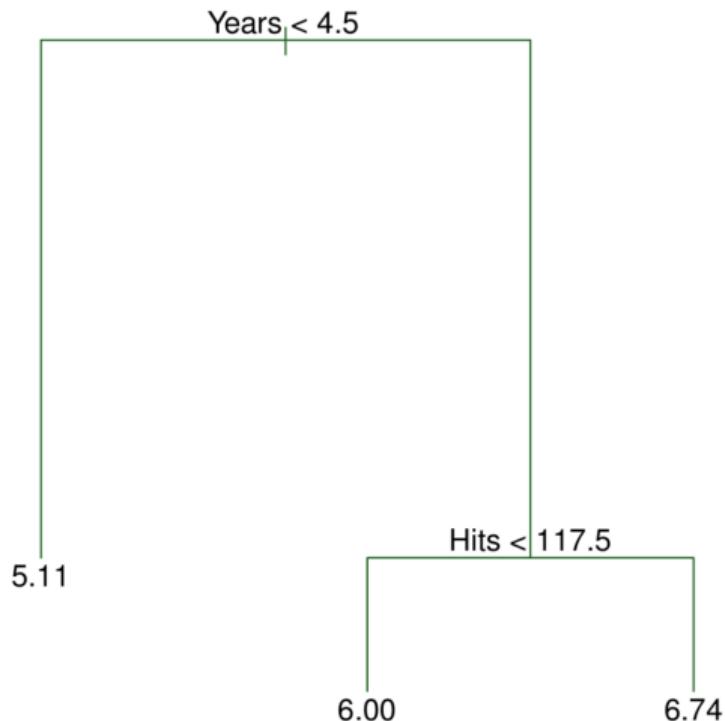
- Decision trees can be applied to both regression and classification problems.
- We first consider regression problems, and then move on to classification.

Baseball salary data: how would you stratify it?

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



Decision tree for these data

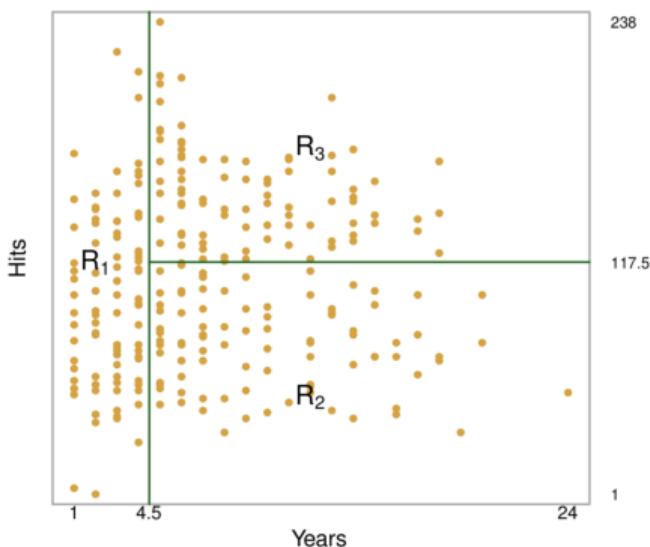


Details of the tree

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

Results

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



Terminology for Trees

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

Interpretation of Results

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

Details of the tree-building process

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

More details of the tree-building process

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

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

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

More details of the tree-building process

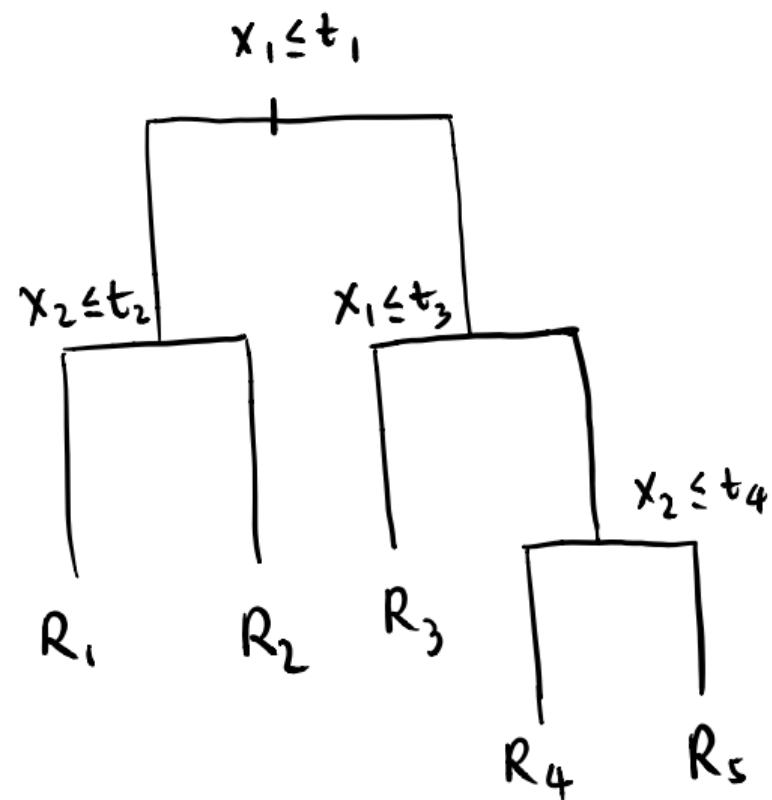
- Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a **top-down, greedy** approach that is known as recursive binary splitting.
- The approach is **top-down** because it begins at the top of the tree and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

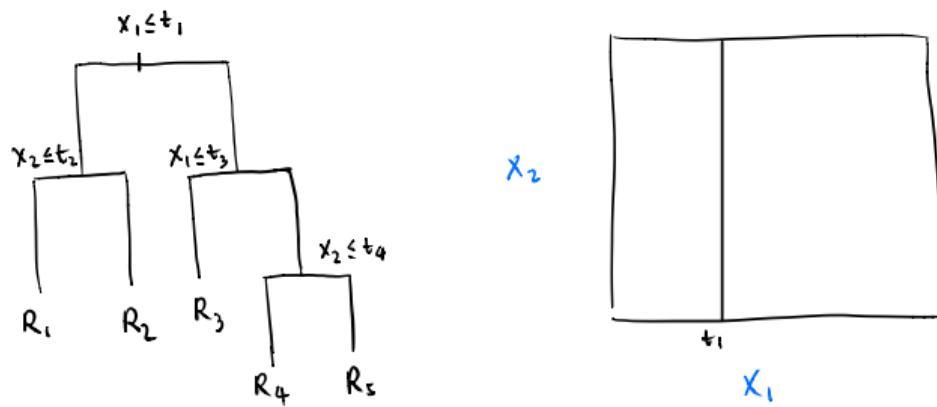
Details– Continued

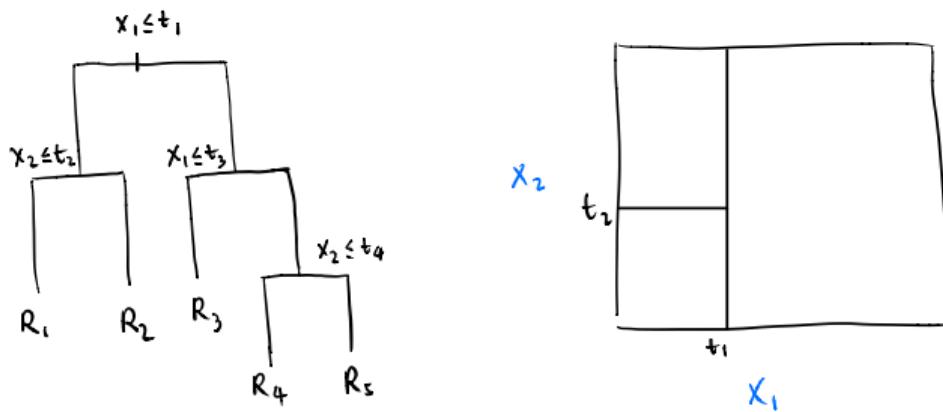
- We first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS.
- Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the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

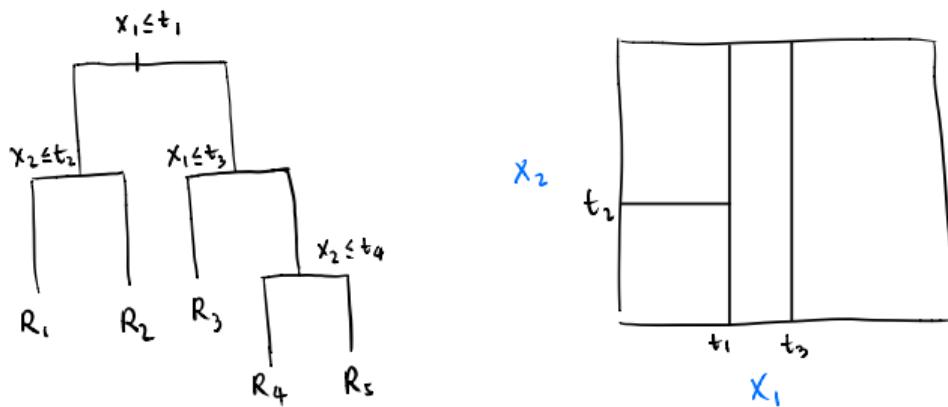
Predictions

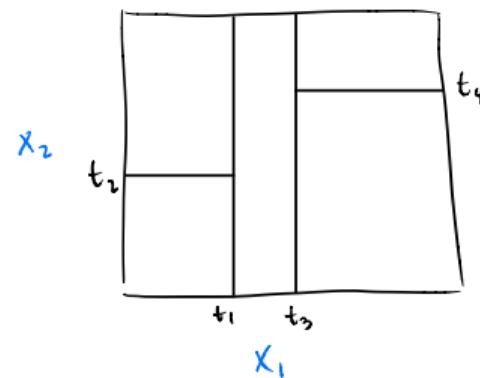
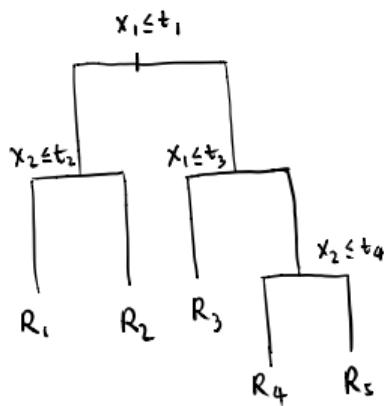
- We predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.
- A five-region example of this approach is shown next.

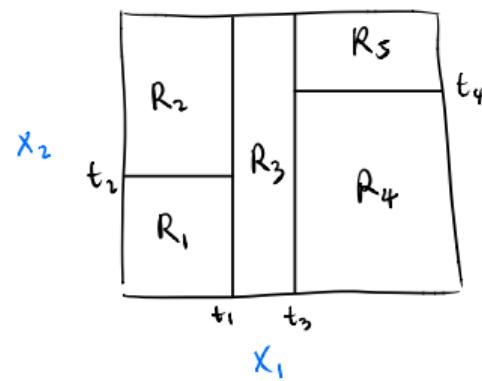
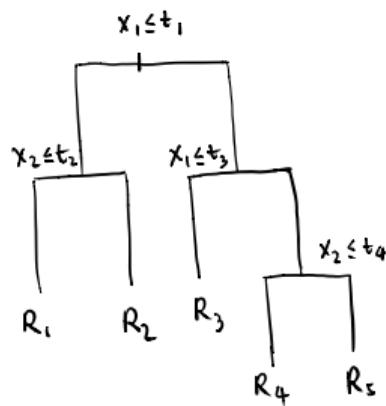




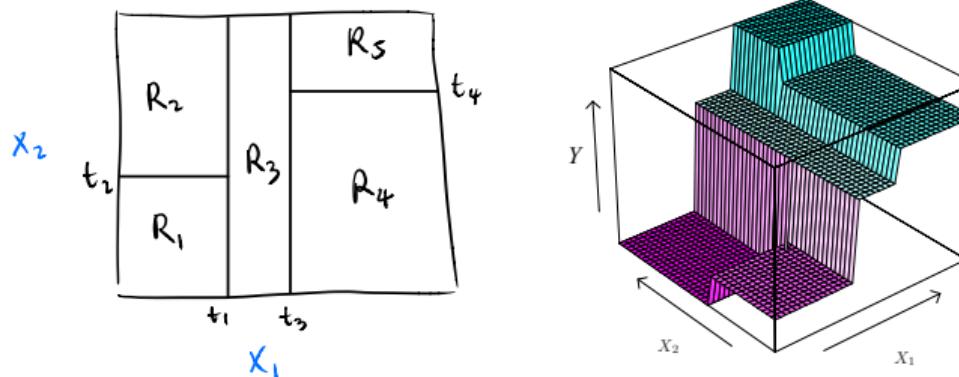








Prediction



To predict Y^* at $X^* = (X_1^*, X_2^*)$:

1. Find the region that X^* belongs to according to the tree or partition.
2. Suppose the region is R_j . Let $Y^* = \text{Ave}_{i \in R_j}(y_i)$.

Pruning a tree

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

Pruning a tree– continued

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

$$\min_{T \subset T_0} \sum_{j=1}^{J_T} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2 + \alpha J_T$$

Here J_T is the number of regions or terminal nodes of the tree T , R_j is the rectangle (i.e. the subset of predictor space) corresponding to the j th terminal node, and \hat{y}_{R_j} is the mean of the training observations in R_j .

Choosing the best subtree

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
- We select an optimal value $\hat{\alpha}$ using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$.

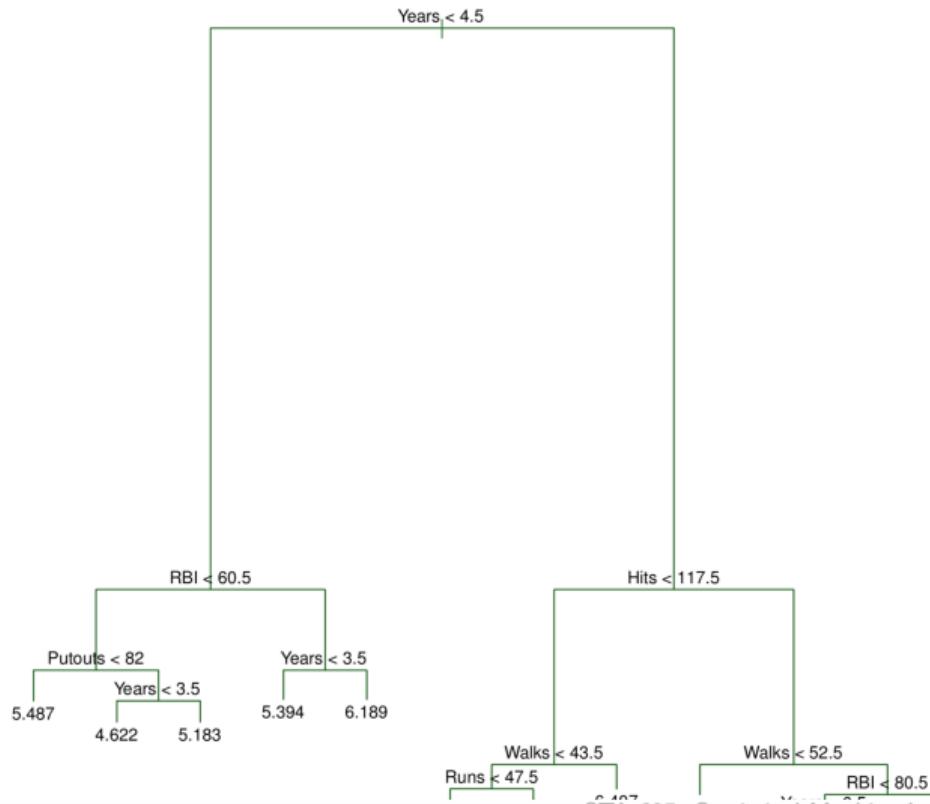
Summary: tree algorithm

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

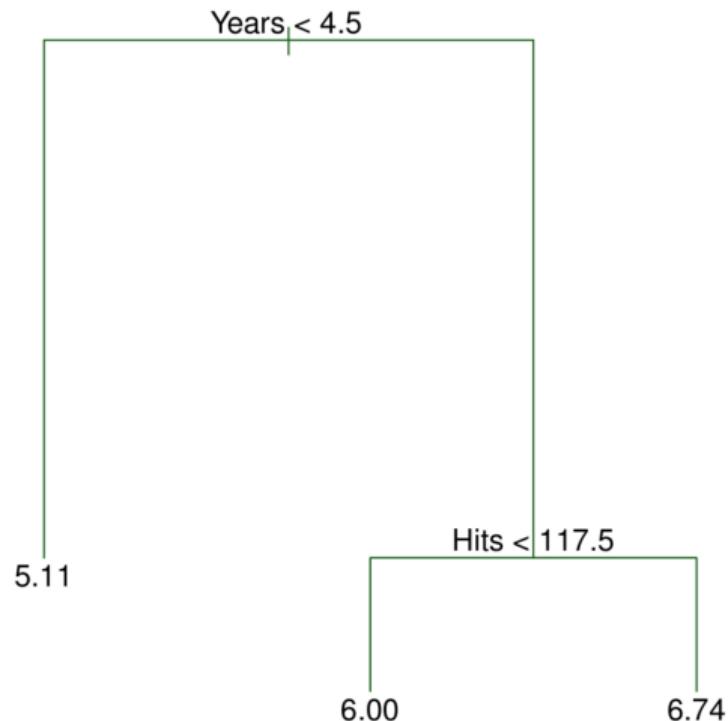
Baseball example continued

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

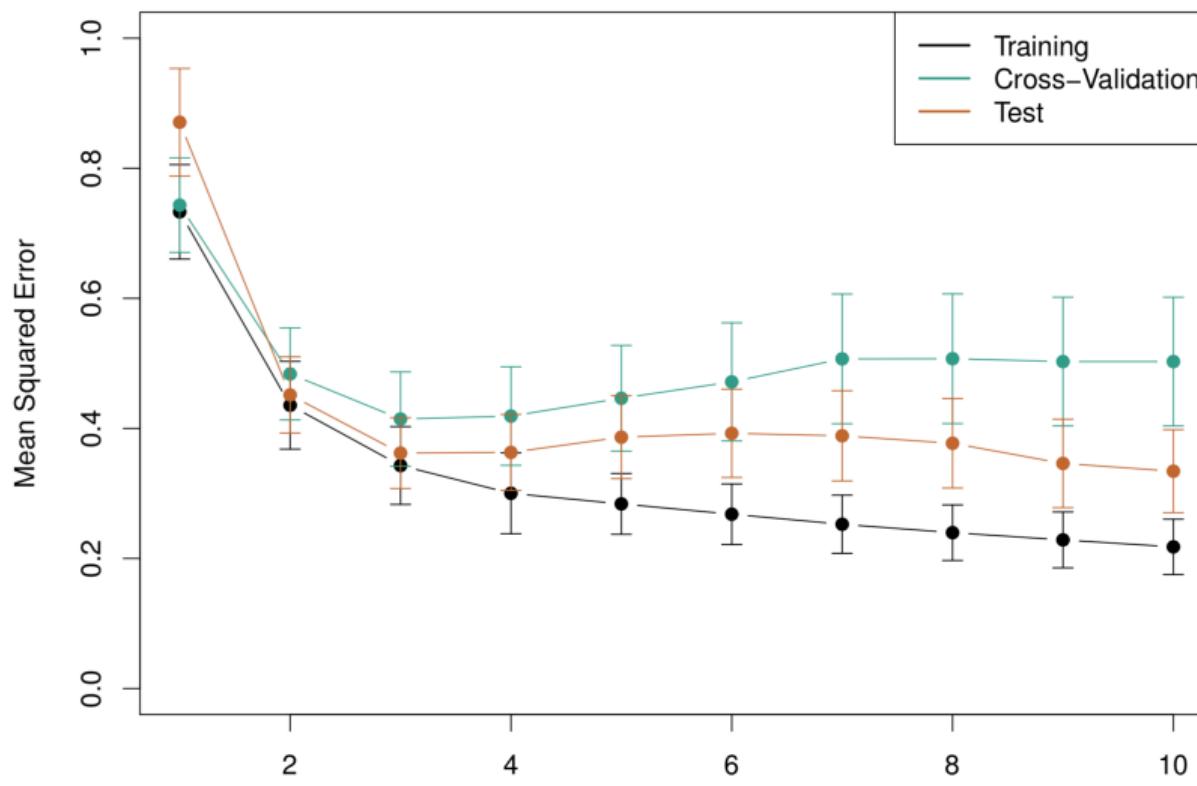
Unpruned tree



Pruned tree



Baseball example continued



Classification Trees

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

Details of classification trees

- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- In the classification setting, RSS cannot be used as a criterion for making the binary splits
- A natural alternative to RSS is the **classification error rate**. This is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E_j = 1 - \max_k(\hat{p}_{jk}).$$

Here \hat{p}_{jk} represents the proportion of training observations in the j th region that are from the k th class.

- However classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable. Will explain later.

Gini index and Deviance

- The **Gini index** is defined by

$$G_j = \sum_{k=1}^K \hat{p}_{jk}(1 - \hat{p}_{jk}),$$

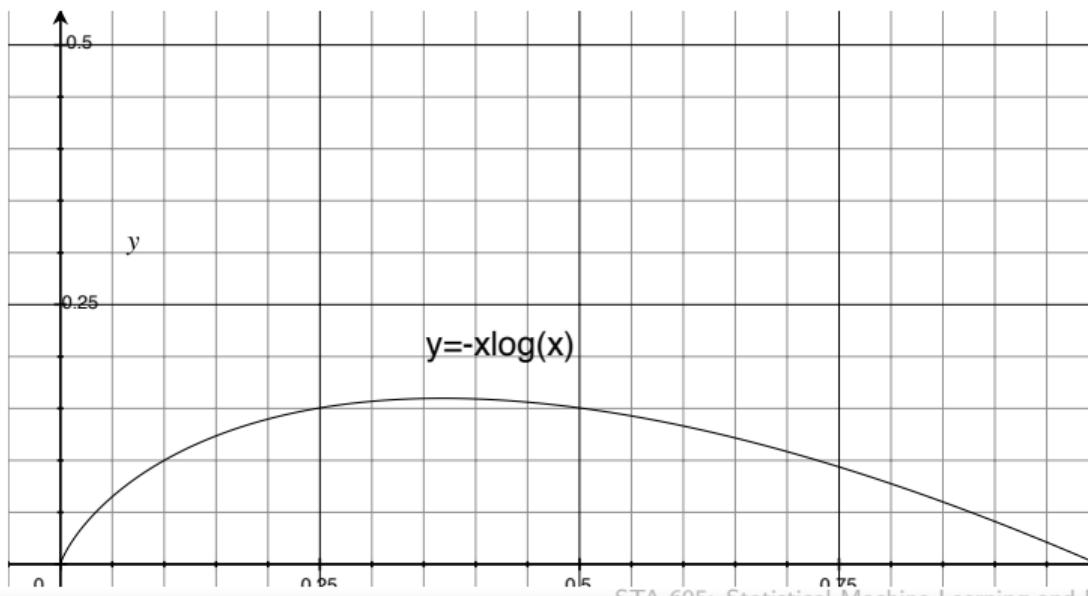
a measure of total variance across the K classes. The Gini index takes on a small value if all of the \hat{p}_{jk} 's are close to zero or one.

- For this reason the Gini index is referred to as a measure of node **purity** — a small value indicates that a node contains predominantly observations from a single class.
- The overall purity of a tree is a weighted average $\frac{1}{n} \sum_{j=1}^J n_j G_j$ with n_j being the number of training observations in region R_j .

Entropy

- An alternative to the Gini index is **entropy**, given by

$$D_j = - \sum_{k=1}^K \hat{p}_{jk} \log \hat{p}_{jk}.$$



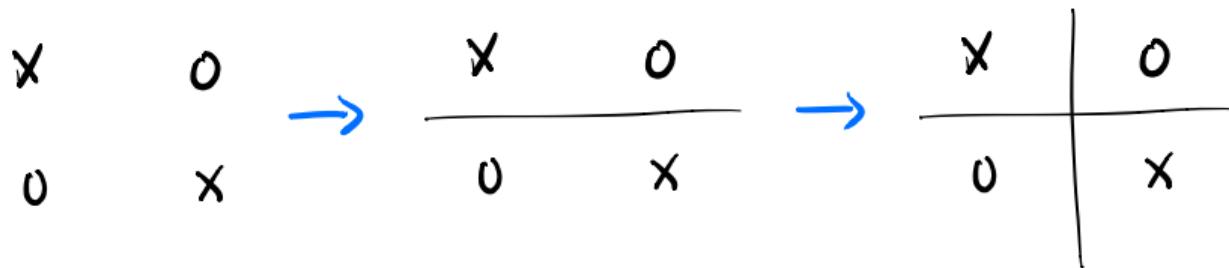
Comparison of classification error rate vs Gini index

Consider two scenarios for three-class classification

- 1 $p_1 = 50\%, p_2 = 25\%, p_3 = 25\%$
 - Classification error 0.5 vs Gini index 0.625
- 2 $p_1 = 50\%, p_2 = 50\%, p_3 = 0\%$
 - Classification error 0.5 vs Gini index 0.5

Gini index would favor second scenario as it is more pure whereas classification error rate cannot distinguish between the two.

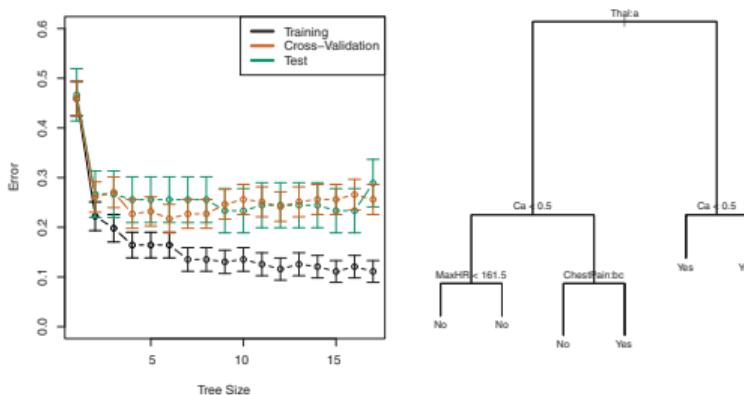
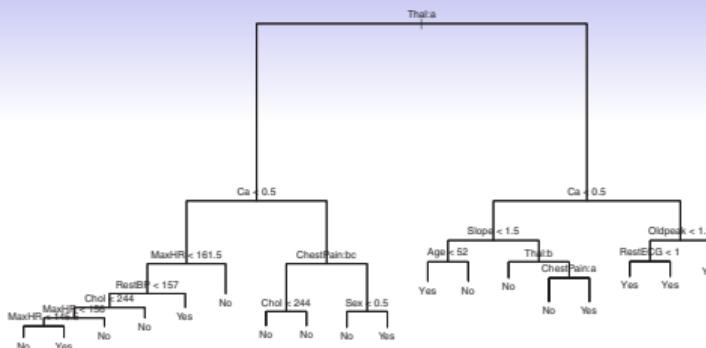
Why stopping growing a tree at some threshold is short-sighted?



ℓ_{ini} 50% ~~→~~ 50% → 0%

Example: heart data

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



Trees Versus Linear Models

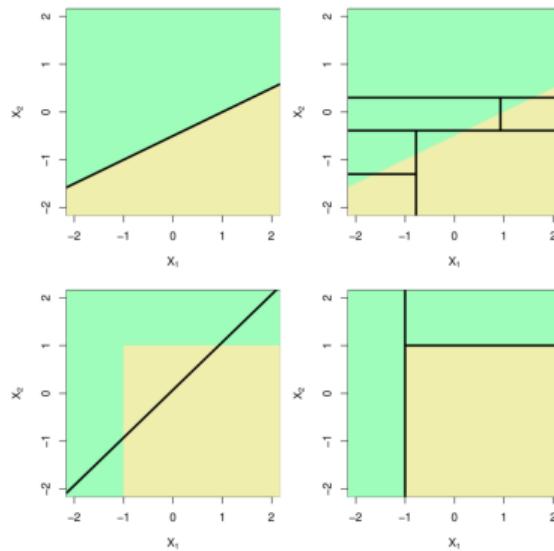


Figure: Top Row: True linear boundary; **Bottom row:** true non-linear boundary. **Left column:** linear model; **Right column:** tree-based model

Advantages and Disadvantages of Trees

- Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- Trees can easily handle qualitative predictors.
- Unfortunately, trees generally do not have the same level of predictive accuracy as some other regression and classification approaches.

However, by aggregating many decision trees, the predictive performance of trees can be substantially improved. We introduce these concepts next.

Model Assessment and Selection

- What is our goal?
 - ① Prediction
 - ② Interpretation
 - ③ Feature selection
- Which model do I use?
 - ① Contest-specific
 - ② Goal-specific
- Which tuning parameter to use?
 - ① Penalized Regression λ
 - ② SVM C/λ tuning parameter margin. Kernel tuning parameter γ, p

Model Assessment and Selection

- Model selection: Out of a class of models, choose the best specific model parameterized by λ .
- Model assessment: How good is my model? How do we expect my model to perform?

```
## Based on Scikit-learn
KNeighborsClassifier(K)
SVC(kernel="linear", C)
SVC(gamma, C)
DecisionTreeClassifier(max_depth=5)
Ridge(alpha)
Lasso(alpha)
ElasticNet(alpha, l1_ratio)
```

Training Error versus Test error

- The **test error/generalization error** is the average error that results from using a statistical learning method to predict the response on a new observation, one that was not used in training the method.
- In contrast, the **training error** can be easily calculated by applying the statistical learning method to the observations used in its training.
- But the training error rate often is quite different from the test error rate, and in particular the former can **dramatically underestimate** the latter.
- This problem is due to **over-fitting**.

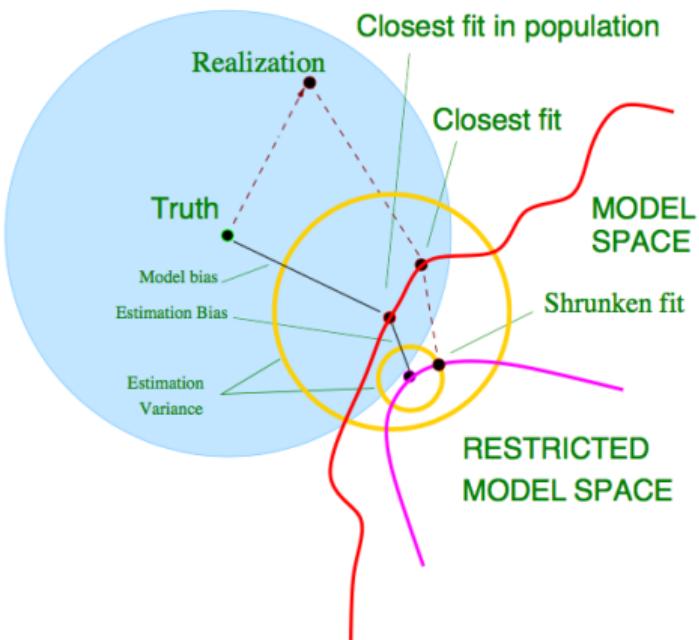
Prediction Error

- Prediction error at $X = x$: $E \left[L(y; \hat{f}(x)) \mid X = x \right]$
- Using squared error loss

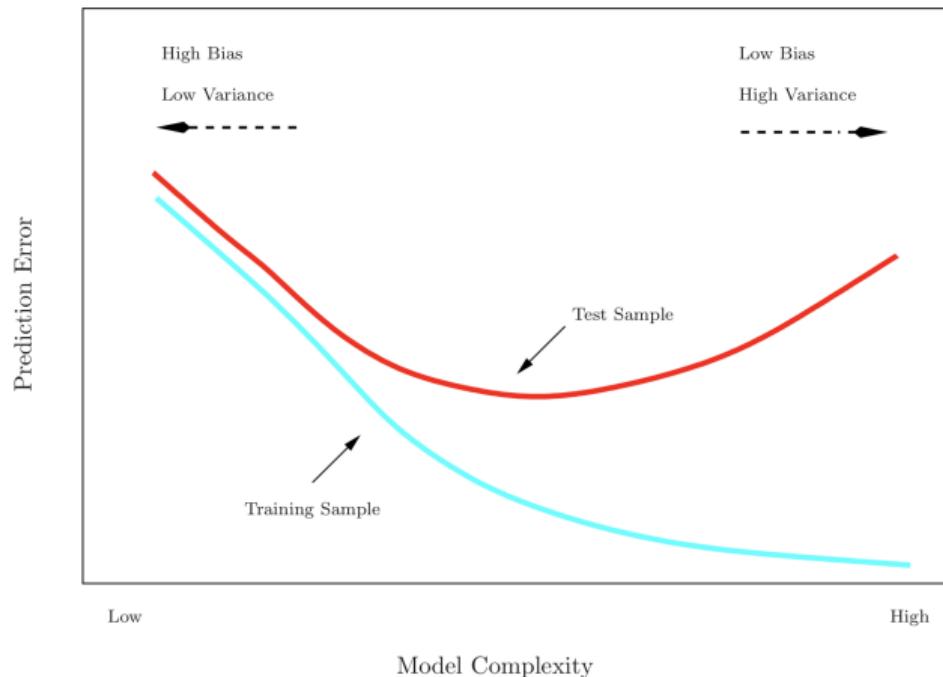
$$\begin{aligned} \text{Err}(x_0) &= E \left[(Y - \hat{f}(x_0))^2 \mid X = x_0 \right] \\ &= \sigma_\varepsilon^2 + \left[E\hat{f}(x_0) - f(x_0) \right]^2 + E \left[\hat{f}(x_0) - E\hat{f}(x_0) \right]^2 \\ &= \sigma_\varepsilon^2 + \text{Bias}^2(\hat{f}(x_0)) + \text{Var}(\hat{f}(x_0)) \\ &= \text{Irreducible Error} + \text{Bias}^2 + \text{Variance}. \end{aligned}$$

- Bias-variance decomposition:
 - ① Variance comes from estimators $\hat{\beta}$ and irreducible errors.
 - ② Bias comes from bias of our estimators and model bias.

Bias-Variance Decomposition



Training- versus Test-Set Performance



More on prediction-error estimates:

- In sample prediction error: training error is always optimistic for prediction error. $\text{TrErr} < \text{PredErr}$

① Using training error

- Mallows Cps/Akaike Information Criterion: Training Error $+ 2\frac{\hat{df}}{n}\hat{\sigma}_\epsilon^2$
- Bayesian Information Criterion: Training Error $+ \frac{\log n}{n}\hat{df}\hat{\sigma}_\epsilon^\alpha$

② Estimating the degree of freedom in a linear model ($\hat{y} = Hy$)

$$\hat{df} = \text{tr}(H) = \sum_{i=1}^n \frac{\text{Cov}(y_i, \hat{y}_i)}{\sigma}$$

- Ridge: $\hat{df} = \text{tr}\left(X^T (X^T X)^{-1} X\right)$
- Lasso: $\hat{df} = |\{\hat{\beta}\}|_0$ number of non-zero β s
- No df estimate for SVM

More on prediction-error estimates:

- Out of sample prediction error: cross-validation; bootstrapping
- Best solution: a large designated test set. Often not available
- We consider a class of methods that estimate the test error by **holding out** a subset of the training observations from the fitting process, and then applying the statistical learning method to those held out observations.

Validation-set approach

- Here we randomly divide the available set of samples into two parts: a **training set** and a **validation** or **hold-out** set.
- The model is fit on the **training set**, and the fitted model is used to predict the responses for the observations in the validation set.
- The resulting validation-set error provides an estimate of the test error. This is typically assessed using MSE in the case of a quantitative response and misclassification rate in the case of a qualitative (discrete) response.

The Validation process



A random splitting into two halves: left part is training set, right part is validation set.

The Validation process



A random splitting into two halves: left part is training set, right part is validation set.

Goals:

- (1) Pick a good tuning parameter
- (2) Estimate the test error

Training, Validation and Test Data

- ➊ Divide the data set into 3 trunks each for model fitting, model selection and model assessment.
- ➋ Fit on $X^{(1)}$ to get $\hat{\beta}(X^{(1)}, \lambda)$ where $\lambda = 1 \cdots \lambda_{\max}$
- ➌ Prediction error $\hat{y}_{(\lambda)}^{(2)} = X^{(2)}\hat{\beta}(X^{(1)}, \lambda)$

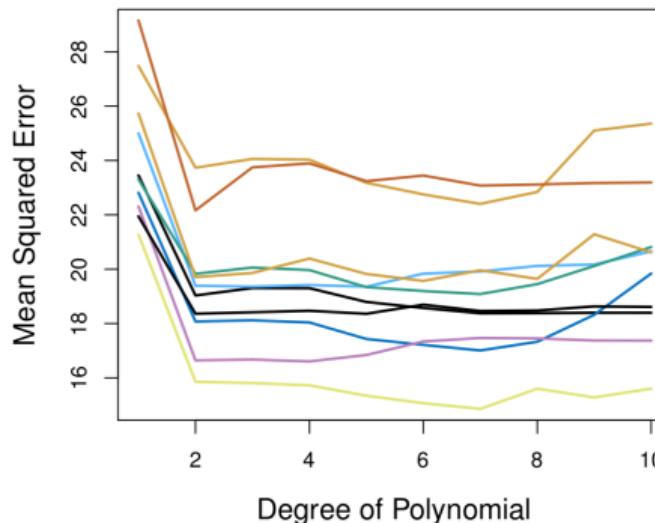
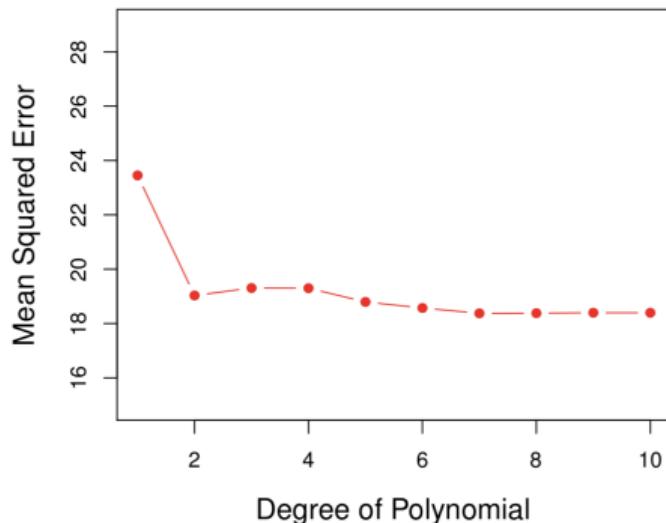
$$\lambda^* = \underset{\lambda}{\operatorname{argmin}} \left\{ \lambda \mid \left\| y^{(2)} - \hat{y}^{(2)} \right\|_2^2 \right\}$$

- ➍ Prediction error to report: $\left\| y^{(3)} - \hat{y}^{(3)} \right\|_2^2$
where $\hat{y}_{(\lambda)}^{(3)} = X^{(3)}\hat{\beta}(X^{(1)}, \lambda^*)$. (Do model selection and assessment separately!)

| Train | Validation | Test |
|-------|------------|------|
| 50% | 25% | 25% |

Example: automobile data

- Want to compare linear vs higher-order polynomial terms in a linear regression.
- We randomly split the 392 observations into two sets, a training set containing 196 of the data points, and a validation set containing the remaining 196 observations.



Drawbacks of validation set approach

- The validation estimate of the test error can be highly variable, depending on precisely which observations are included in the training set and which observations are included in the validation set.
- In the validation approach, only a subset of the observations — those that are included in the training set rather than in the validation set — are used to fit the model.
- This suggests that the validation set error may tend to **overestimate** the test error for the model fit on the entire data set.

Drawbacks of validation set approach

- The validation estimate of the test error can be highly variable, depending on precisely which observations are included in the training set and which observations are included in the validation set.
- In the validation approach, only a subset of the observations — those that are included in the training set rather than in the validation set — are used to fit the model.
- This suggests that the validation set error may tend to **overestimate** the test error for the model fit on the entire data set. **Why?**

K-fold Cross-validation

- **Widely used approach** for estimating test error.
- Estimates can be used to select best model, and to give an idea of the test error of the final chosen model.
- Idea is to randomly divide the data into K equal-sized parts. We leave out part k , fit the model to the other $K - 1$ parts (combined), and then obtain predictions for the left-out k th part (randomly reuse our data in K chunks for model selection and estimating the test error.).
- This is done in turn for each part $k = 1, 2, \dots, K$, and then the results are combined.
- Can we use the same prediction error for model selection and model assessment? **No!**

K-fold Cross-validation in detail.

Divide data into K roughly equal-sized parts ($K = 5$ here)

| 1 | 2 | 3 | 4 | 5 |
|------------|-------|-------|-------|-------|
| Validation | Train | Train | Train | Train |

K-fold Cross-validation in detail.

Divide data into K roughly equal-sized parts ($K = 5$ here)

| 1 | 2 | 3 | 4 | 5 |
|-------------------|-------------------|-------|-------|-------|
| <i>Validation</i> | Train | Train | Train | Train |
| Train | <i>Validation</i> | Train | Train | Train |

K-fold Cross-validation in detail.

Divide data into K roughly equal-sized parts ($K = 5$ here)

| 1 | 2 | 3 | 4 | 5 |
|-------------------|-------------------|-------------------|-------|-------|
| <i>Validation</i> | Train | Train | Train | Train |
| Train | <i>Validation</i> | Train | Train | Train |
| Train | Train | <i>Validation</i> | Train | Train |

K-fold Cross-validation in detail.

Divide data into K roughly equal-sized parts ($K = 5$ here)

| 1 | 2 | 3 | 4 | 5 |
|-------------------|-------------------|-------------------|-------------------|-------|
| <i>Validation</i> | Train | Train | Train | Train |
| Train | <i>Validation</i> | Train | Train | Train |
| Train | Train | <i>Validation</i> | Train | Train |
| Train | Train | Train | <i>Validation</i> | Train |

K-fold Cross-validation in detail.

Divide data into K roughly equal-sized parts ($K = 5$ here)

| 1 | 2 | 3 | 4 | 5 |
|------------|------------|------------|------------|------------|
| Validation | Train | Train | Train | Train |
| Train | Validation | Train | Train | Train |
| Train | Train | Validation | Train | Train |
| Train | Train | Train | Validation | Train |
| Train | Train | Train | Train | Validation |

The details

- Let the K parts be C_1, C_2, \dots, C_K , where C_k denotes the indices of the observations in part k . There are n_k observations in part k : if N is a multiple of K , then $n_k = n/K$.
- For $k = 1 \dots K$, $(Y^{\text{val}}, X^{\text{val}}) = (Y, X)_{i \in C_k}$; $(Y^{\text{tr}}, X^{\text{tr}}) = (Y, X)_{i \notin C_k}$
- Compute

$$\text{CV}_{(K)} = \frac{1}{n} \sum_{k=1}^K \sum_{i \in C_k} (y_i - \hat{y}_i)^2 = \sum_{k=1}^K \frac{n_k}{n} \text{MSE}_k,$$

where $\text{MSE}_k = \sum_{i \in C_k} (y_i - \hat{y}_i)^2 / n_k$ and \hat{y}_i is the fit for observation i , obtained from the data with part k removed.

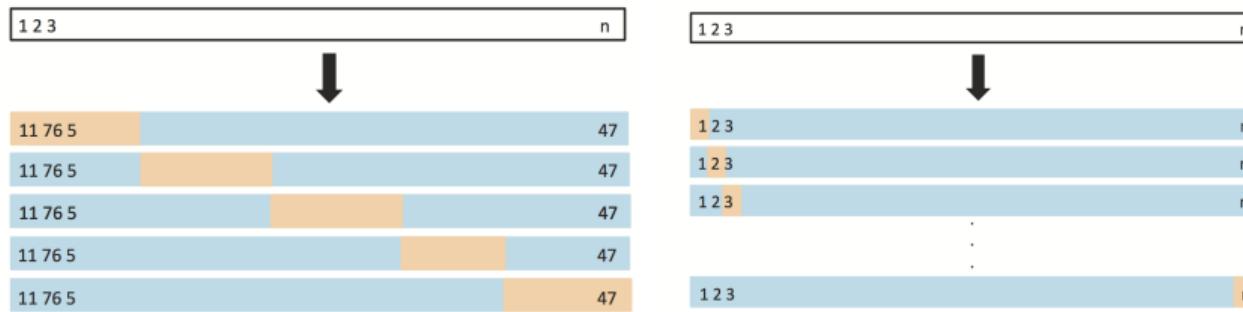
- Typically, $K = 5$ or 10

Leave-one-out Cross Validation (LOOCV)

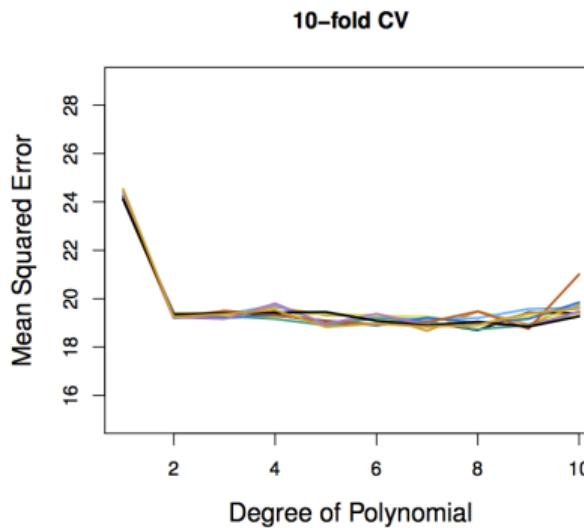
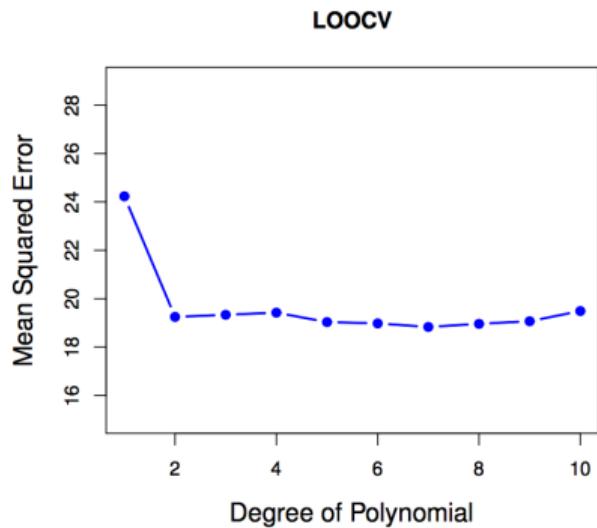
- Setting $K = n$ yields n-fold or leave-one-out cross-validation (LOOCV).
- Pros:
 - Approximately unbiased estimates of prediction error
 - Reproducible (less randomness)
- Cons:
 - High variance of prediction error
 - Computational expensive!
- Typically used in a linear model $\hat{Y} = HY$

$$LOOCV == \frac{1}{n} \sum \left[\frac{\hat{y}_i - y_i}{1 - H_{ii}} \right]^2$$

K-fold vs. LOOCV



K-fold vs. LOOCV



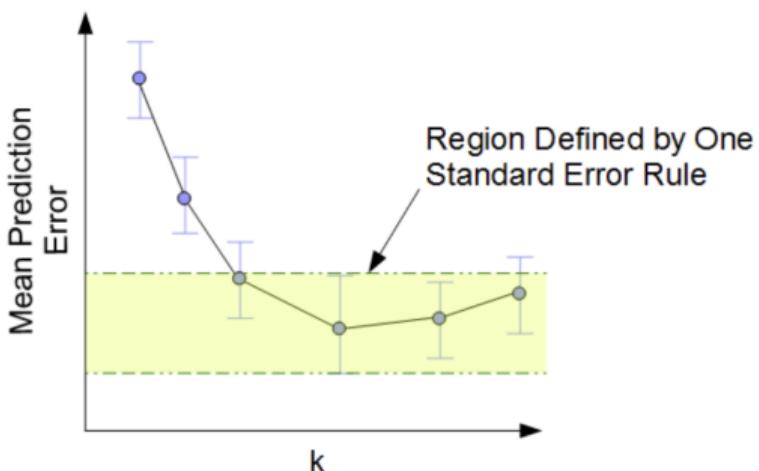
The details

- Cross Validation K folds for model selection
 - ① Reuse data for both fitting and model selection.
 - ② Fitting model K times (not friendly to computationally expensive scenarios)
- Steps:
 - ① Randomly Chunk our data set into K equal groups.
 - ② For $\lambda = \lambda_1 \cdots \lambda_{\max}$. Fit model $\hat{\beta}(X^{tr}, \lambda)$ for $k = 1 \cdots K$ and obtain $CV_{(K)}(\lambda)$
 - ③ Two rules for model selection:
 - Minimum Error: $\lambda^{\min} = \operatorname{argmin}_{\lambda} [CV_{(K)}(\lambda)]$
 - One SE rule: largest λ within 1 SE of min CV

$$\hat{SE}(\lambda) = \sqrt{\operatorname{Var}(CV_{(K)}(\lambda))/K}$$

$$\lambda^{1SE} = \operatorname{argmax}_{\lambda} \{ CV_{(K)}(\lambda) < CV_{(K)}(\lambda^{\min}) + \hat{SE}(\lambda^{\min}) \}$$

One SE rule



Cross-Validation for Classification Problems

- We divide the data into K roughly equal-sized parts C_1, C_2, \dots, C_K . C_k denotes the indices of the observations in part k . There are n_k observations in part k : if n is a multiple of K , then $n_k = n/K$.
- Compute

$$CV_K = \frac{1}{n} \sum_{k=1}^K \sum_{i \in C_k} I(y_i \neq \hat{y}_i) = \sum_{k=1}^K \frac{n_k}{n} Err_k$$

where $Err_k = \sum_{i \in C_k} I(y_i \neq \hat{y}_i)/n_k$.

Cross-validation: right and wrong

- Consider a simple classifier applied to some two-class data:
 - Starting with 5000 predictors and 50 samples, find the 100 predictors having the largest correlation with the class labels.
 - We then apply a classifier such as logistic regression, using only these 100 predictors.

How do we estimate the test set performance of this classifier?

Cross-validation: right and wrong

- Consider a simple classifier applied to some two-class data:
 - Starting with 5000 predictors and 50 samples, find the 100 predictors having the largest correlation with the class labels.
 - We then apply a classifier such as logistic regression, using only these 100 predictors.

How do we estimate the test set performance of this classifier?

Can we apply cross-validation in step 2, forgetting about step 1?

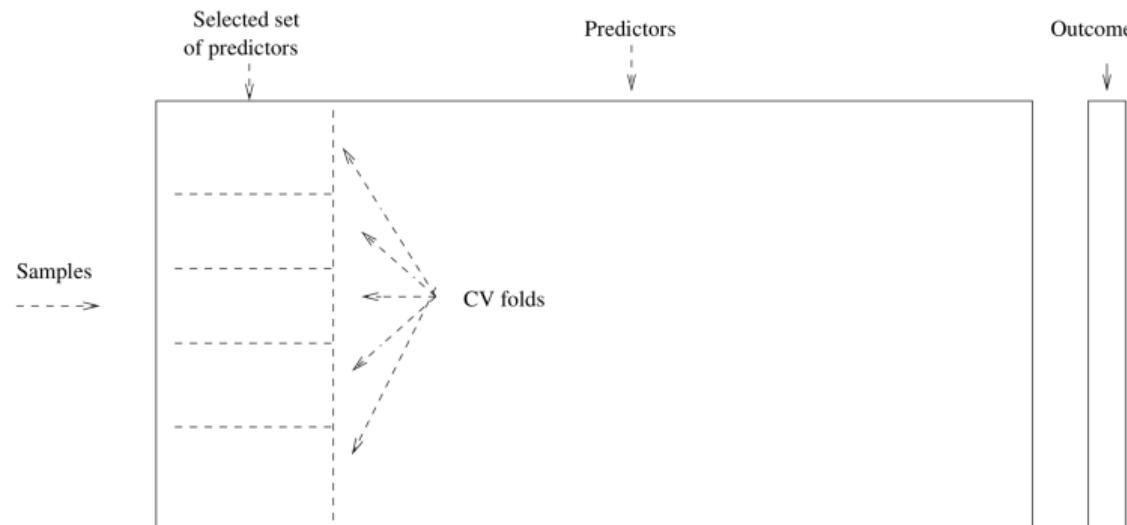
NO!

- This would ignore the fact that in Step 1, the procedure **has already seen the labels of the training data**, and made use of them. This is a form of training and must be included in the validation process.
- It is easy to simulate realistic data with the class labels independent of the outcome, so that true test error =50%, but the CV error estimate that ignores Step 1 is almost zero!
Try to do this yourself!
- This error has made in many high profile genomics papers.

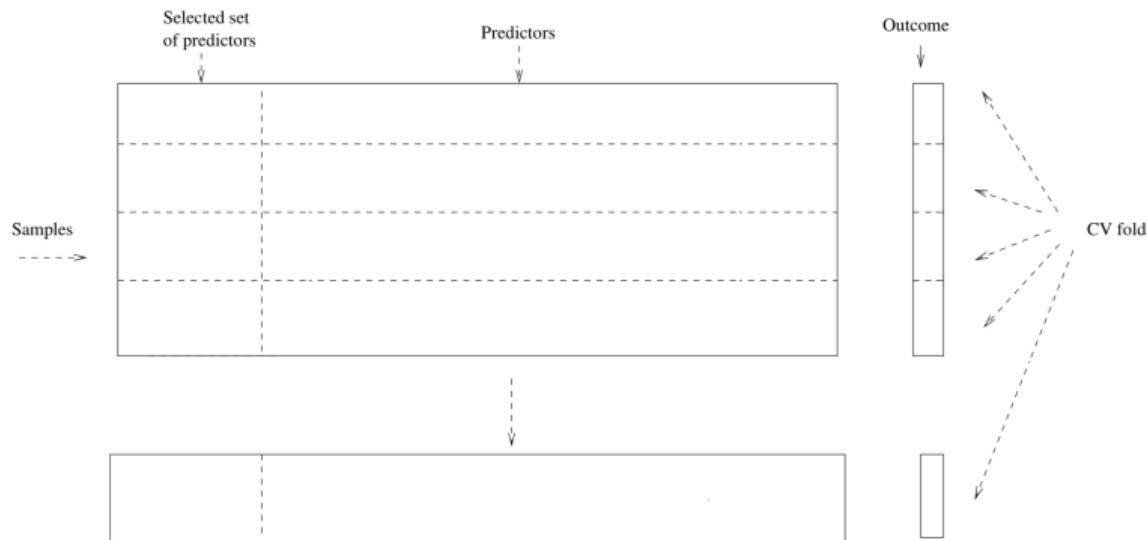
The Wrong and Right Way

- **Wrong:** Apply cross-validation in step 2.
- **Right:** Apply cross-validation to steps 1 and 2.

Wrong way



Right way



Something about Cross Validation

- What K ?
 - When K small for fitting, there will be high bias for estimation (training size is small) but computation time saving
 - When K large, computation time is quite intense and overlapping data (less data shake up), but good for model fitting
- Each of the K models are highly correlated
- The CV is not a good estimation of the prediction error: You used a different n . Since we use less n to fit the model, the cross validation error would be larger than the prediction error. What matters is the CV error has the same shape as the prediction error.
- For the lasso, the CV always over-select.
- Stratified CV: Make sure each fold has an equivalent class distribution to the original training data set.

The Bootstrap

- The **bootstrap** is a flexible and powerful statistical tool that can be used to quantify the uncertainty associated with a given estimator or statistical learning method.
- For example, it can provide an estimate of the standard error of a coefficient, or a confidence interval for that coefficient.

A simple example

- Suppose that we wish to invest a fixed sum of money in two financial assets that yield returns of X and Y , respectively, where X and Y are random quantities.
- We will invest a fraction α of our money in X , and will invest the remaining $1 - \alpha$ in Y .
- We wish to choose α to minimize the total risk, or variance, of our investment. In other words, we want to minimize $\text{Var}(\alpha X + (1 - \alpha) Y)$.
- One can show that the value that minimizes the risk is given by

$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}},$$

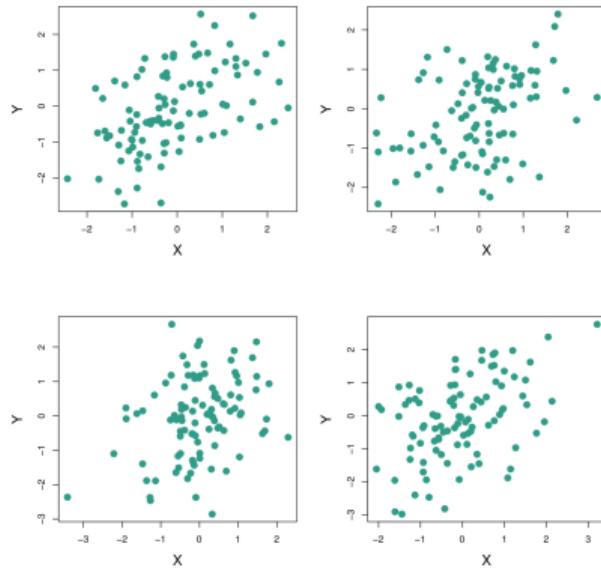
where $\sigma_X^2 = \text{Var}(X)$, $\sigma_Y^2 = \text{Var}(Y)$, and $\sigma_{XY} = \text{Cov}(X, Y)$.

Example continued

- But the values of σ_X^2 , σ_Y^2 , and σ_{XY} are unknown.
- We can compute estimates for these quantities, $\hat{\sigma}_X^2$, $\hat{\sigma}_Y^2$, and $\hat{\sigma}_{XY}$, using a data set that contains measurements for X and Y .
- We can then estimate the value of α that minimizes the variance of our investment using

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

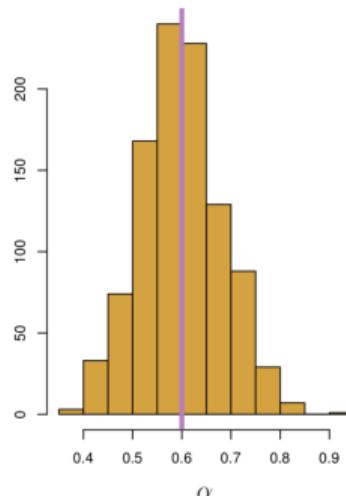
Example continued



Each panel displays 100 simulated returns for investments X and Y. From left to right and top to bottom, the resulting estimates for α are 0.576, 0.532, 0.657, and 0.651.

Example continued

- To estimate the standard deviation of $\hat{\alpha}$, we repeated the process of simulating 100 paired observations of X and Y , and estimating α 1,000 times.
- We thereby obtained 1,000 estimates for α , which we can call $\hat{\alpha}_1, \hat{\alpha}_2, \dots, \hat{\alpha}_{1000}$.



Example continued

- The mean over all 1,000 estimates for α is

$$\bar{\alpha} = \frac{1}{1000} \sum_{r=1}^{1000} \hat{\alpha}_r = 0.5996,$$

very close to $\alpha = 0.6$, and the standard deviation of the estimates is

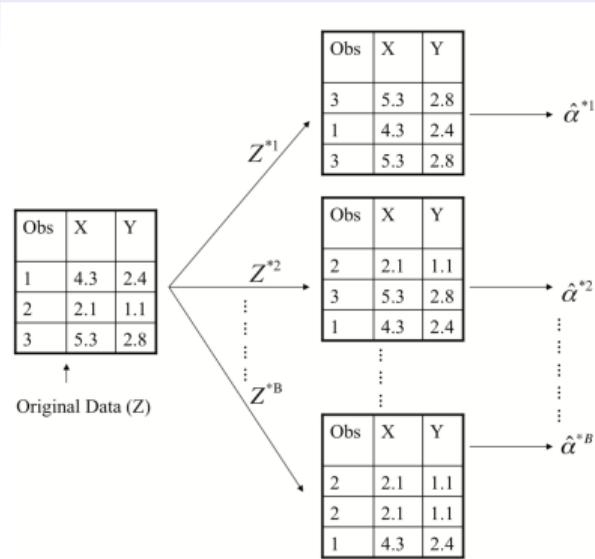
$$\sqrt{\frac{1}{1000 - 1} \sum_{r=1}^{1000} (\hat{\alpha}_r - \bar{\alpha})^2} = 0.083.$$

- This gives us a very good idea of the accuracy of $\hat{\alpha}$: $SE(\hat{\alpha}) = 0.083$.

Now back to the real world

- The procedure outlined above cannot be applied, because for real data we cannot generate new samples from the original population.
- However, the bootstrap approach allows us to use a computer to mimic the process of obtaining new data sets, so that we can estimate the variability of our estimate without generating additional samples.
- Rather than repeatedly obtaining independent data sets from the population, we instead obtain distinct data sets by repeatedly sampling observations from the original data set **with replacement**.
- Each of these “bootstrap data sets” is created by sampling **with replacement**, and is the **same size** as our original dataset. As a result some observations may appear more than once in a given bootstrap data set and some not at all.

Example with just 3 observations



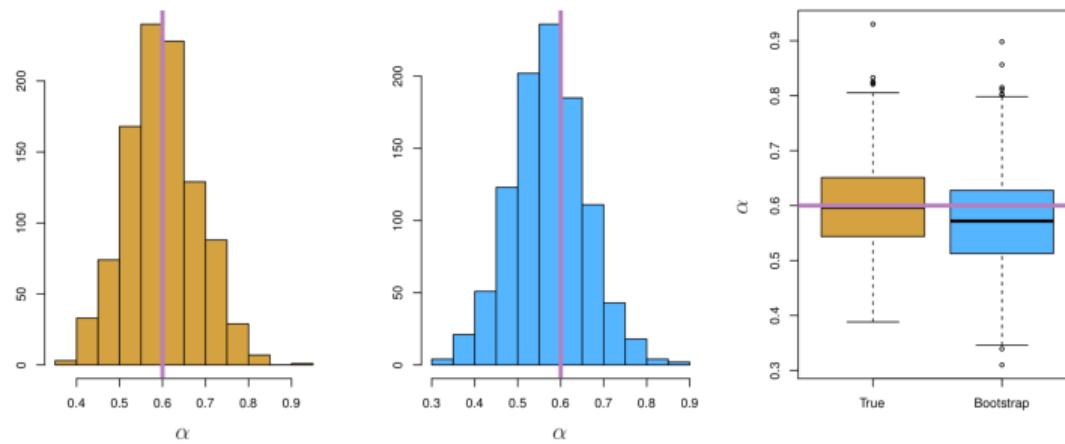
A graphical illustration of the bootstrap approach on a small sample containing $n = 3$ observations. Each bootstrap data set contains n observations, sampled with replacement from the original data set. Each bootstrap data set is used to obtain an estimate of α .

- Denoting the first bootstrap data set by Z^{*1} , we use Z^{*1} to produce a new bootstrap estimate for α , which we call $\hat{\alpha}^{*1}$.
- This procedure is repeated B times for some large value of B (say 100 or 1000), in order to produce B different bootstrap data sets, $Z^{*1}, Z^{*2}, \dots, Z^{*B}$, and B corresponding α estimates, $\hat{\alpha}^{*1}, \hat{\alpha}^{*2}, \dots, \hat{\alpha}^{*B}$.
- We estimate the standard error of these bootstrap estimates using the formula

$$\text{SE}_B(\hat{\alpha}) = \sqrt{\frac{1}{B-1} \sum_{r=1}^B (\hat{\alpha}^{*r} - \bar{\hat{\alpha}}^*)^2}.$$

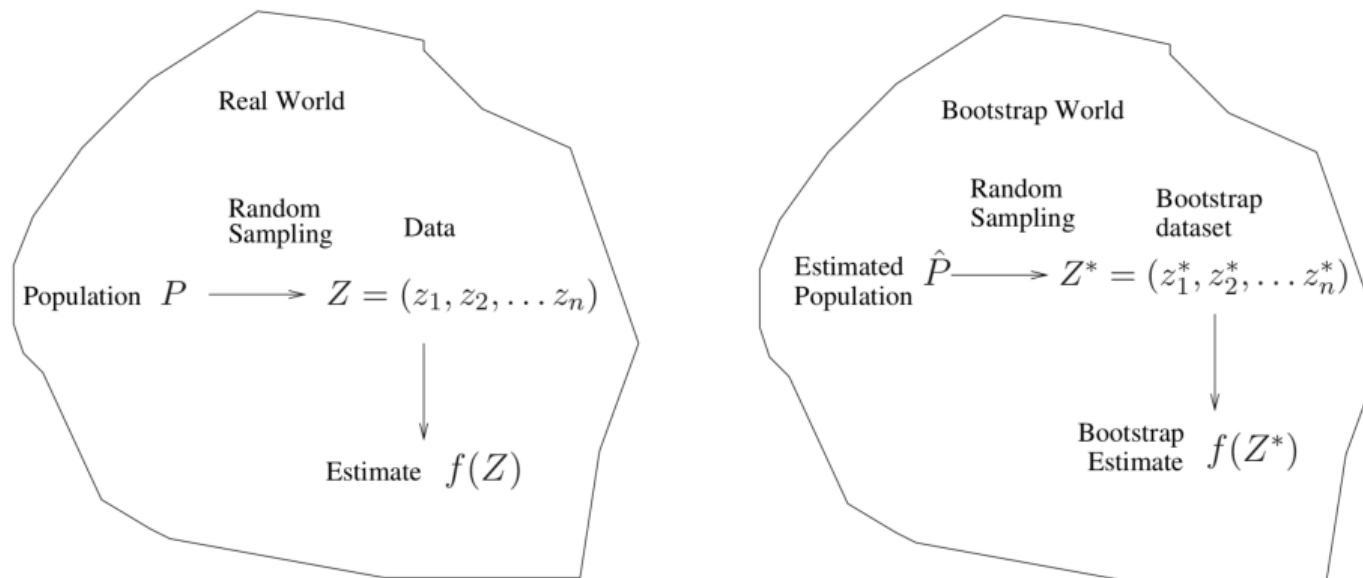
- This serves as an estimate of the standard error of $\hat{\alpha}$ estimated from the original data set.

Results



Left: A histogram of the estimates of α obtained by generating 1,000 simulated data sets from the true population. $SE(\hat{\alpha}) = 0.083$. **Center:** A histogram of the estimates of α obtained from 1,000 bootstrap samples from a single data set. $SE_B(\hat{\alpha}) = 0.087$. **Right:** The estimates of α displayed in the left and center panels are shown as boxplots. In each panel, the pink line indicates the true value of α .

A general picture for the bootstrap



Key Point Summary

- Model Selection: Choose tuning parameters for a class of models.
- Model Assessment: How well will the model perform in the future?
- Bias-Variance tradeoff
- Use different samples for model fitting, model selection, and model assessment.
- Cross-validation to select a model. $CV(\lambda^*) < \text{PredErr}$ (no peeking at test set).
- Right way to do CV: must be applied to the entire sequence of modeling steps (including any selection or filtering steps that use the label!)

Ensemble Learning

- Idea: combine strengths of several weak learners

$$\hat{f}(x) = \sum_{b=1}^B \hat{\omega}_b \hat{f}^b(x).$$

where $\hat{f}^b(x)$ is a weak learner.

- Ensemble learning: build a prediction model by combining the strengths of a collection of simpler base models.
 - 1 Develop a population of base learners from the training data
 - 2 Combine them to form the composite predictor
- Methods:
 - Bagging
 - Boosting
 - Model averaging and stacking

Bagging

- **Bootstrap aggregation, or bagging,** is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.
- Recall that given a set of n independent observations Z_1, \dots, Z_n , each with variance σ^2 , the variance of the mean \bar{Z} of the observations is given by σ^2/n .
- In other words, **averaging a set of observations reduces variance**. Of course, this is not practical because we generally do not have access to multiple training sets.

Bagging– continued

- Instead, we can bootstrap, by taking repeated samples from the (single) training data set.
- In this approach we generate B different bootstrapped training data sets. We then train our method on the b th bootstrapped training set in order to get $\hat{f}^{*b}(x)$, the prediction at a point x . We then average all the predictions to obtain

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

This is called **bagging**.

Bagging classification trees

- The above prescription applies to regression trees
- For classification trees: for each test observation, we record the class predicted by each of the B trees, and take a **majority vote**: the overall prediction is the most commonly occurring class among the B predictions.

Out-of-Bag Error Estimation

- It turns out that there is a very straightforward way to estimate the test error of a bagged model.
- Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations. One can show that on average, each bagged tree makes use of around two-thirds of the observations. ($P(i \in B) = 1 - (1 - \frac{1}{n})^n \approx 1 - e^{-1} \approx 0.632$)
- 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 i th observation using each of the trees in which that observation was OOB. This will yield around $B/3$ predictions for the i th observation, which we average.
- This estimate is essentially the leave-one-out (LOO) cross-validation error for bagging, if B is large.

Out-of-Bag Error Estimation

① For $b = 1, \dots, B$:

- Take a bootstrap sample z^{*b} (of n observations) to use as a training set
- Fit a regression tree \hat{f}^{*b}

② For $i = 1, \dots, n$:

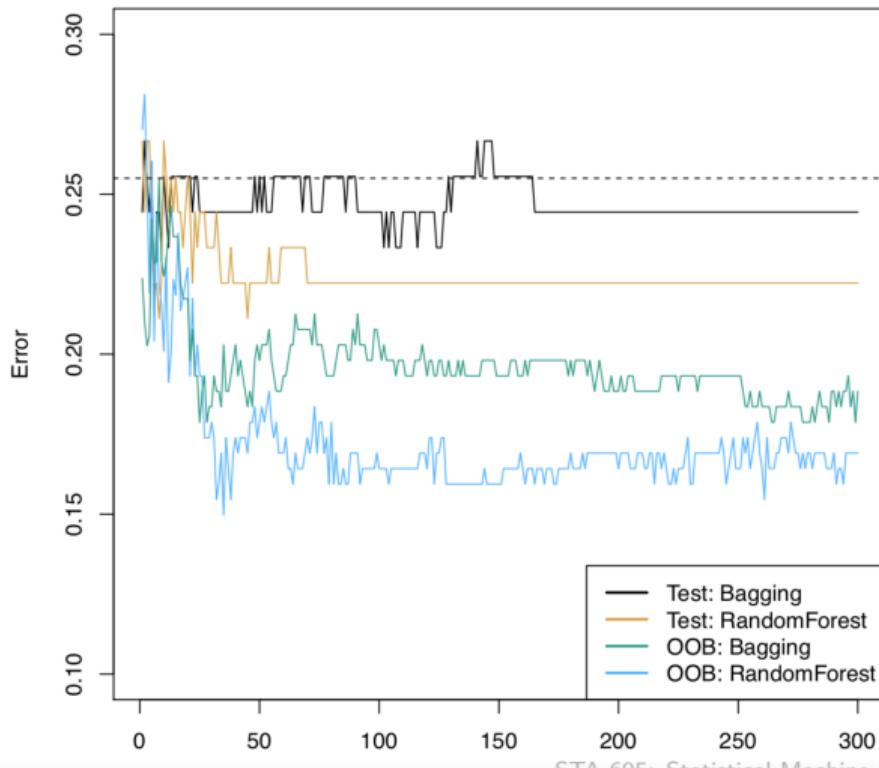
- Set $S = \{b : (y_i, x_i) \notin z^{*b}\}$
- Calculate

$$OOB_{SE}[i] = \left(y_i - \frac{1}{|S|} \sum_{b \in S} \hat{f}^{*b}(x_i) \right)^2$$

③ Final OOB MSE estimate = $\frac{1}{n} \sum_{i=1}^n OOB_{SE}[i]$

Note: if B (and hence $|S|$) is large then $OOB_{SE}[i]$ is approximately the same as $LOO_{SE}[i]$ i.e. removing sample i and finding squared error of the bagging prediction based on the $n - 1$ remaining observations

Bagging the heart data



Details of previous figure

Bagging and random forest results for the Heart data.

- The test error (black and orange) is shown as a function of B , the number of bootstrapped training sets (or equivalently the number of trees) used.
- Random forests were applied with $m = \sqrt{p}$.
- The dashed line indicates the test error resulting from a single classification tree.
- The green and blue traces show the OOB error, which in this case is considerably lower.
- Note that the error does not increase as $B \rightarrow \infty$.

More about Bagging

- For regression tree, we would like to have

$$\text{MSE}(\hat{f}_{\text{tree}}) > \text{MSE}(\hat{f}_{\text{bag}})$$

- Averaging reduce variance! (Law of Large Number)
- Does bagging overfit? No, but if individual learners overfit, then bagging might overfit.
- No interpretation.
- Effect of Bagging
 - Tree Model: Bagging will not decrease bias but decrease variance of Tree Regression.
 - Linear Model: Both Bias and variance are unchanged, since linear model's BLUE model or minimal variance models.
 - However, for sparse regression with penalty bagging will take effect.
- Are our trees independent? No!

Random Forests

- **Random forests** provide an improvement over bagged trees by way of a small tweak that **decorrelates** the trees. This reduces the variance when we average the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, a **random selection of m predictors** is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors.
- A fresh selection of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$ — that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors (4 out of the 13 for the Heart data).

Random Forests

RF Algorithm: (B , m : # number of splits each tree, $|T|$: tree size)

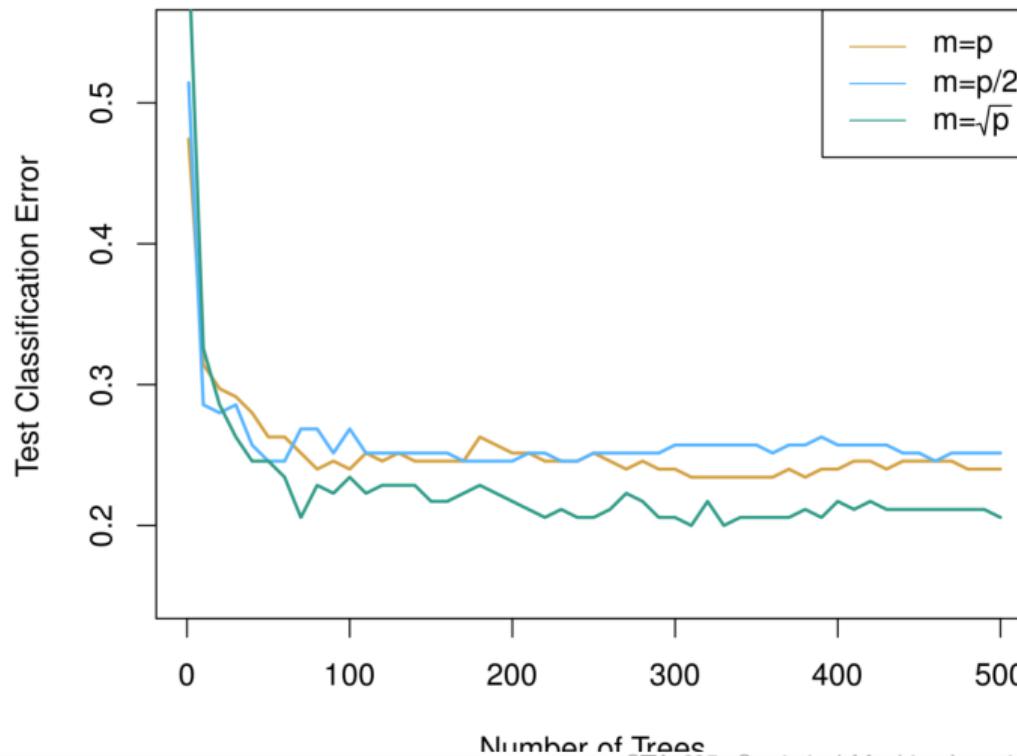
For $b = 1 \cdots B$

- ① Bootstrap sample of size n .
- ② Grow a RF tree T_b .
 - ① Select m vars randomly out of p variables.
 - ② Find the best split and split point out of the m vars.
 - ③ Split into 2 children.
 - ④ Recursively repeat until converge.

Example: gene expression data

- Now we apply random forests to a high-dimensional biological data set consisting of expression measurements of 4,718 genes measured on tissue samples from 349 patients.
- There are around 20,000 genes in humans, and individual genes have different levels of activity, or expression, in particular cells, tissues, and biological conditions.
- Each of the patient samples has a qualitative label with 15 different levels: either normal or one of 14 different types of cancer.
- We use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.
- We randomly divided the observations into a training and a test set, and applied random forests to the training set for three different values of the number of splitting variables m .

Results: gene expression data



Details of previous figure

- Results from random forests for the fifteen-class gene expression data set with $p = 500$ predictors.
- The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m , the number of predictors available for splitting at each interior tree node.
- Random forests ($m < p$) lead to a slight improvement over bagging ($m = p$). A single classification tree has an error rate of 45.7%.

Boosting

- Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification. We only discuss boosting for decision trees.
- Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model.
- Notably, 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 is grown using information from previously grown trees.
- Sequential additive models "week" learners in a Greedy manner: $\hat{f} = \sum_{m=1}^M \alpha_m \hat{f}_m(x)$
- Key strategies: fit sequentially according to the residuals and adjust the observation weights; additive ensemble of week learners.

Boosting algorithm for regression trees

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

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

- Update the residuals,

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

3. Output the boosted model,

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

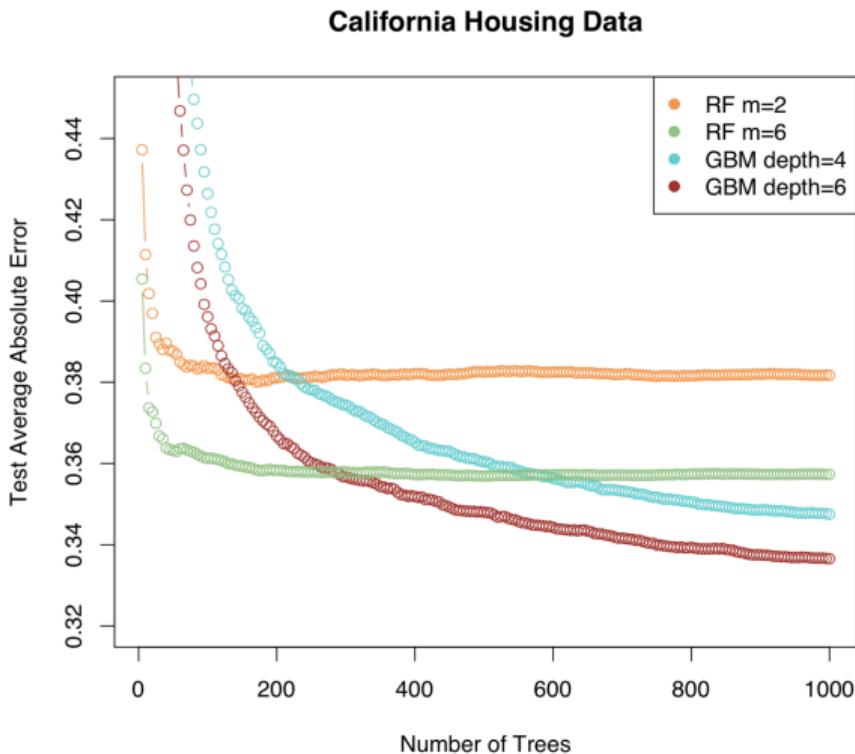
Tuning parameters for boosting

1. The **number of trees** B . Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B .
2. The **shrinkage parameter** λ , a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.
3. The **number of splits** d in each tree, which controls the complexity of the boosted ensemble. Often $d = 1$ works well, in which case each tree is a **stump**, consisting of a single split and resulting in an additive model. More generally d is the **interaction depth**, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

What is the idea behind this procedure?

- Unlike fitting a single large decision tree to the data, which amounts to **fitting the data hard** and potentially overfitting, the boosting approach instead **learns slowly**.
- Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals.
- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.
- By fitting small trees to the residuals, we slowly improve \hat{f} in areas where it does not perform well. The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals.

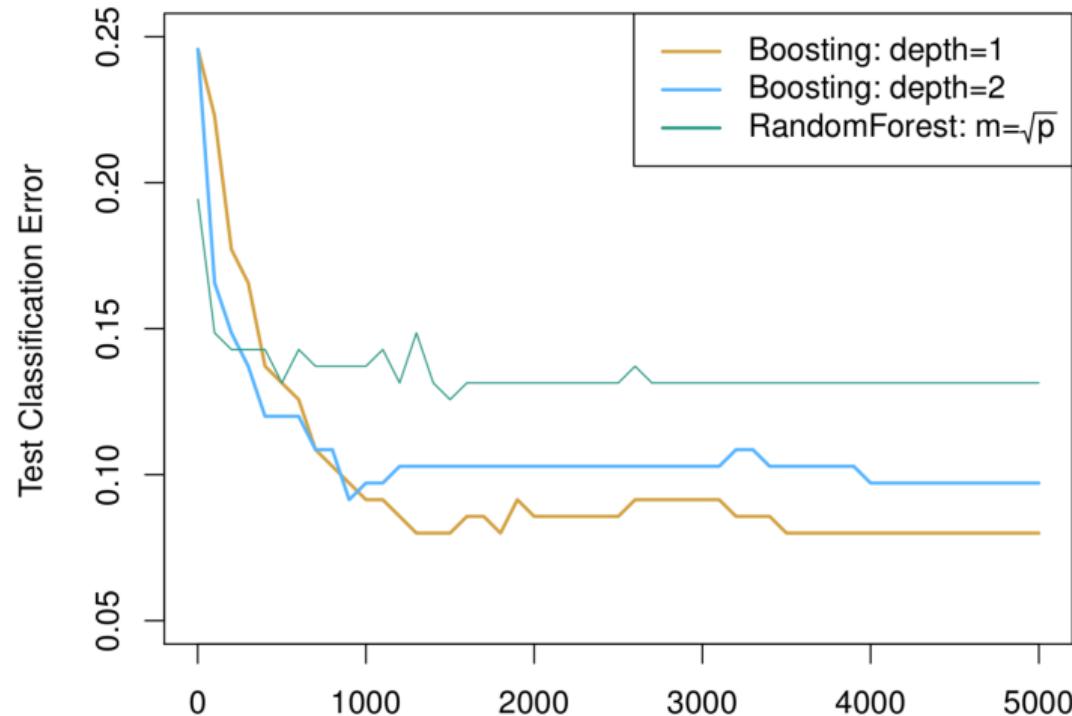
Regression examples



Boosting for classification

- Boosting for classification is similar in spirit to boosting for regression but is a bit more complex.
- The R package **gbm** (gradient boosted models) handles a variety of regression and classification problems.
- LightGBM (Light Gradient Boosting Machine) a free and open-source distributed gradient-boosting framework for machine learning (Python, R)
- XGBoost (eXtreme Gradient Boosting): a Scalable, Portable and Distributed Gradient Boosting software library that provides a regularizing gradient boosting framework for C++, Java, Python, R, Julia, Perl, and Scala
- GBM in H2O: Distributed, Fast & Scalable Machine Learning Platform

Gene expression data continued



Details of previous figure

- Results from performing boosting and random forests on the fifteen-class gene expression data set in order to predict **cancer** versus **normal**.
- The test error is displayed as a function of the number of trees. For the two boosted models, $\lambda = 0.01$. Depth-1 trees slightly outperform depth-2 trees, and both outperform the random forest, although the standard errors are around 0.02, making none of these differences significant.
- The test error rate for a single tree is 24%.

Forward Stagewise Additive Modeling

- For $m = 1 \cdots M$

$$(\alpha_m, \gamma_m) = \underset{\alpha, \gamma}{\operatorname{argmin}} \sum_{i=1}^n L(y_i, f_{m-1}(x_i) + \alpha g(x_i, \gamma))$$

and update the model as

$$f_m(x) = f_{m-1}(x) + \alpha_m g(x, \gamma_m)$$

- Exponential loss: Adaboost

Boosting algorithm for regression trees

- For $m = 1, \dots, M$

$$(\alpha_m, \gamma_m) = \underset{\alpha, \gamma}{\operatorname{argmin}} \sum_{i=1}^n L(y_i, \underbrace{f_{m-1}(x_i)}_{\text{fitting to the residuals}} + \alpha g(x, \gamma))$$

$$f_m(x) = f_{m-1}(x) + \alpha_m g(x, \gamma_m)$$

- Output $\hat{f}_m(x) = \sum_{m=1}^M \hat{\alpha}_m \hat{g}(x, \gamma_m)$
where $L()$ can be different loss function and $g()$ can be different learners.
- Accelerate learning $f_m(x) = f_{m-1}(x) + \eta \alpha_m g(x, \gamma_m)$
where η is the learning rate $\in (0, 1)$ (tuning parameter). And $\eta \downarrow$ slow learner (preferred) and $\eta \uparrow$ faster learner (the faster the learner the higher tendency of overfit).

Adaboost

Suppose we have $Y \in \{-1, 1\}$ and $X_{n \times p}$.

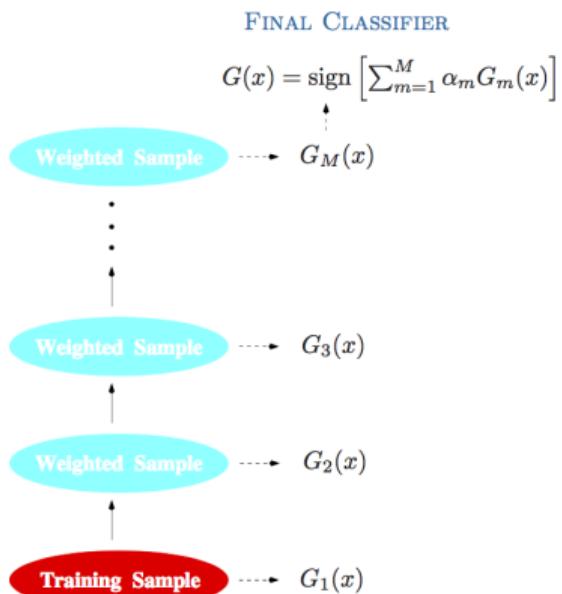
- ① Initialize weights as $w_i = \frac{1}{n}$
- ② Repeat for $m = 1 \cdots M$
 - ① Fit a classifier $f_m(x)$ with weights to Y that minimizes the weighted sum error of misclassification
 - ② Calculate weighted misclassification error

$$err_m = \frac{\sum_{i=1}^m w_i I(y_i \neq f_m(x_i))}{\sum_{i=1}^m w_i}$$

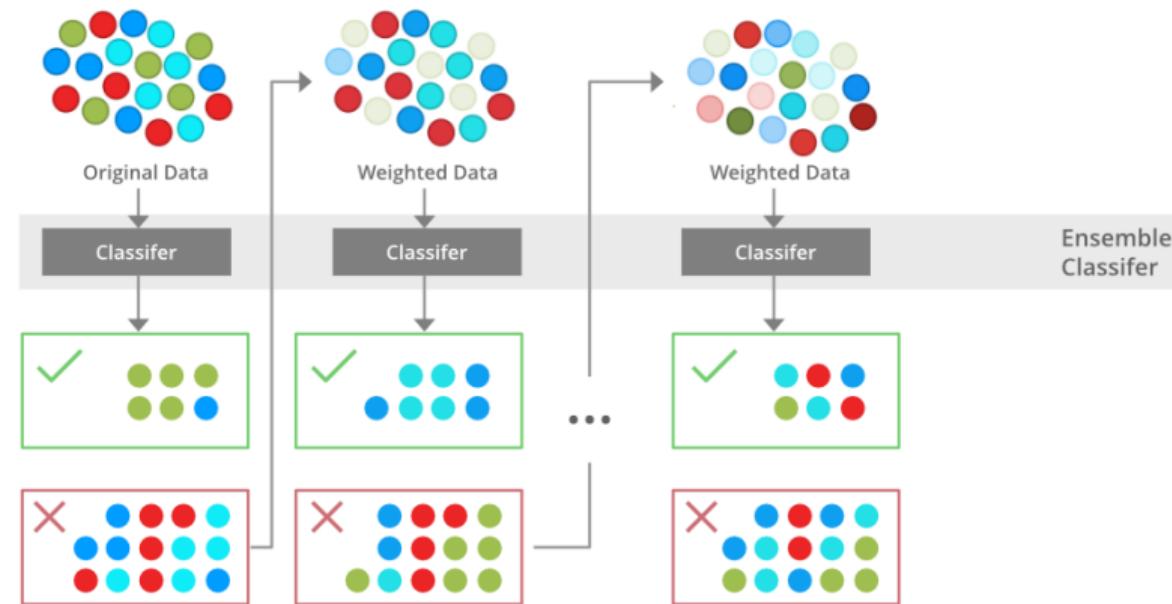
- ③ Calculate the α_m (the log odds of weighted errors) defined as $\alpha_m = \log((1 - err_m)/err_m)$
- ④ Now we update our weights as $w_i := w_i \exp[\alpha_m I(y_i \neq f_m(x_i))]$ and normalize our weights such that $\sum_{i=1}^n w_i = 1$
- ⑤ Output our final classifier as

$$\text{sign} \left[\sum_{m=1}^M \alpha_m f_m(x) \right]$$

Adaboost



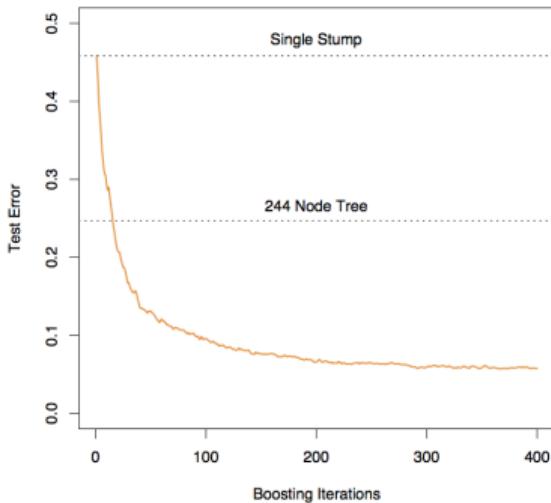
Adaboost



Adaboost

- The above process means the following
 - Good classifier f_m has larger coefficients α_m
 - Observation misclassified by $\hat{f}_m(x_i)$, then we update the observation weights as $w_i := w_i \exp(\alpha_m)$ such that the weights will increase on misclassified observations.
 - Weights are adjusted to weigh up all misclassified observations.
 - A classifier with 50% accuracy is given a weight of zero and a classifier with less than 50% accuracy (flipping!) is given a negative weight
- Use tree stumps as base learners
- Adaboost minimizes the exponential loss criterion within a forward-stagewise additive modeling framework

Adaboost



Adaboost

- Exponential loss function

$$L(y, f(x)) = \exp(-yf(x))$$

Positive margin: $f(x_i)y_i > 0$, correctly classified

Negative margin: $f(x_i)y_i < 0$, miss-classified

- Sequential greedy optimization for

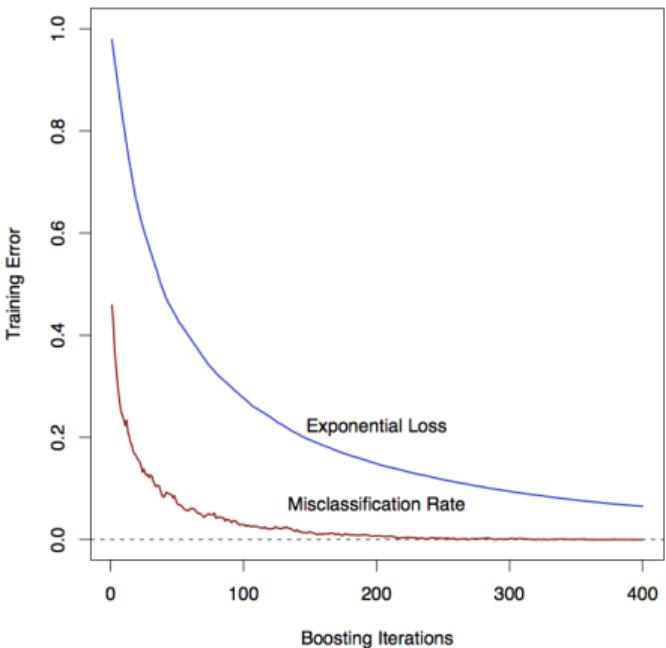
$$L(y, f(x)) = \sum_i \exp\left(-\frac{1}{2}y_i \sum_{m=1}^M \alpha_m \hat{f}_m(x)\right)$$

- The population minimizer:

$$\frac{1}{2} \log \left(\frac{P(Y=1)}{P(Y=-1)} \right)$$

which is the same as logistic loss/binomial deviation loss $\sum_i \log(1 + e^{-y_i f(x_i)})$

Adaboost



Adaboost

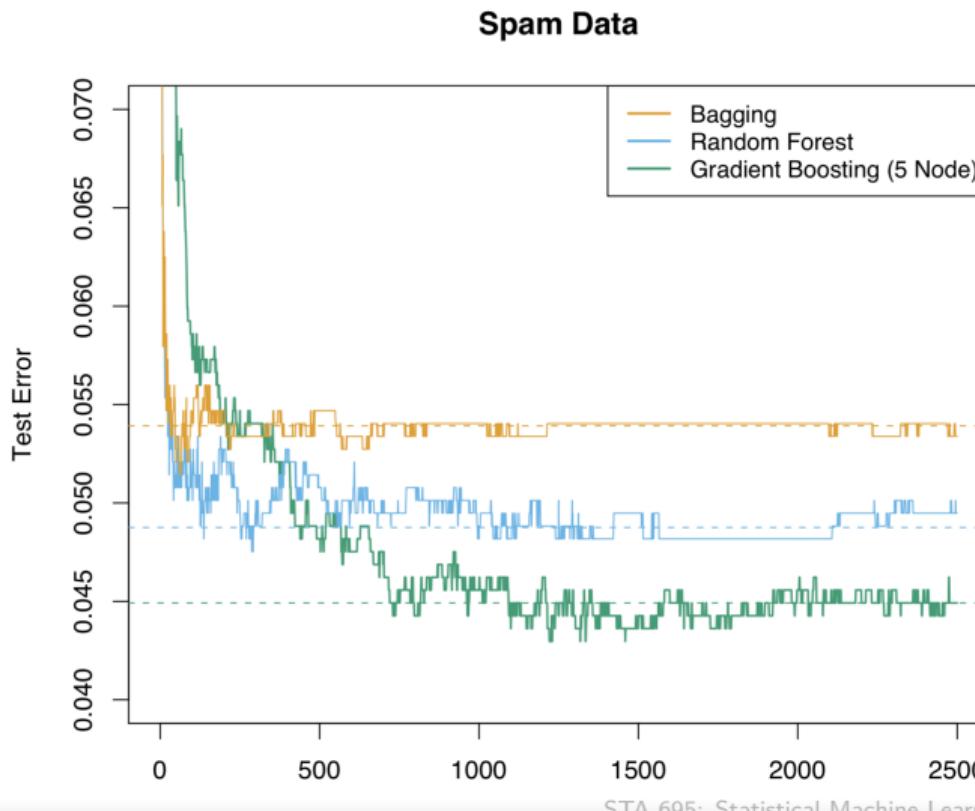
Real AdaBoost

1. Start with weights $w_i = 1/N$, $i = 1, 2, \dots, N$.
 2. Repeat for $m = 1, 2, \dots, M$:
 - (a) Fit the classifier to obtain a class probability estimate $p_m(x) = \hat{P}_w(y = 1|x) \in [0, 1]$, using weights w_i on the training data.
 - (b) Set $f_m(x) \leftarrow \frac{1}{2} \log p_m(x)/(1 - p_m(x)) \in R$.
 - (c) Set $w_i \leftarrow w_i \exp[-y_i f_m(x_i)]$, $i = 1, 2, \dots, N$, and renormalize so that $\sum_i w_i = 1$.
 3. Output the classifier $\text{sign}[\sum_{m=1}^M f_m(x)]$.
-

More about Boosting

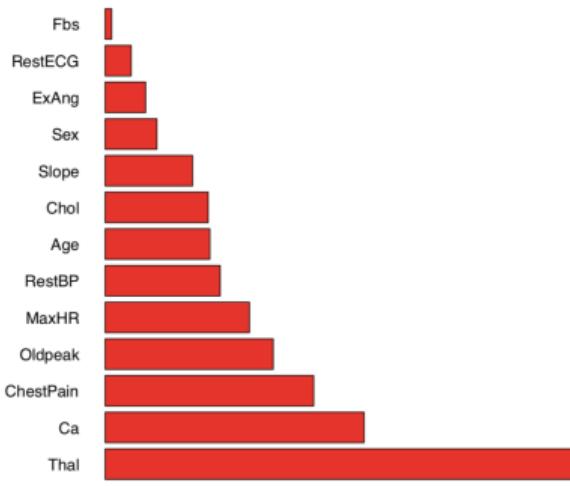
- Why stumps instead of trees?
 - Stumps: split on only 1 feature / one-level decision tree
 - High interpretability; Low complexity
- Can Boosting overfit? - Yes! but not easy with slow learners.
- Tuning Parameters: η and M (i.e., B) (Validation set or CV)
- Difficult interpretability
- Are trees independent? No, since we fit sequentially.
- Gradient Tree Boosting: 1. any loss function; 2. trees as base learners; 3. regression with nonlinear data.

Another classification example



Variable importance measure

- For bagged/RF regression trees, we record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor.
- Similarly, for bagged/RF classification trees, we add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.



Model Stacking

- Given predictions $\hat{f}_1(x), \hat{f}_2(x), \dots, \hat{f}_M(x)$, under squared-error loss, we can seek the weights $w = (w_1, w_2, \dots, w_M)$ such that

$$\hat{w} = \underset{w}{\operatorname{argmin}} E_{\mathcal{P}} \left[Y - \sum_{m=1}^M w_m \hat{f}_m(x) \right]^2.$$

- The full regression has smaller error than any single model

$$E_{\mathcal{P}} \left[Y - \sum_{m=1}^M \hat{w}_m \hat{f}_m(x) \right]^2 \leq E_{\mathcal{P}} \left[Y - \hat{f}_m(x) \right]^2 \forall m$$

Model Stacking

- Stacked generalization/stacking: Let $\hat{f}_m^{-i}(x)$ be the prediction at x , using model m , applied to the dataset with the i th training observation removed. The stacking estimate of the weights is obtained from the least squares linear regression of y_i on $\hat{f}_m^{-i}(x_i), m = 1, 2, \dots, M$.

$$\hat{w}^{\text{st}} = \underset{w}{\operatorname{argmin}} \sum_{i=1}^N \left[y_i - \sum_{m=1}^M w_m \hat{f}_m^{-i}(x_i) \right]^2.$$

- The final prediction is $\sum_m \hat{w}_m^{\text{st}} \hat{f}_m(x)$. By using the cross-validated predictions $\hat{f}_m^{-i}(x)$, stacking avoids giving unfairly high weight to models with higher complexity.

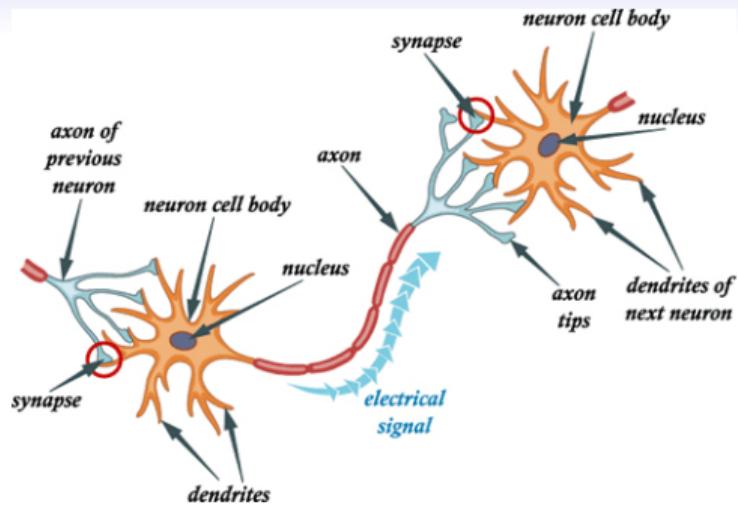
Summary

- Decision trees are simple and interpretable models for regression and classification
- However they are often not competitive with other methods in terms of prediction accuracy
- Bagging, random forests and boosting are good methods for improving the prediction accuracy of trees. They work by growing many trees on the training data and then combining the predictions of the resulting ensemble of trees.
- The latter two methods— random forests and boosting— are among the state-of-the-art methods for supervised learning. However their results can be difficult to interpret.

High-level overview

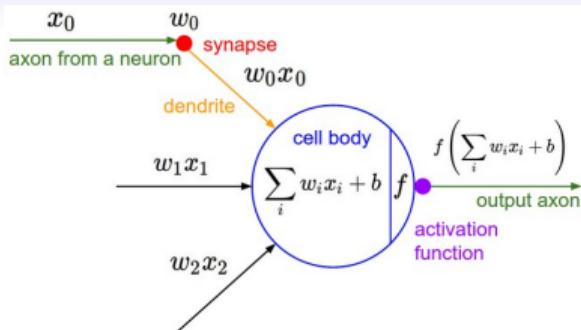
- **Neural Networks (NN)** are a family of functions: $y = g(\mathbf{x})$.
- NN is extremely flexible: it can approximate any continuous function arbitrarily well.
- NN can be used for **regression** and **classification** tasks.
- Historically, NN was inspired by modeling biological neural networks.
- Deep Learning is NN combined with modern learning techniques

Biological neural networks



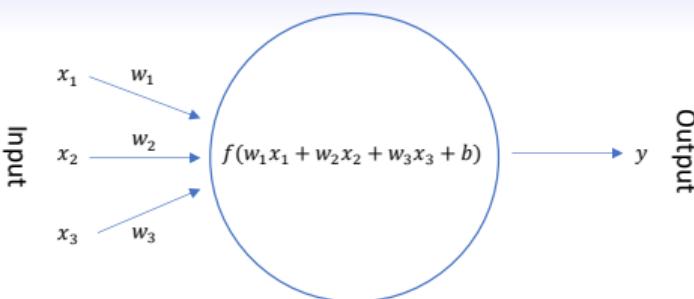
- Our neural system has ~ 86 billion neurons and they are connected with 10^{14} - 10^{15} synapses
- Neuron receives **input** signals from its dendrites and produces **output** signals along its axon
- The axon eventually branches out and connects via synapses to dendrites of other neurons.

Simplistic mathematical modeling of biological neural networks



- x_0, x_1, x_2 : input signals
- w_0, w_1, w_2 : weights or synaptic strengths. Excitatory (positive weight) or inhibitory (negative weight).
- If the weighted sum of the input signals is above certain threshold, the neuron fires and sends a spike along its axon.
- $f(\cdot)$: activation function which outputs the frequency of neuron firing.

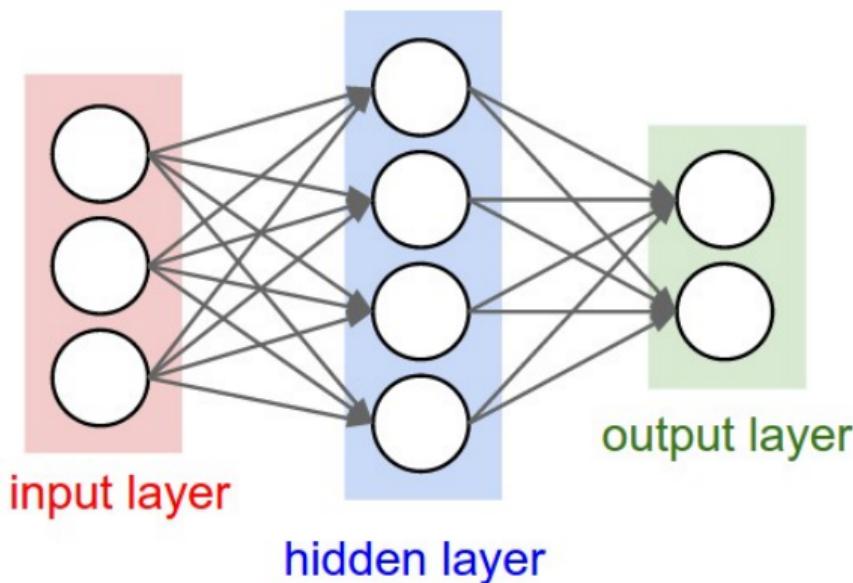
A Single Neuron



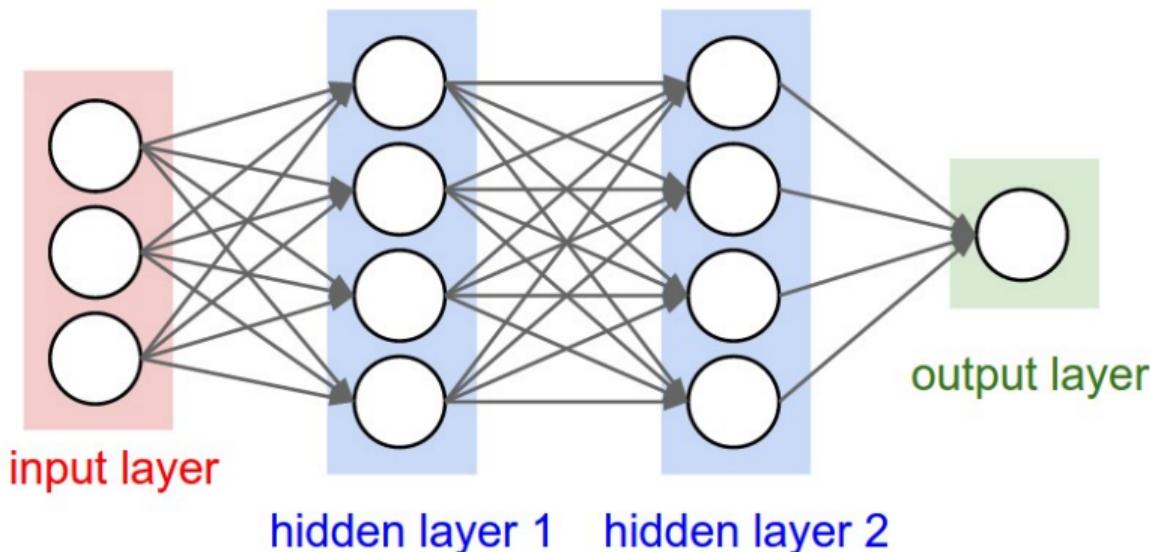
- Neuron is also known as a **node** or **unit**.
- x_1, x_2, x_3 are **inputs**.
- w_1, w_2, w_3 are **weights**. b is **bias**.
- $f(\cdot)$ is an **activation function**.
- y is an **output**.
- A neuron is simply a transformation of its inputs.

Feed-forward NN

A **feed-forward NN (also known as multilayer perceptron, MLP)** is a collection of neurons that are connected in a graph. In other words, the outputs of some neurons can become inputs to other neurons.



Three-layer NN



General Feed-forward NN

Notations:

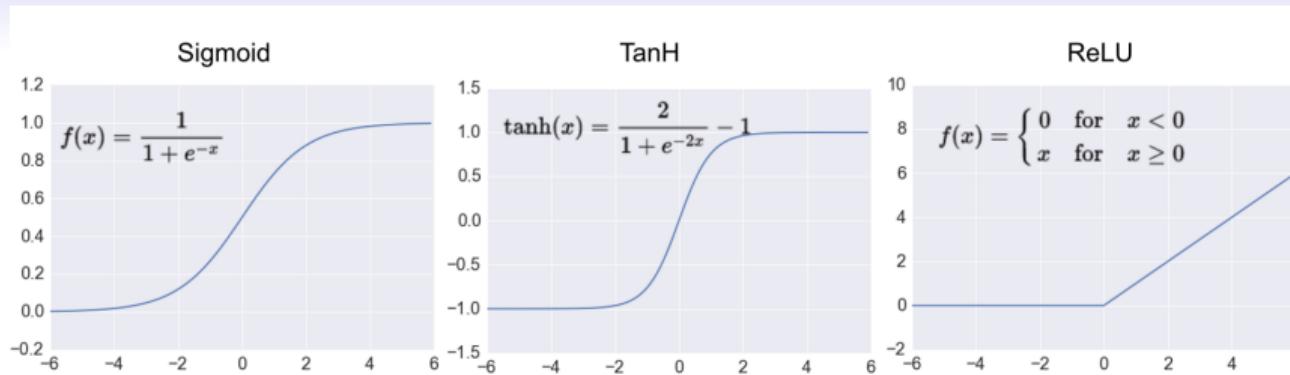
- L layers ($L - 1$ hidden layers)
- p_ℓ : number of nodes in layer $\ell = 0, \dots, L$ where p_0 is the number of input variables and p_L is the number of output variables. In this course, we've only considered $p_L = 1$, i.e. univariate response variable.
- $h^{(\ell)}$ is a p_ℓ -dim vector of nodes in layer ℓ . $h^{(0)} = (x_1, \dots, x_{p_0})^T$ is the set of inputs and $h^{(L)} = (y_1, \dots, y_{p_L})^T$ is the set of outputs.
- $b^{(\ell)}$ is a p_ℓ -dim vector of biases
- $W^{(\ell)}$ is a $p_{\ell-1} \times p_\ell$ weight matrix

Then a feed-forward NN is defined recursively for $\ell = 1, \dots, L$

$$h^{(\ell)} = f \left(W^{(\ell)T} h^{(\ell-1)} + b^{(\ell)} \right)$$

where the activation function $f(\cdot)$ is applied element by element. This is also known as **forward-propagation**.

Popular activation functions



- **Sigmoid:** takes a real-valued input and squashes it to range between 0 and 1
- **TanH:** takes a real-valued input and squashes it to the range [-1, 1]
- **ReLU (most popular):** ReLU stands for Rectified Linear Unit. It takes a real-valued input and thresholds it at zero (replaces negative values with zero).
- We only consider **nonlinear** activation functions because linear activation degenerates (linear transformation of linear transformation is still linear transformation).

NN Training

- **Regression:** find weights which minimize the **squared error loss**

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \frac{1}{2} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

where $\hat{y}_i = g(\mathbf{x}_i, \mathbf{w})$ is the output from the NN with input \mathbf{x}_i and weights \mathbf{w} .

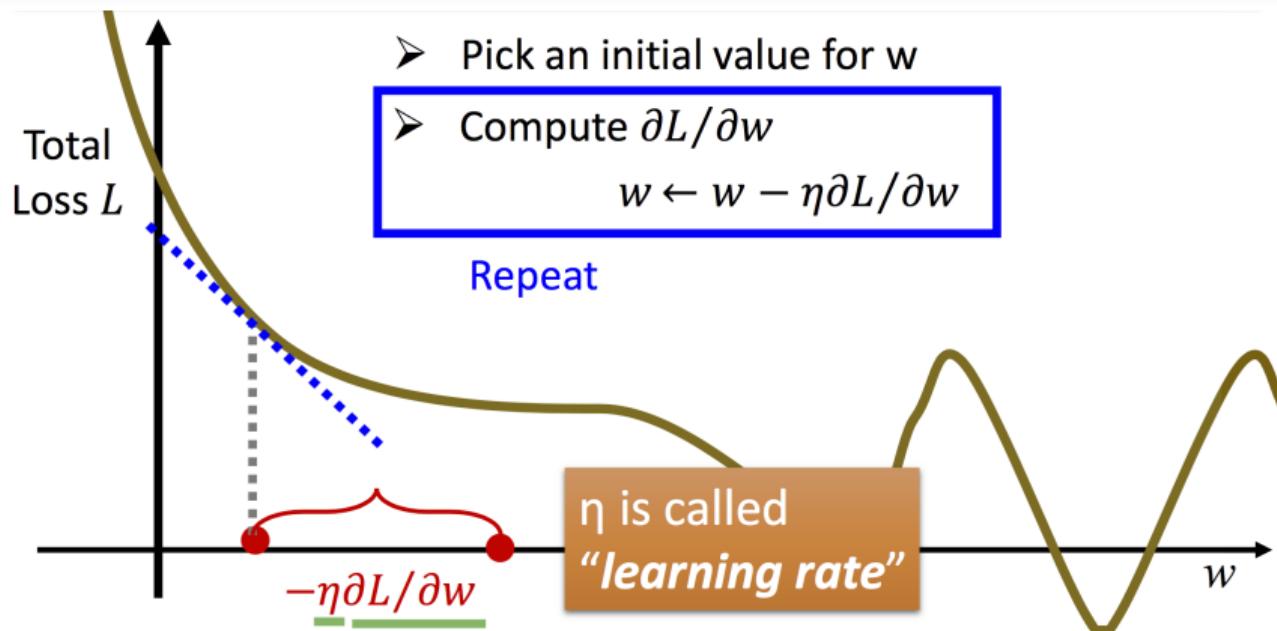
- **Binary classification:** find weights which minimize the **cross-entropy loss**

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} - \sum_{i=1}^n \{y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)\}$$

where $0 \leq \hat{y}_i \leq 1$ is the output from NN with sigmoid activation function in the last layer. \hat{y}_i can be interpreted as the probability of $y_i = 1$. Multi-class classification can be trained in a similar fashion (see PRML 5.2).

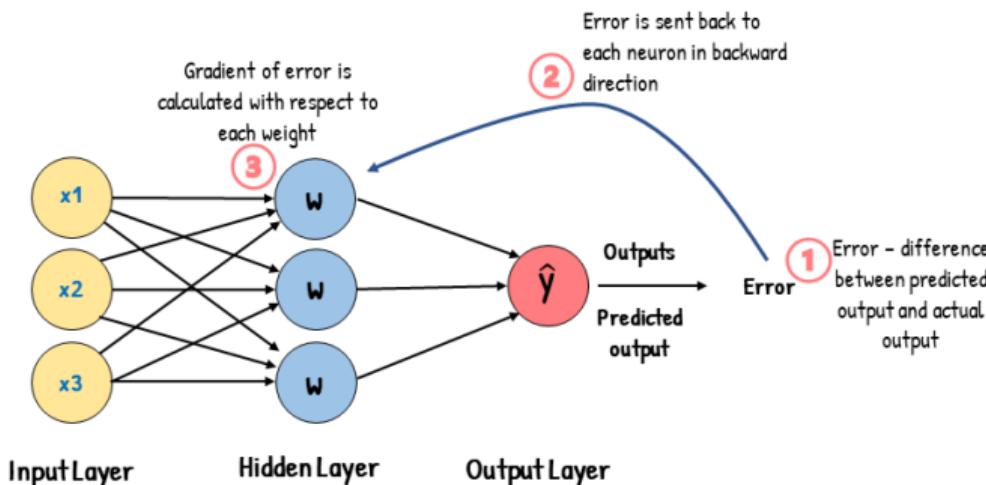
- To find $\hat{\mathbf{w}}$, we use **back-propagation**: (stochastic) gradient descent + chain rule.

Gradient descent

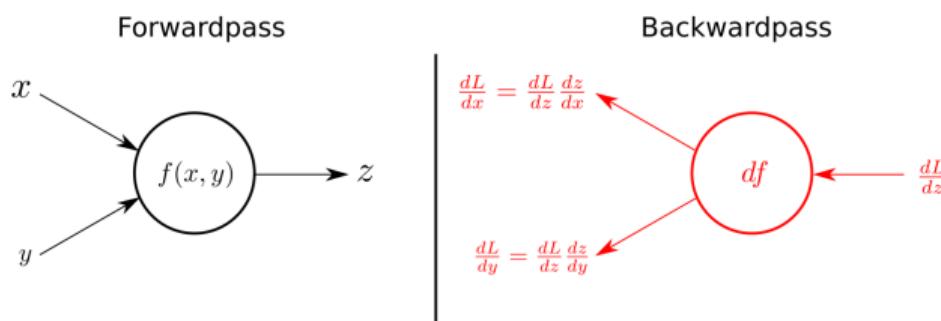


Back propagate

Backpropagation



Back propagate



Toolkits



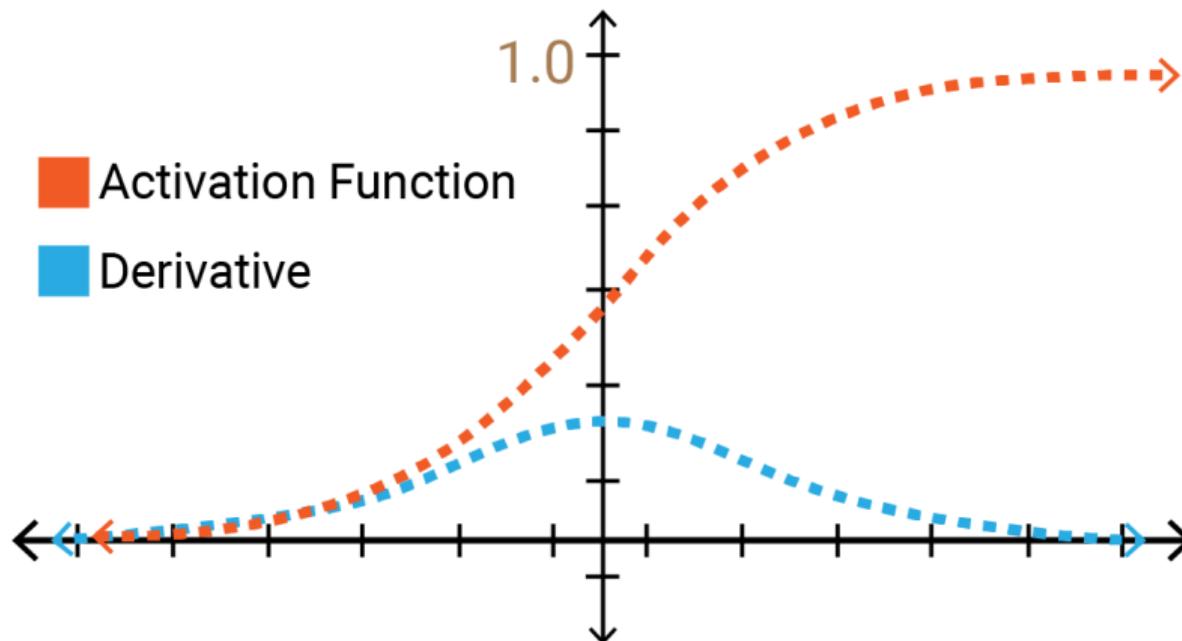
theano



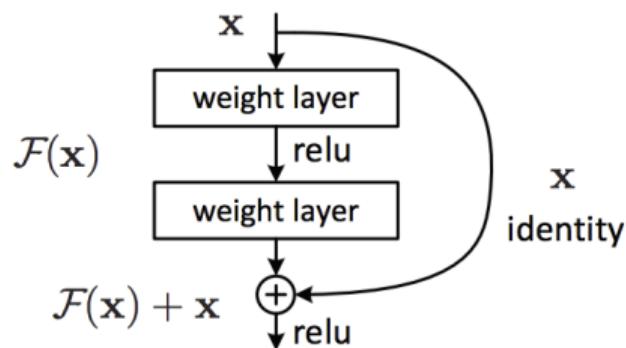
Caffe



Vanishing gradient problem



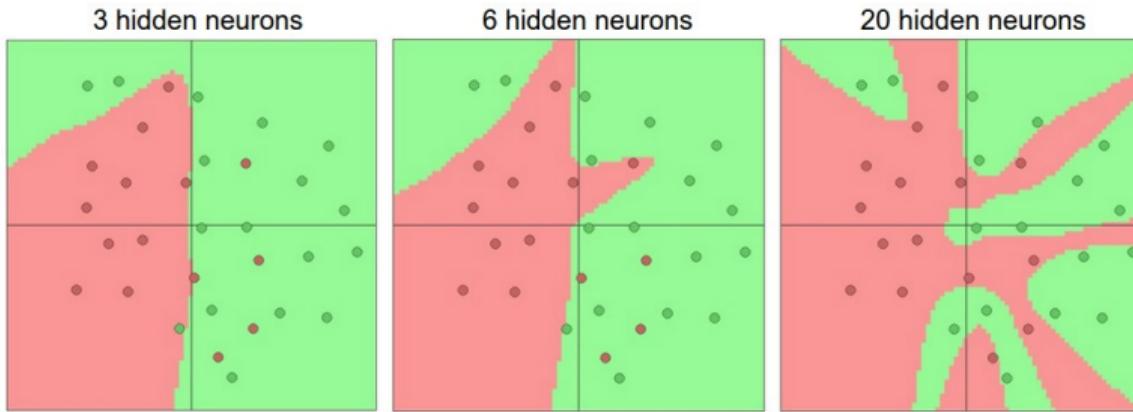
Residual learning



Most cited paper in AI: Deep Residual Learning for Image Recognition (~ 180000)!

How to choose number of layers and number of hidden units?

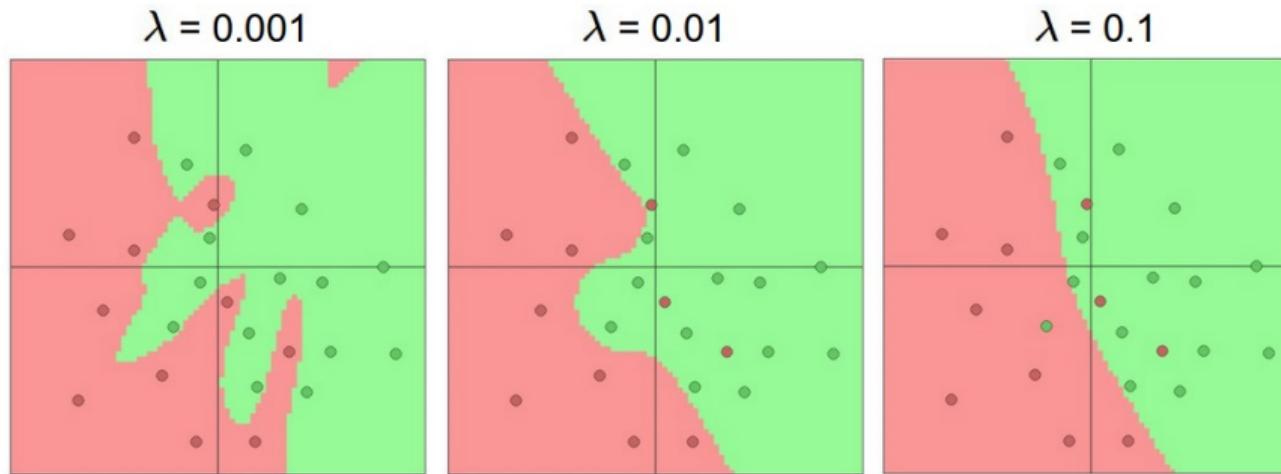
- More layers and hidden units increase the flexibility of the NN but tends to overfit.



- The model with one hidden layer and 20 hidden units fits all the training data but at the cost of segmenting the space into many disjoint red and green decision regions.
- Rule of thumb for simple problems:** choose 2 or 3 hidden layers and moderate to large number of hidden units and use regularization (e.g. ℓ_1/ℓ_2 regularization, dropout) to prevent overfitting.

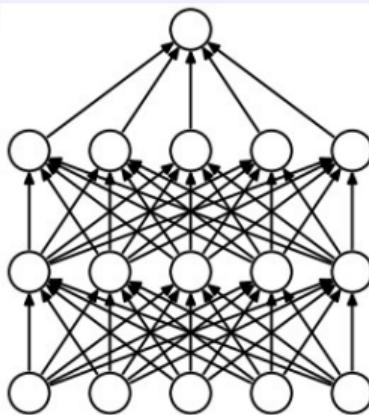
ℓ_2 regularization (weight decay)

- Add $\frac{1}{2}\lambda w^2$ to the loss/objective function for each weight w . λ is a tuning parameter that controls the strength of the regularization.

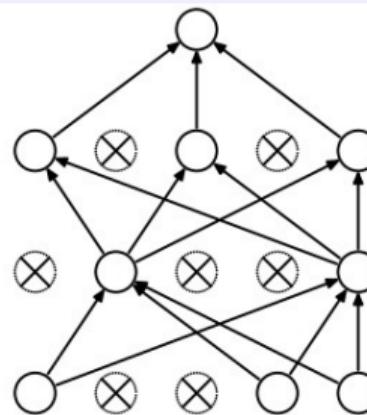


- Each neural network above has 20 hidden units, but changing the regularization strength makes its final decision regions smoother with a higher regularization.
- In practice, λ is chosen using cross-validation for simple problems; **AutoML**

Dropout



(a) Standard Neural Net



(b) After applying dropout.

- While training, dropout is implemented by only keeping a neuron active with some probability p
- During testing, there is no dropout applied with each weight is multiply by p .
- In practice, $p = 50\%$ works well.

Practical issues

- **Data preprocessing**

- Normalization: standardization or scale each variable
- Whitening: principal component analysis (will discuss later in this course)

- **Weight initialization**

- **DON'T** set all the initial weights to zero. If every neuron in the network computes the same output, then they will also all compute the same gradients during backpropagation and undergo the exact same parameter updates. In other words, there is no source of asymmetry between neurons if their weights are initialized to be the same.
- Instead, for a ReLU neuron with m inputs, draw $w_i \sim N(0, 2/m)$ for $i = 1, \dots, m$. This ensures that the input and the output have the same variance (also the same distribution). As a consequence, all neurons in the network initially have approximately the same output distribution and empirically improves the rate of convergence.

Batch normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

Pytorch example

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)  # 5*5 from image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square, you can specify with a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = torch.flatten(x, 1) # flatten all dimensions except the batch dimension
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net()
```

Pytorch example

```
output = net(input)
target = torch.randn(10) # a dummy target, for example
target = target.view(1, -1) # make it the same shape as output
criterion = nn.MSELoss()

loss = criterion(output, target)
print(loss)

net.zero_grad()      # zeroes the gradient buffers of all parameters

print('conv1.bias.grad before backward')
print(net.conv1.bias.grad)

loss.backward()

print('conv1.bias.grad after backward')
print(net.conv1.bias.grad)
```

Pytorch example

```
import torch.optim as optim

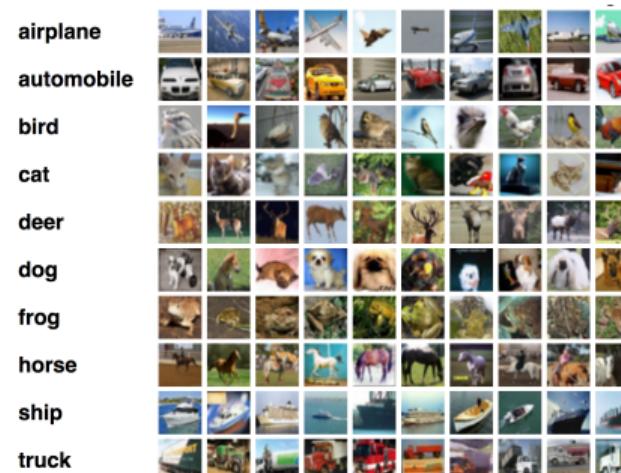
# create your optimizer
optimizer = optim.SGD(net.parameters(), lr=0.01)

# in your training loop:
optimizer.zero_grad()    # zero the gradient buffers
output = net(input)
loss = criterion(output, target)
loss.backward()
optimizer.step()          # Does the update
```

Convolutional Neural Networks

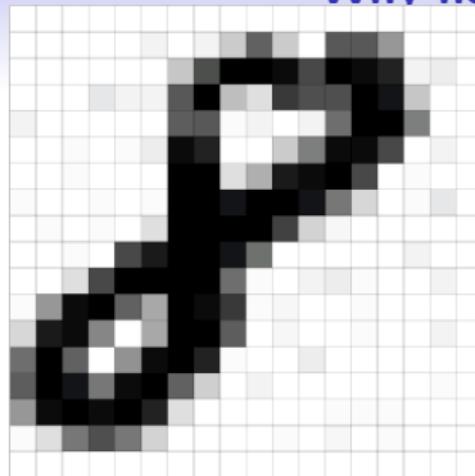
- **Convolutional Neural Networks (CNN or ConvNet)** is a NN specifically designed for image inputs.
- Very popular in computer vision and imaging analysis.
- Most common task is to classify images.

0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9



STA 695: Statistical Machine Learning and Predictive Modeling

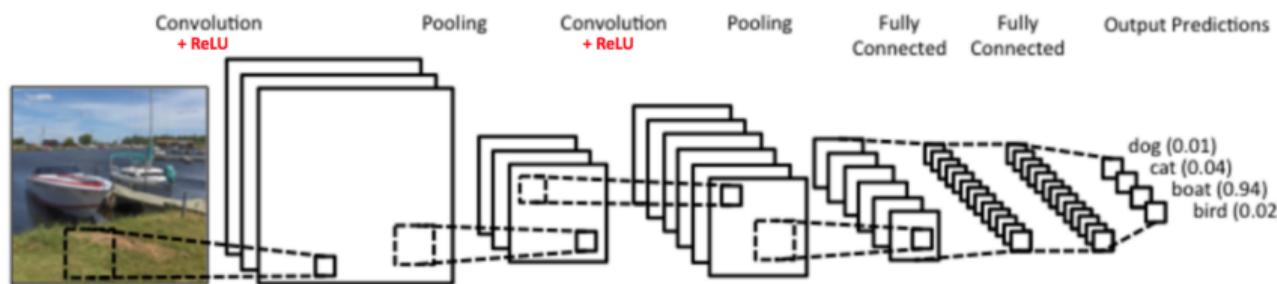
Why not use regular NN?



| | | | | | | | | | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 12 | 0 | 11 | 39 | 137 | 37 | 0 | 152 | 147 | 84 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 41 | 160 | 250 | 255 | 235 | 162 | 255 | 238 | 206 | 11 | 13 | 0 | 0 | 0 |
| 0 | 0 | 0 | 16 | 9 | 9 | 150 | 251 | 45 | 21 | 184 | 159 | 154 | 255 | 233 | 40 | 0 | 0 | 0 | 0 |
| 18 | 0 | 0 | 0 | 0 | 0 | 145 | 146 | 3 | 10 | 0 | 11 | 124 | 253 | 255 | 187 | 0 | 0 | 0 | 0 |
| 0 | 0 | 3 | 0 | 4 | 15 | 236 | 216 | 0 | 0 | 38 | 109 | 247 | 248 | 169 | 0 | 11 | 0 | 0 | 0 |
| 1 | 0 | 2 | 0 | 0 | 0 | 253 | 253 | 23 | 62 | 224 | 241 | 255 | 164 | 0 | 5 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 4 | 0 | 3 | 252 | 250 | 228 | 255 | 255 | 234 | 112 | 28 | 0 | 2 | 17 | 0 | 0 | 0 |
| 0 | 2 | 1 | 4 | 0 | 21 | 255 | 253 | 251 | 255 | 172 | 31 | 8 | 3 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 4 | 0 | 163 | 225 | 251 | 255 | 229 | 120 | 0 | 0 | 0 | 0 | 0 | 11 | 0 | 0 | 0 | 0 |
| 0 | 0 | 21 | 162 | 255 | 255 | 254 | 254 | 255 | 126 | 6 | 0 | 10 | 14 | 6 | 0 | 0 | 9 | 0 | 0 |
| 3 | 79 | 242 | 255 | 141 | 66 | 255 | 245 | 189 | 7 | 8 | 0 | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 26 | 221 | 237 | 98 | 0 | 67 | 251 | 255 | 144 | 0 | 8 | 0 | 0 | 7 | 0 | 0 | 11 | 0 | 0 | 0 |
| 125 | 255 | 141 | 0 | 87 | 244 | 255 | 268 | 3 | 0 | 0 | 13 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 145 | 248 | 228 | 116 | 235 | 255 | 141 | 34 | 0 | 11 | 0 | 1 | 0 | 0 | 0 | 1 | 3 | 0 | 0 | 0 |
| 85 | 237 | 253 | 246 | 255 | 210 | 21 | 1 | 0 | 1 | 0 | 0 | 6 | 2 | 4 | 0 | 0 | 0 | 0 | 0 |
| 6 | 23 | 112 | 157 | 114 | 32 | 0 | 0 | 0 | 0 | 2 | 0 | 8 | 0 | 7 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

- A 2D color image is a 3 dimensional array ($\text{width} \times \text{height} \times \text{depth}$) of pixel values.
Depth=3 corresponding to 3 color channels (red, green, blue).
- A greyscale image is a matrix ($\text{width} \times \text{height}$) of pixel values.
- Each pixel value is an input.
- A small 200×200 color image would lead to $200 \times 200 \times 3 = 120,000$ weights for each hidden unit.

ConvNet Architecture



Four building blocks

- ① Convolution
- ② ReLU
- ③ Pooling
- ④ Fully Connected Layer (MLP)

Convolution

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

5×5 image

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

3×3 filter

Convolution: continued

What's going on the last slide

We slide the orange matrix over our original image (green) by 1 pixel (also called **stride**) and for every position, we compute element wise multiplication (between the two matrices) and add the multiplication outputs to get the final integer which forms a single element of the output matrix (pink). Note that the 3×3 matrix “sees” only a part of the input image in each stride.

In ConvNet terminology, the 3×3 matrix is called a **filter** or kernel or feature detector and the matrix formed by sliding the filter over the image and computing the dot product is called the Convolved Feature or Activation Map or the **Feature Map**. It is important to note that filters acts as feature detectors from the original input image.

Different filter effects

Different filters can detect different features from an image, for example edges, curves, blobs of color etc.

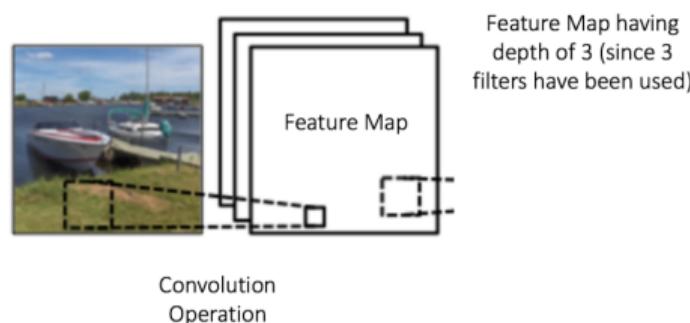
In practice, a ConvNet learns the values of these filters on its own during the training process

| | Operation | Filter | Convolved Image |
|----------------------------------|----------------|--|---|
| Identity | | $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |  |
| | | $\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$ |  |
| | Edge detection | $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ |  |
| | | $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ |  |
| Sharpen | | $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ |  |
| Box blur (normalized) | | $\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ |  |
| Gaussian blur (approximation) | | $\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ |  |

Feature Map

The size of the Feature Map is controlled by three parameters

- **Depth:** the number of filters used for the convolution operation.



- **Stride:** the number of pixels by which we slide our filter matrix over the input matrix. In practice, we rarely use stride greater than 2.
- **Zero-padding:** pad the input matrix with zeros (i.e. black margins) so that it allows us to control the size of the feature maps

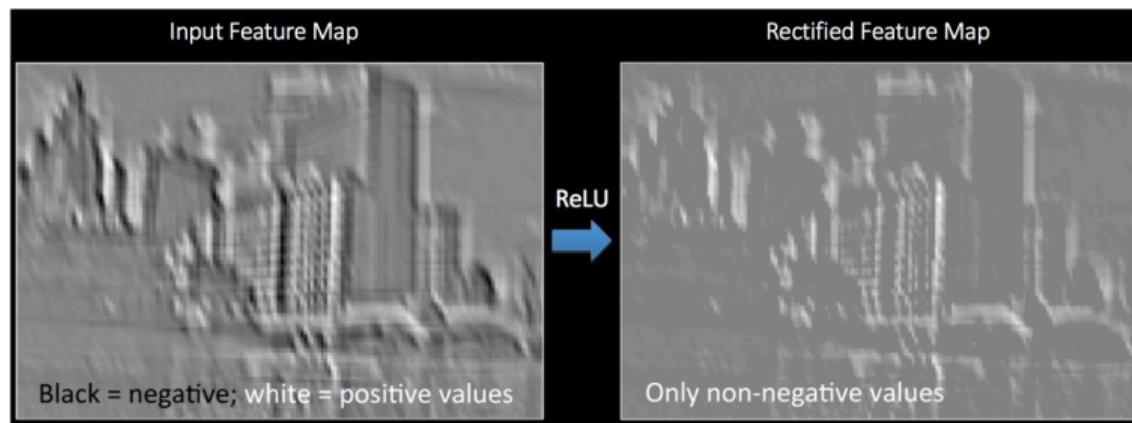
Example filters learned from ImageNet



- Each of the 96 filters shown here is of size $11 \times 11 \times 3$.

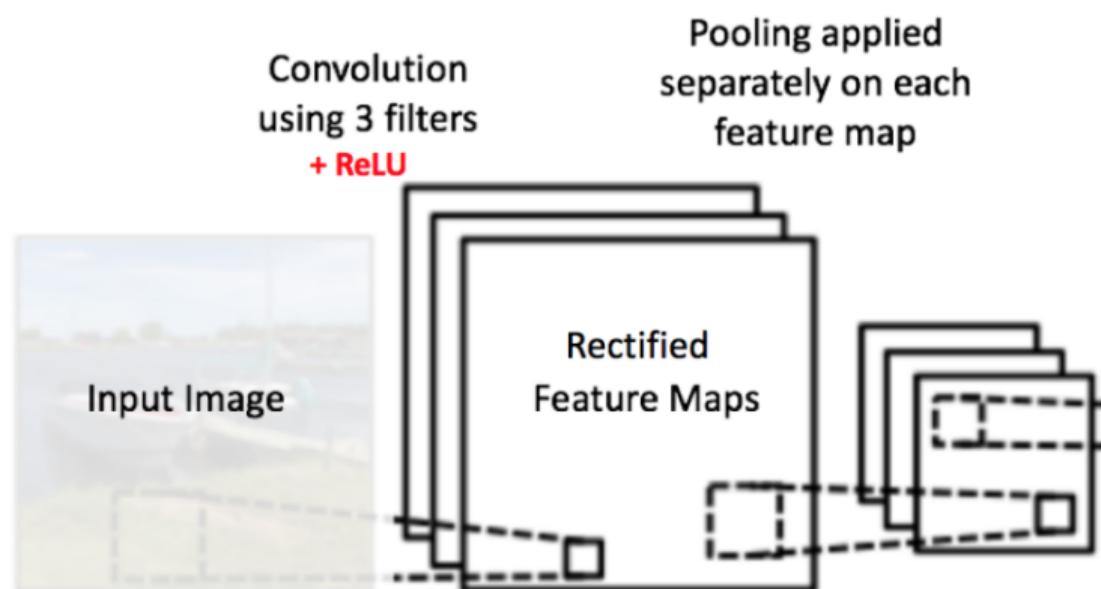
ReLU

- Apply ReLU (element wise) to the feature map to introduce non-linearity.



Max Pooling

- Max pooling progressively reduce the spatial size of each feature map while keeping the most important information. It reduces the amount of parameters and computation in the network, and hence to also control overfitting.



Max Pooling: illustration

- The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2. Every MAX operation would in this case be taking a max over 4 numbers.

Single depth slice

| | | | |
|---|---|---|---|
| 1 | 1 | 2 | 4 |
| 5 | 6 | 7 | 8 |
| 3 | 2 | 1 | 0 |
| 1 | 2 | 3 | 4 |

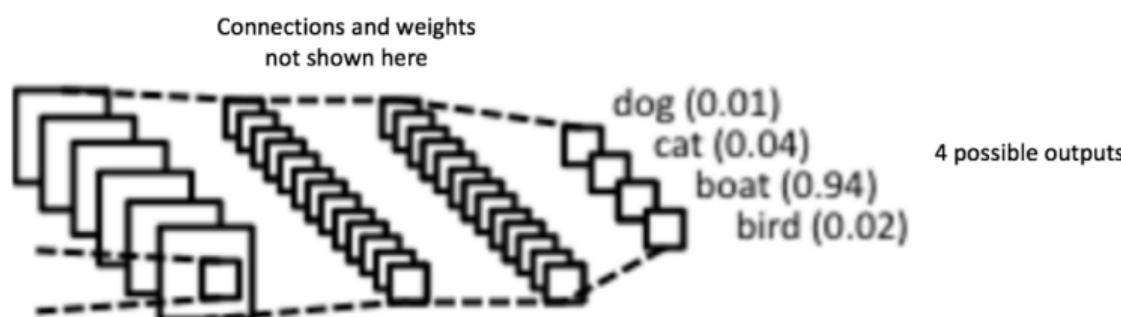
max pool with 2x2 filters
and stride 2



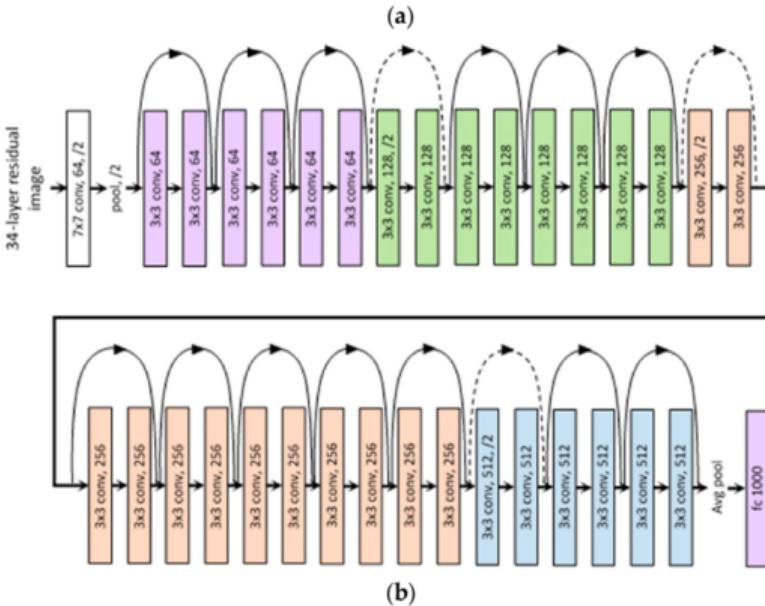
| | |
|---|---|
| 6 | 8 |
| 3 | 4 |

Fully Connected Layer

- The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset.



Resnet



Unsupervised Learning

- We only observe the variables or features X_1, X_2, \dots, X_p
- There is no response Y or class labels
- This generally means analysis goals are not clearly defined and we cannot directly validate any findings
- However, there are still many important questions that we can consider. Is there an informative way to visualize the data? Can we discover subgroups among the variables or among the observations? Is there any hidden pattern of the data?

Unsupervised Learning

- We only observe the variables or features X_1, X_2, \dots, X_p
- There is no response Y or class labels
- This generally means analysis goals are not clearly defined and we cannot directly validate any findings
- However, there are still many important questions that we can consider. Is there an informative way to visualize the data? Can we discover subgroups among the variables or among the observations? Is there any hidden pattern of the data?

Examples:

- groups of shoppers characterized by their browsing and purchase histories
- reduce the dimensionality

Unsupervised Learning

- We only observe the variables or features X_1, X_2, \dots, X_p
- There is no response Y or class labels
- This generally means analysis goals are not clearly defined and we cannot directly validate any findings
- However, there are still many important questions that we can consider. Is there an informative way to visualize the data? Can we discover subgroups among the variables or among the observations? Is there any hidden pattern of the data?

Examples:

- groups of shoppers characterized by their browsing and purchase histories
- reduce the dimensionality

We are going to discuss

- Clustering
- PCA

Clustering Methods

- **Clustering** refers to a very broad set of techniques for finding **subgroups**, or **clusters**, in a data set.
- We seek a partition of the data into distinct groups so that the observations within each group are quite similar to each other.
- To make this concrete, we must define what it means for two or more observations to be **similar** or **different**.
- Indeed, this is often a domain-specific consideration that must be made based on knowledge of the data being studied.

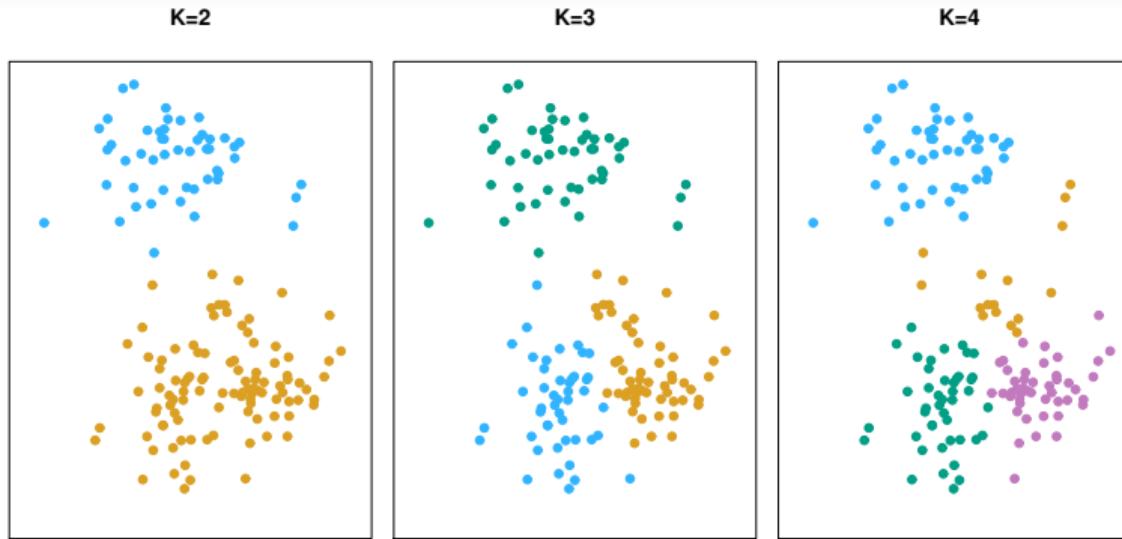
Example: Market Segmentation

- Suppose we have access to a large number of measurements (e.g. median household income, occupation, distance from nearest urban area, 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 receptive to a particular form of advertising, or more likely to purchase a particular product.
- The task of performing market segmentation amounts to clustering the people in the data set.

Clustering Methods

- **K-means clustering:** we seek to partition the observations into a pre-specified number of clusters.
- **Gaussian mixture models:** we look for soft cluster assignment in a probabilistic way so that the level of uncertainty over the most appropriate assignment can be quantified.
- **Density-based clustering:** we do not constrain the shape of the clusters to be ellipsoid or convex; instead, we use local density of points to determine the clusters whose shape can be arbitrary.
- **Hierarchical clustering:** we do not know in advance how many clusters we want; in fact, we end up with a tree-like visual representation of the observations, called a dendrogram, that allows us to view at once the clusterings obtained for each possible number of clusters, from 1 to n .

K-means clustering



A simulated data set with 150 observations in 2-dimensional space. Panels show the results of applying K-means clustering with different values of K , the number of clusters. The color of each observation indicates the cluster to which it was assigned using the K-means clustering algorithm.

Details of K-means clustering

Let C_1, \dots, C_K denote sets containing the indices of the observations in each cluster. These sets satisfy two properties:

- ① $C_1 \cup C_2 \cup \dots \cup C_K = \{1, \dots, n\}$. In other words, each observation belongs to at least one of the K clusters.
- ② $C_k \cap C_{k'} = \emptyset$ for all $k \neq k'$. In other words, the clusters are non-overlapping: no observation belongs to more than one cluster.

For instance, if the i th observation is in the k th cluster, then $i \in C_k$.

Details of K-means clustering: continued

- The idea behind K -means clustering is that a **good** clustering is one for which the **within-cluster variation** is as small as possible.
- The within-cluster variation for cluster C_k is a measure $\text{WCV}(C_k)$ of the amount by which the observations within a cluster differ from each other.
- Hence we want to solve the problem

$$\underset{C_1, \dots, C_K}{\text{minimize}} \left\{ \sum_{k=1}^K \text{WCV}(C_k) \right\}. \quad (2)$$

- In words, this formula says that we want to partition the observations into K clusters such that the total within-cluster variation, summed over all K clusters, is as small as possible.

How to define within-cluster variation?

- Typically we use Euclidean distance

$$\text{WCV}(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^p (x_{ij} - x_{i'j})^2, \quad (3)$$

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

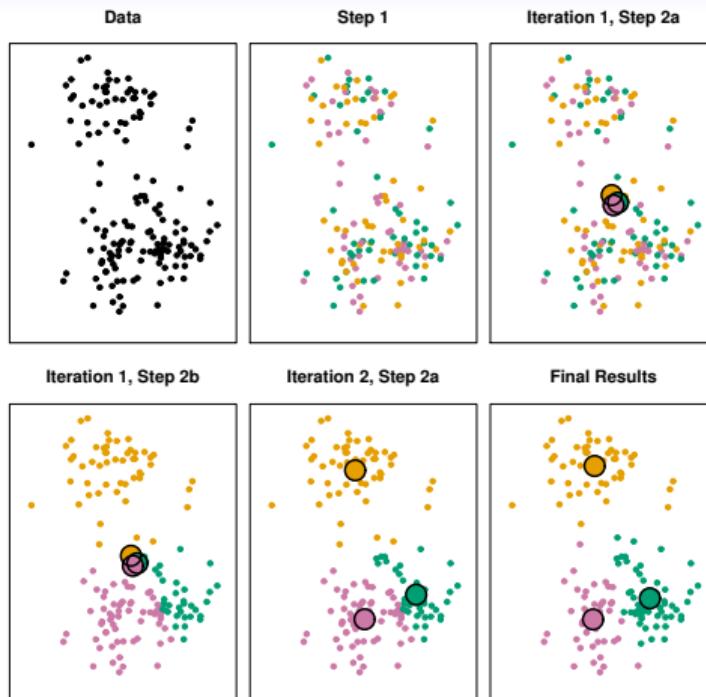
- Combining (2) and (3) gives the optimization problem that defines K -means clustering,

$$\underset{C_1, \dots, C_K}{\text{minimize}} \left\{ \sum_{k=1}^K \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^p (x_{ij} - x_{i'j})^2 \right\}. \quad (4)$$

K-Means Clustering Algorithm

- ① Randomly assign a number, from 1 to K , to each of the observations. These serve as initial cluster assignments for the observations.
- ② Iterate until the cluster assignments stop changing:
 - (a) For each of the K clusters, compute the cluster **centroid**. The k th cluster centroid is the vector of the p feature means for the observations in the k th cluster.
 - (b) Assign each observation to the cluster whose centroid is closest (where **closest** is defined using Euclidean distance).

Example



Properties of the Algorithm

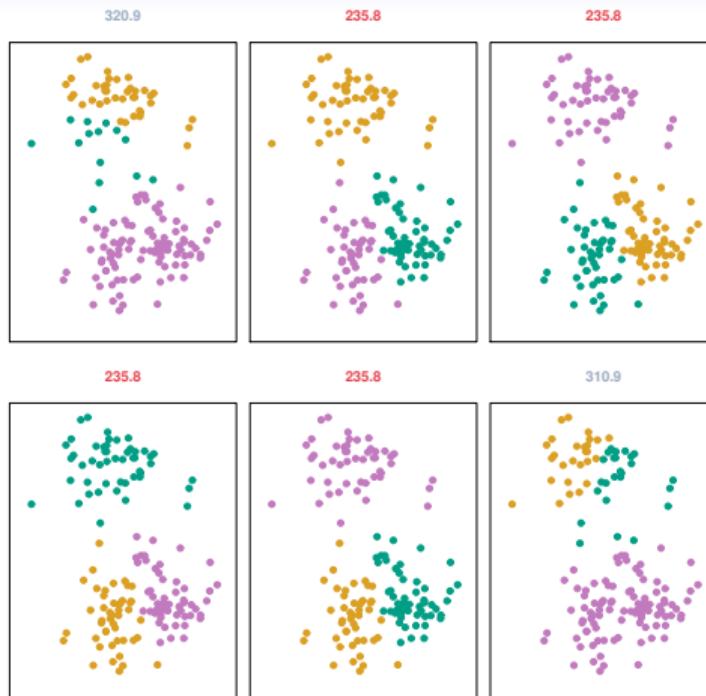
- This algorithm is guaranteed to decrease the value of the objective (4) at each step.
Why? Note that

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

where $\bar{x}_{kj} = \frac{1}{|C_k|} \sum_{i \in C_k} x_{ij}$ is the mean for feature j in cluster C_k .

- However, it is not guaranteed to give the global minimum since the objective is not convex. **We may want to run K-means multiple times with different initializations.**

Example: different starting values



Details of Previous Figure

K-means clustering performed six times on the data from previous figure with $K = 3$, each time with a different random assignment of the observations in Step 1 of the K-means algorithm.

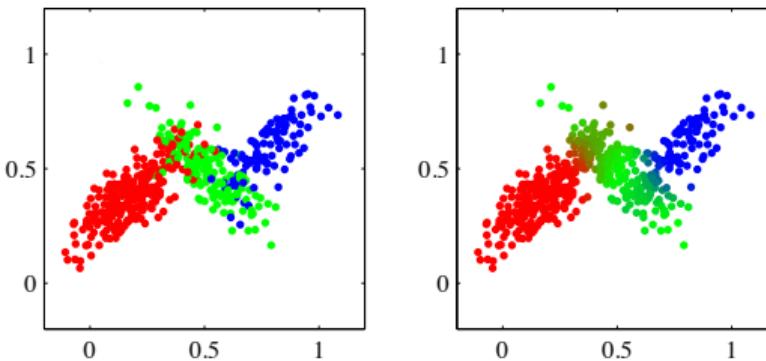
Above each plot is the value of the objective (4). Three different local optima were obtained, one of which resulted in a smaller value of the objective and provides better separation between the clusters. Those labeled in red all achieved the same best solution, with an objective value of 235.8

Mixture models

- K-means assigns each data point uniquely to one and only one cluster. Some data points may lie roughly midway between cluster centroids. It is not clear that the **hard** assignment to the nearest cluster is the most appropriate.
- **Mixture models** adopt a probabilistic approach and obtain **soft** assignments of data points to clusters in a way that reflects the level of **uncertainty** of cluster assignment.
- In this section, we introduce **Gaussian** mixture models, one of the most commonly used mixture models.

Gaussian mixture models: soft assignment

500 observations drawn from the mixture of three 2-dimensional Gaussians.



- Left: true labels indicated by red, green and blue.
- Right: soft assignment with proportions of red, blue, and green colors.

Gaussian mixture models: definition

Let $\mathbf{x}_i = (x_{i1}, \dots, x_{ip})$ and let $N(\cdot | \boldsymbol{\mu}, \boldsymbol{\Sigma})$ be a multivariate Gaussian density with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$. A Gaussian mixture model with K components is a weighted average of K Gaussian densities

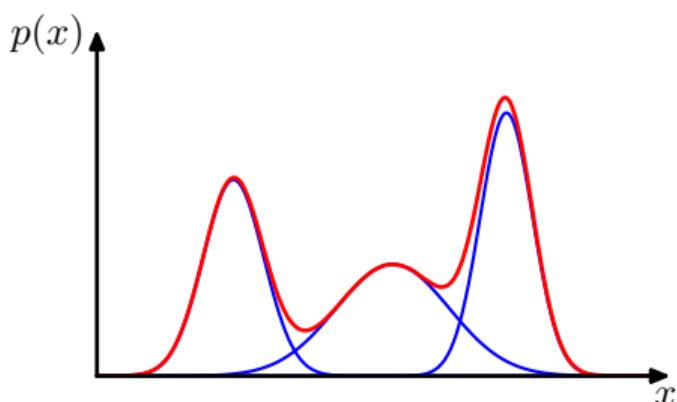
$$p(\mathbf{x}_i) = \sum_{k=1}^K \pi_k N(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

with

$$\sum_{k=1}^K \pi_k = 1$$

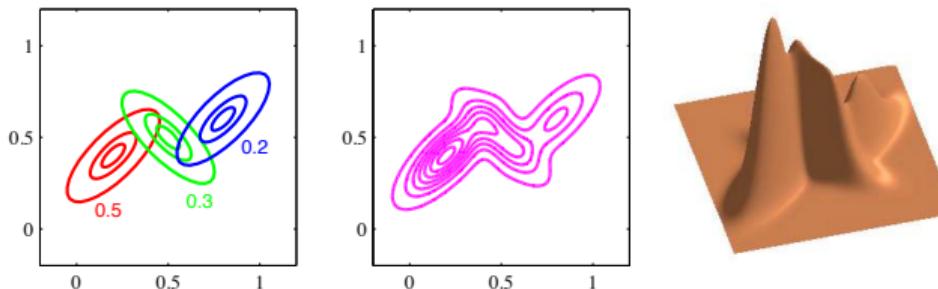
- $N(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ is a **component** of the mixture.
- $0 \leq \pi_k \leq 1$ is the **mixture weight** or **mixing coefficients**.

One-dimensional example



Three Gaussians (each scaled by a mixture weight) in blue and their sum in red.

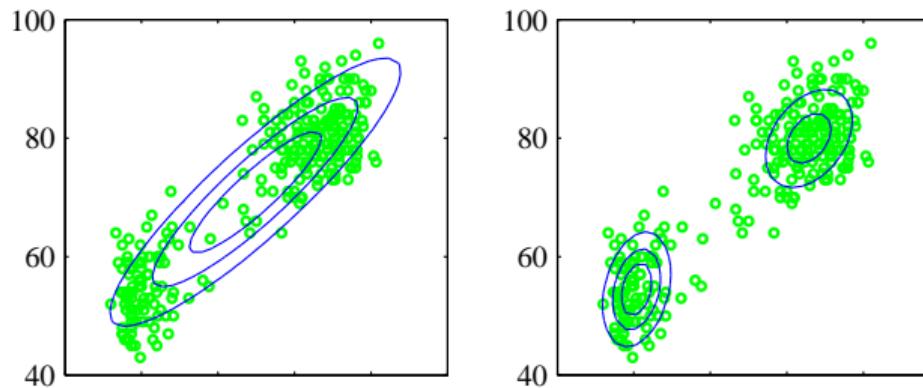
Two-dimensional example



- Left: contours of constant density for each of the 3 Gaussians (red, blue and green), and the mixture weights (numbers below each component).
- Center: contour of the weighted average of 3 Gaussians.
- Right: surface plot of the weighted average of 3 Gaussians.

Old faithful data

Old faithful geyser at Yellowstone. Eruption duration in minutes (x-axis) vs lapse time in minutes (y-axis)



- Left: a single Gaussian² distribution. This⁶ distribution fails to capture the two clumps in the data and indeed places much of its probability mass in the central region between the clumps where the data are relatively sparse.
- Right: two Gaussians which seem to fit better. **Each Gaussian forms a cluster!**

Latent variable representation

- To make the connection between Gaussian mixtures and clustering concrete, we introduce a latent variable $s_i \in \{1, \dots, K\}$ for each observation i and assume

$$p(s_i = k) = \pi_k$$

$$p(\mathbf{x}_i | s_i = k) = N(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

- The marginal distribution of (1) and (2)

$$p(\mathbf{x}_i) = \sum_{k=1}^K p(\mathbf{x}_i | s_i = k) p(s_i = k)$$

is equivalent to the weighted average representation.

- Interpretation:** $s_i = k$ indicates observation i belongs to cluster k .

Soft assignment through Bayes theorem

- Since we have a proper probability model, we can calculate conditional probability $p(s_i = k | \mathbf{x}_i)$ using Bayes theorem

$$p(s_i = k | \mathbf{x}_i) = \frac{\pi_k N(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{l=1}^K \pi_l N(\mathbf{x}_i | \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}$$

- Soft assignment:** $p(s_i = k | \mathbf{x}_i)$ is the posterior probability that observation i belongs to cluster k .

Estimation through EM algorithm

Expectation-maximization algorithm:

- ① Initialize the means μ_k , covariances Σ_k and mixture weights π_k
- ② E step. Evaluate the posterior probabilities using the current parameter values

$$p(s_i = k | \mathbf{x}_i) = \frac{\pi_k N(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{l=1}^K \pi_l N(\mathbf{x}_i | \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}$$

- ③ M step. Re-estimate the parameters using the posterior probabilities

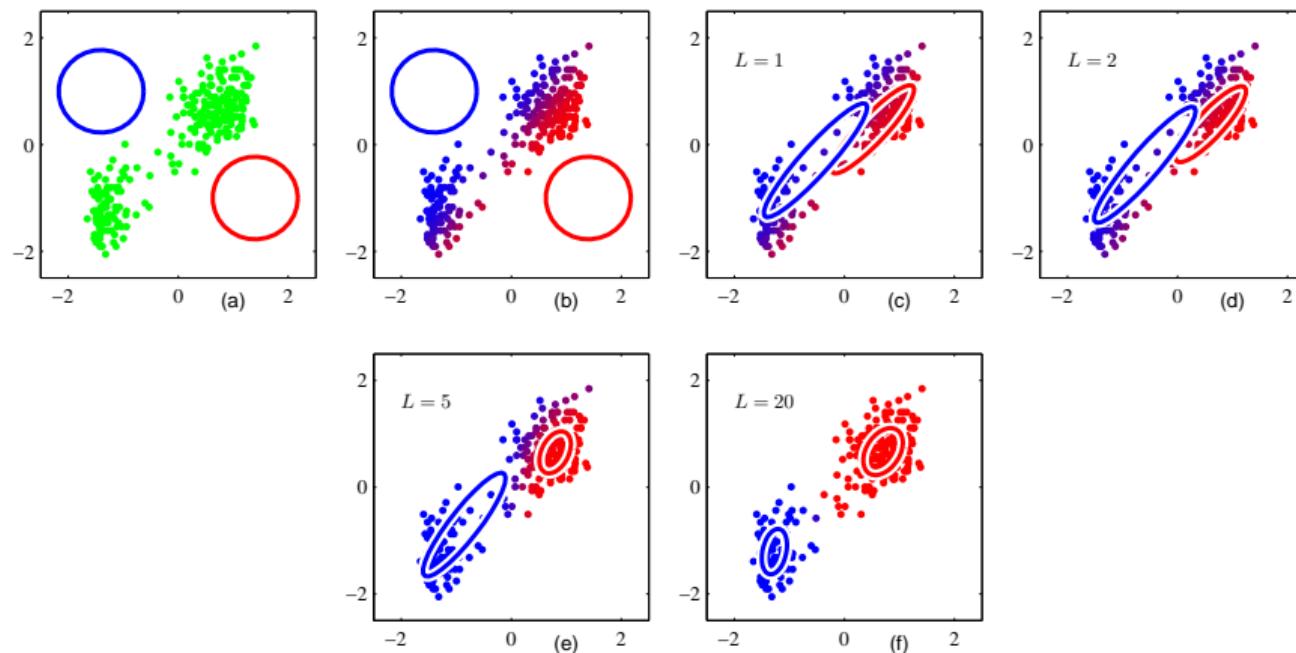
$$\boldsymbol{\mu}_k^{new} = \frac{1}{n_k} \sum_{i=1}^n p(s_i = k | \mathbf{x}_i) \mathbf{x}_i;$$

$$\boldsymbol{\Sigma}_k^{new} = \frac{1}{n_k} \sum_{i=1}^n p(s_i = k | \mathbf{x}_i) (\mathbf{x}_i - \boldsymbol{\mu}_k^{new})(\mathbf{x}_i - \boldsymbol{\mu}_k^{new})^T; \pi_k^{new} = \frac{n_k}{n}$$

where $n_k = \sum_{i=1}^n p(s_i = k | \mathbf{x}_i)$ is the effective number of observations assigned to cluster k .

- ④ Check convergence (e.g. through parameters or likelihood). If not convergent, return to step 2.

Illustration of EM algorithm



Comparison with K-means

- K-means outputs hard cluster assignment whereas Gaussian mixture model outputs soft cluster assignment.
- If $\Sigma_k = \epsilon I_p$ (variance times an identity matrix) for all $k = 1, \dots, K$, when ϵ is small, the soft assignment and hard assignment are similar

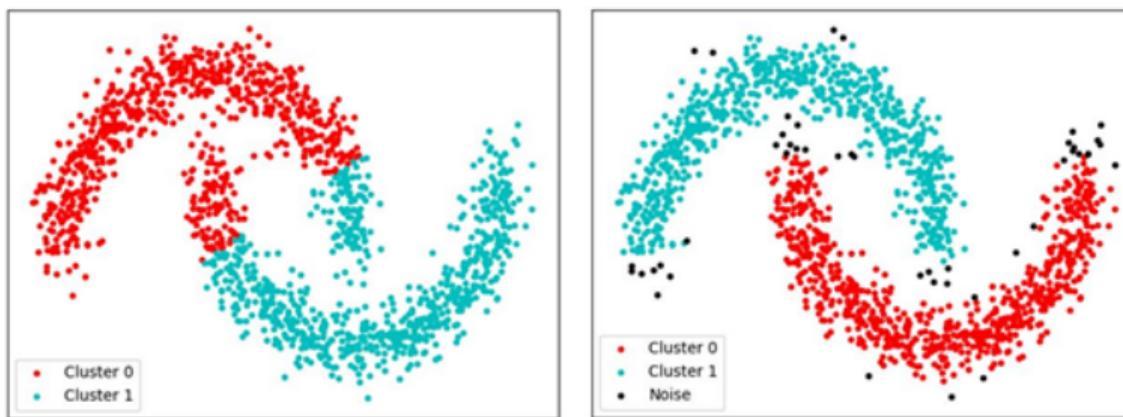
$$p(s_i = k | \mathbf{x}_i) \approx 1 \text{ for } k = \arg \min_k \sum_{j=1}^p (x_{ij} - \mu_{kj})^2$$
$$p(s_i = l | \mathbf{x}_i) \approx 0 \text{ for } l \neq k$$

Essentially, each observation is assigned to the cluster having the closest mean.

Density-based clustering

- K-means and Gaussian mixtures assume ellipsoidal/convex clusters.
- In practice, we can have irregular-shaped/**non-convex** clusters.
- Two observations from different clusters may be closer than two observations in the same cluster.
- **Density-based clustering** is able to discover non-convex clusters.
- Next, we introduce **density-based spatial clustering of applications with noise (DBSCAN)** algorithm.

Non-convex shaped clusters



Core, border and noise points

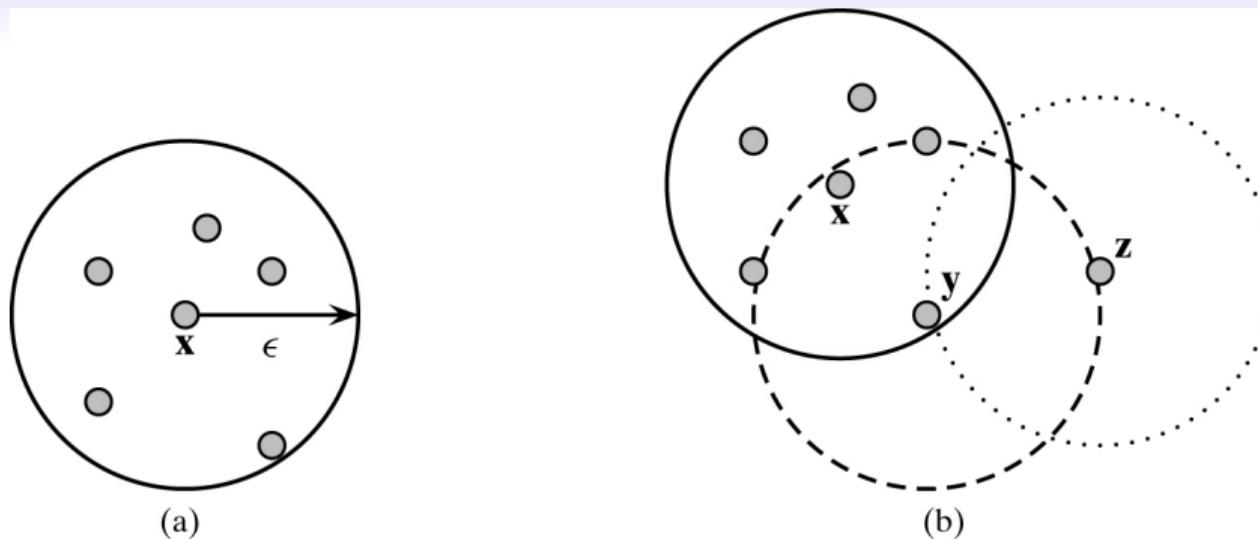
ϵ -neighborhood: a ball of radius ϵ around a point $\mathbf{x} = (x_1, \dots, x_p)$

$$N_\epsilon(\mathbf{x}) = \left\{ \mathbf{y} \middle| \sum_{j=1}^p (x_j - y_j)^2 \leq \epsilon \right\}$$

Core points: \mathbf{x} is a core point if $|N_\epsilon(\mathbf{x})| \geq m$, i.e. there are at least m points in its ϵ -neighborhood.

Border points: \mathbf{x} is a border point if it is not a core point and it belongs to the ϵ -neighborhood of some core point \mathbf{z} , that is, $\mathbf{x} \in N_\epsilon(\mathbf{z})$.

Noise points: neither a core nor a border point.



(a) ϵ -neighborhood of x . (b) Core point x , border point y , and noise point z with $m = 6$.

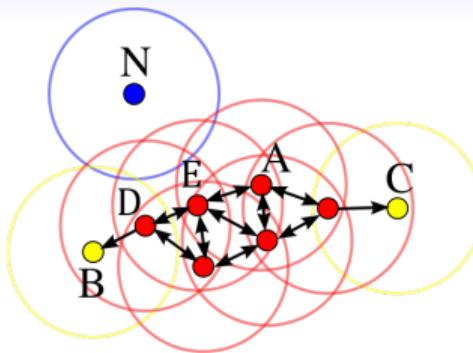
Reachability

Directly density reachable: x is directly density reachable from another point y if $x \in N_\epsilon(y)$ and y is a core point.

Density reachable: x is density reachable from y if there is set of core points leading from y to x , i.e. there exists a sequence of points $y = x_0, x_1, \dots, x_I = x$ such that x_i is directly density reachable from x_{i-1} for all $i = 1, \dots, I$.

Density connected: x and y are density connected if there exists a core point z such that both x and y are density reachable from z .

Illustration



$m = 4$. Point A and the other red points are core points, because the ϵ -neighborhood of these points contains at least 4 points (including the point itself). B (border point) is directly density reachable from D since B is in the ϵ -neighborhood of a core point D. B is density reachable from A because B is directly density reachable from D, D is directly density reachable from E, and E is directly density reachable from A. B and C are density connected since they are both density reachable from A. N is a noise point.

DBSCAN algorithm

A density-based cluster is defined as a maximal set of density connected points.

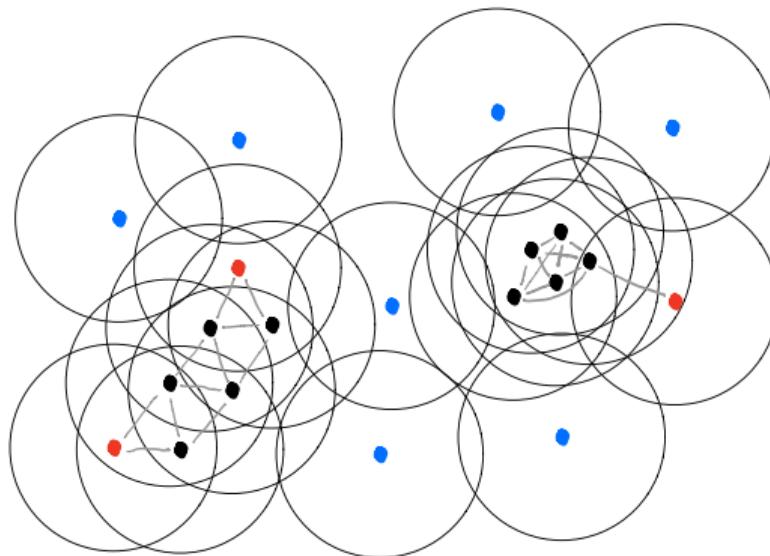
- ① Compute the ϵ -neighborhood $N_\epsilon(\mathbf{x}_i)$ for each observation \mathbf{x}_i and checks if it is a core point.
- ② For each core point, if it's not already assigned to a cluster, create a new cluster. Find recursively all its density connected points and assign them to the same cluster as the core point.
- ③ Observations that do not belong to any cluster are noise points.

In plain English..

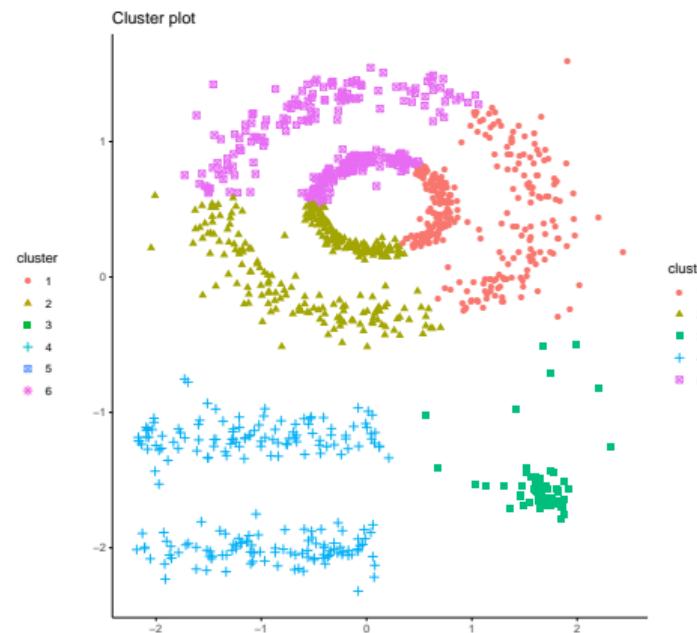
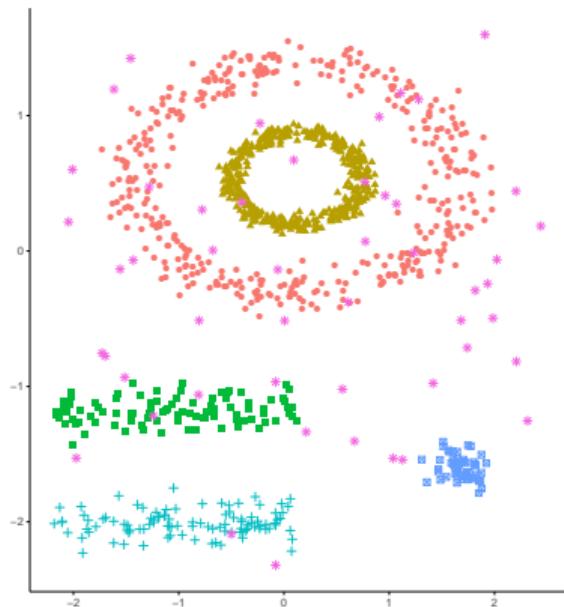
We begin by picking an arbitrary point in our dataset. If there are more than m points within a distance of ϵ from that point, (including the original point itself), we consider all of them to be part of a "cluster". We then expand that cluster by checking all of the new points and seeing if they too have more than m points within a distance of ϵ , growing the cluster recursively if so.

Eventually, we run out of points to add to the cluster. We then pick a new arbitrary point and repeat the process. Now, it's entirely possible that a point we pick has fewer than m points in its ϵ ball, and is also not a part of any other cluster. If that is the case, it's considered a noise point not belonging to any cluster.

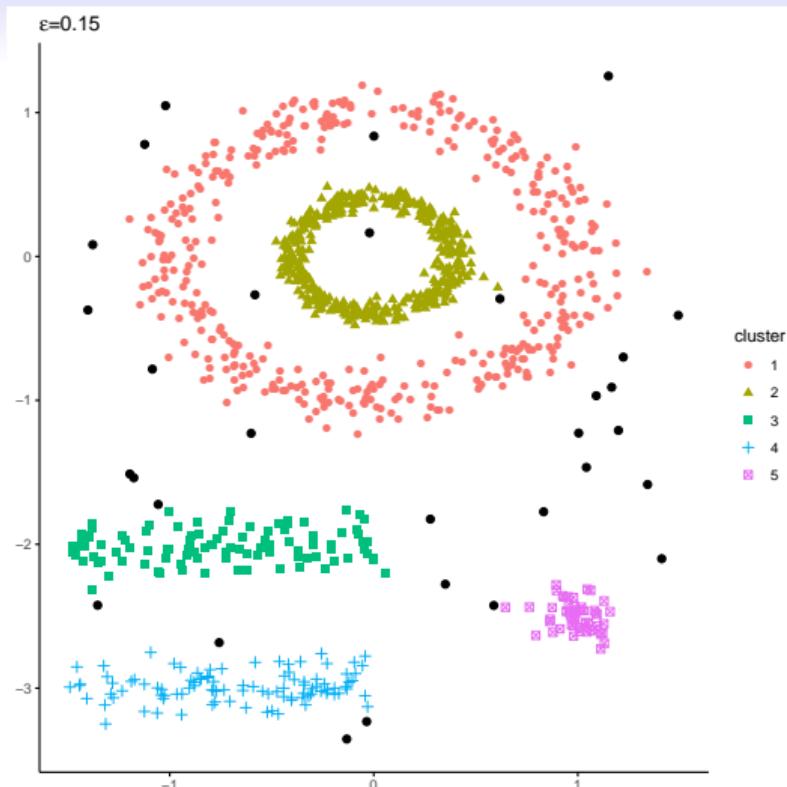
Illustration



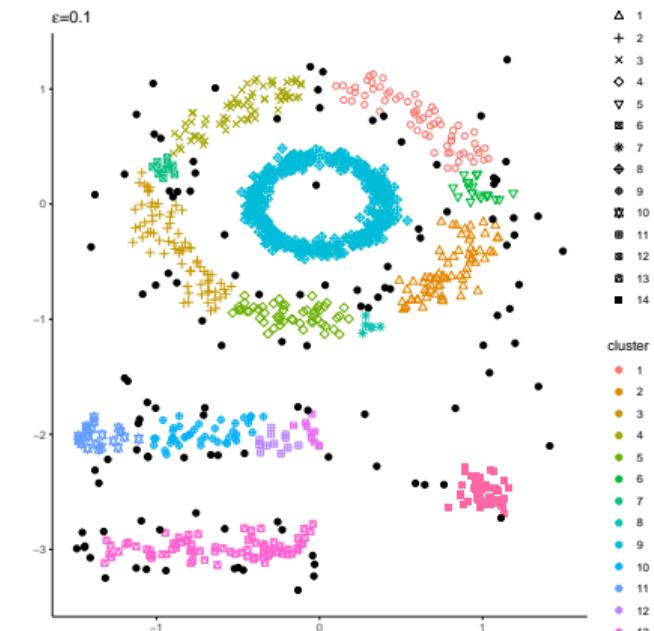
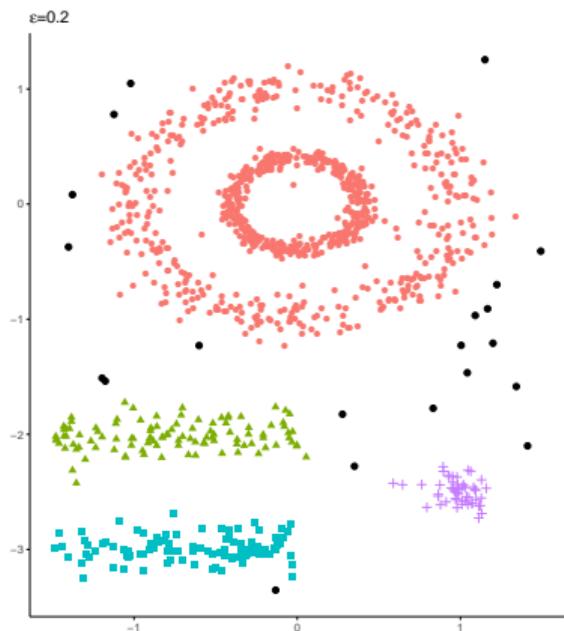
Simulated example



DBSCAN



But it's quite sensitive to the choice of ϵ



More examples of DBSCAN

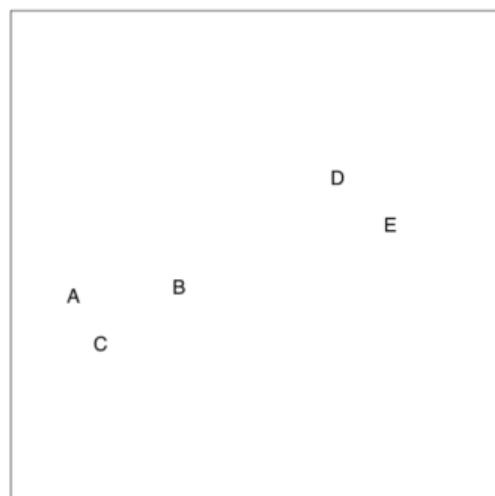
<https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/>

Hierarchical Clustering

- K-means clustering, mixture models and density-based clustering require us to **pre-specify** the number of clusters K . This can be a disadvantage.
- **Hierarchical clustering** is an alternative approach which does not require that we commit to a particular choice of K .
- In this section, we describe **bottom-up** or **agglomerative** clustering. This is the most common type of hierarchical clustering, and refers to the fact that a dendrogram is built starting from the leaves and combining clusters up to the trunk.

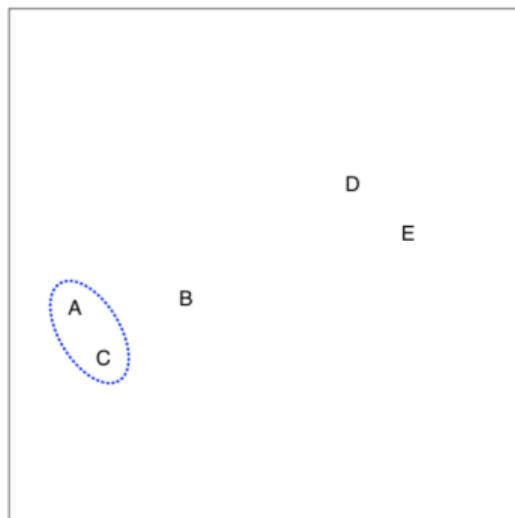
Hierarchical Clustering: the idea

Builds a hierarchy in a “bottom-up” fashion...



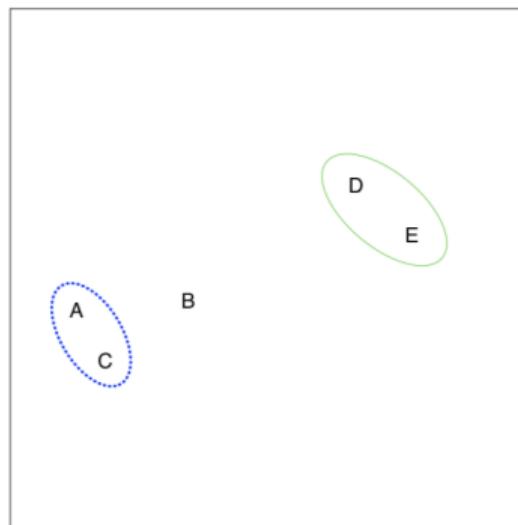
Hierarchical Clustering: the idea

Builds a hierarchy in a “bottom-up” fashion...



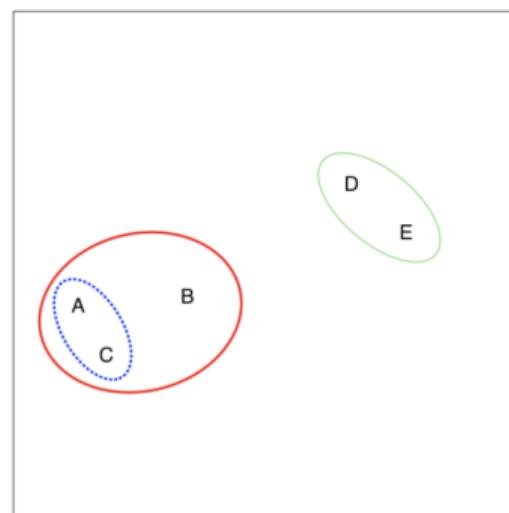
Hierarchical Clustering: the idea

Builds a hierarchy in a “bottom-up” fashion...



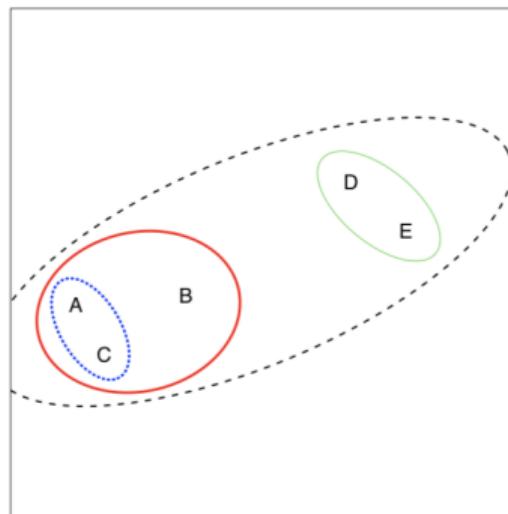
Hierarchical Clustering: the idea

Builds a hierarchy in a “bottom-up” fashion...



Hierarchical Clustering: the idea

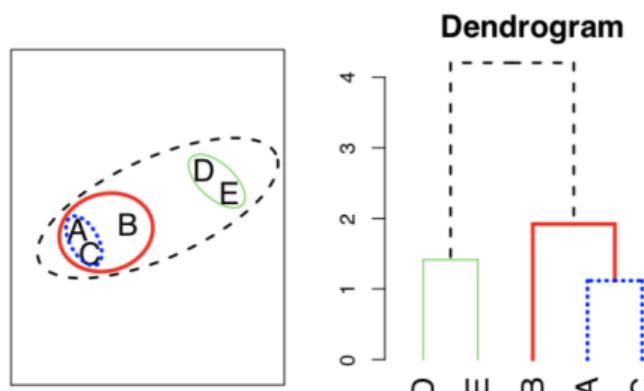
Builds a hierarchy in a “bottom-up” fashion...



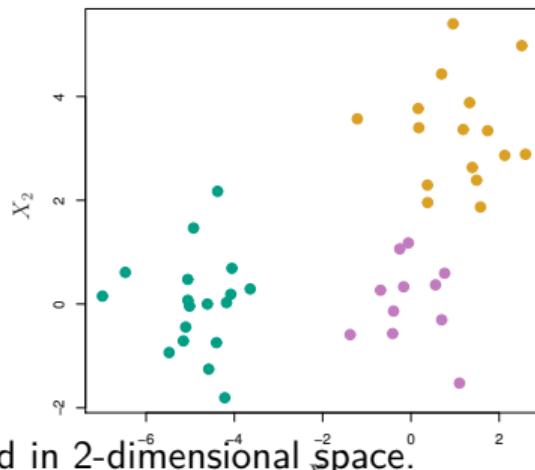
Hierarchical Clustering Algorithm

The approach in words:

- ① Start with each point in its own cluster.
- ② Identify the “closest” two clusters and merge them.
- ③ Repeat.
- ④ Ends when all points are in a single cluster.

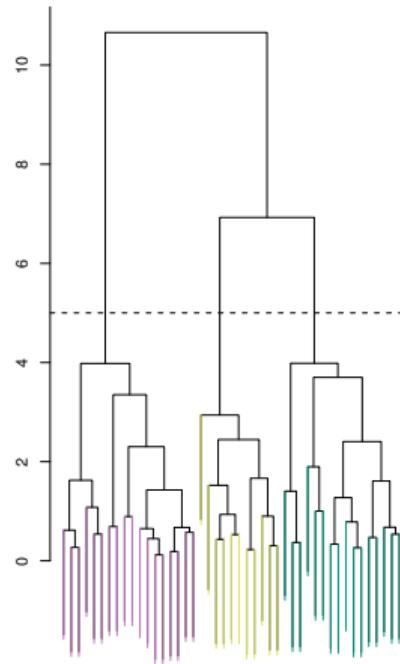
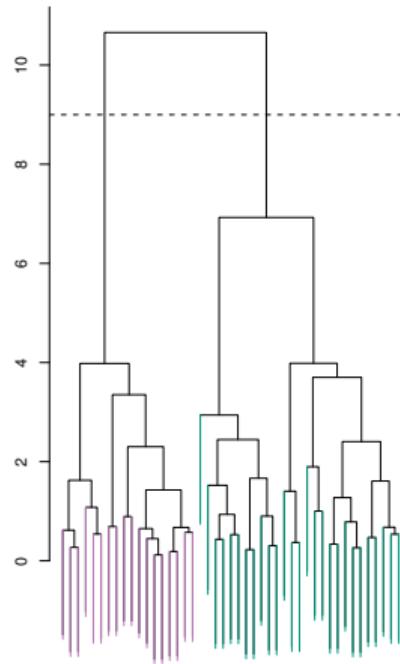
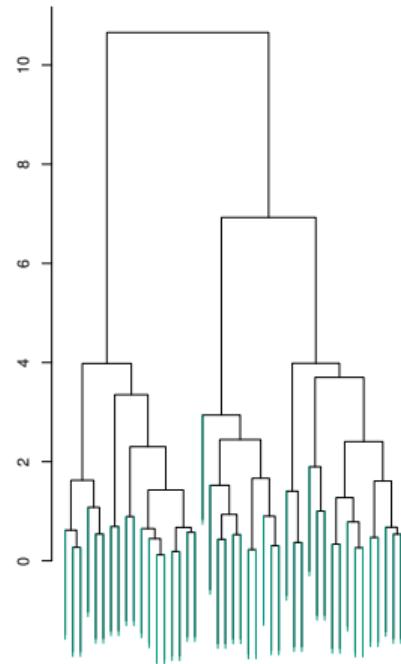


An Example



- 45 observations generated in 2-dimensional space.
- In reality there are three distinct classes, shown in separate colors.
- However, we will treat these class labels as unknown and will seek to cluster the observations in order to discover the classes from the data.

Application of hierarchical clustering



Details of previous figure

- Left: Dendrogram obtained from hierarchically clustering the data from previous slide, with “complete linkage” and Euclidean distance.
- Center: The dendrogram from the left-hand panel, cut at a height of 9 (indicated by the dashed line). This cut results in two distinct clusters, shown in different colors.
- Right: The dendrogram from the left-hand panel, now cut at a height of 5. This cut results in three distinct clusters, shown in different colors.

Common Linkages

Complete: Maximal inter-cluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the **largest** of these dissimilarities.

Single: Minimal inter-cluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the **smallest** of these dissimilarities.

Average: Mean inter-cluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the **average** of these dissimilarities.

Choice of Dissimilarity Measure

- So far we have used Euclidean distance.
- An alternative is correlation-based distance which considers two observations to be similar if their features are highly correlated.
- This is an unusual use of correlation, which is normally computed between variables; here it is computed between the observation profiles for each pair of observations.
- Other choices: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.distance_metrics.html#sklearn.metrics.pairwise.distance_metrics

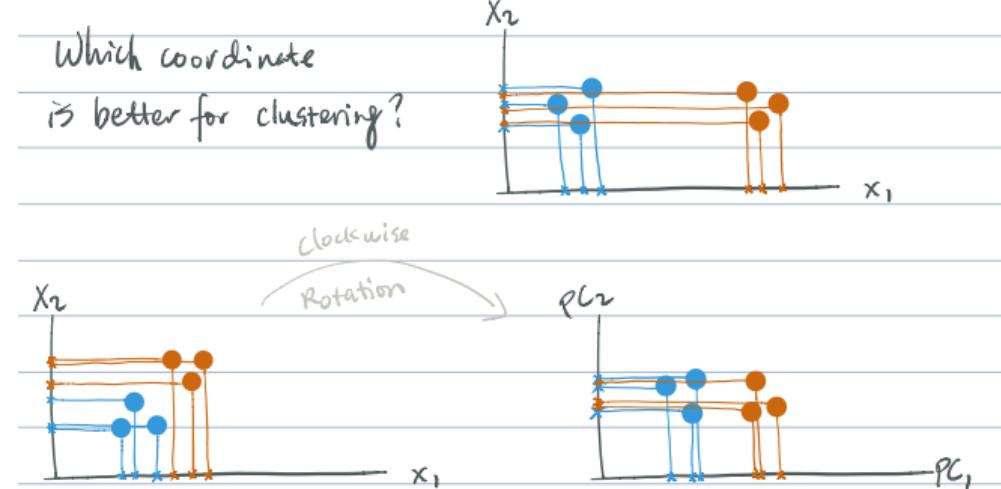
Practical issues in Clustering

- Should the observations or features first be standardized in some way? For instance, maybe the variables should be centered to have mean zero and scaled to have standard deviation one.
- In the case of hierarchical clustering,
 - What dissimilarity measure should be used?
 - What type of linkage should be used?
- How many clusters to choose? Difficult problem. No agreed-upon method. See Elements of Statistical Learning, section 14.3.11 for more details.

Principal component analysis (PCA)

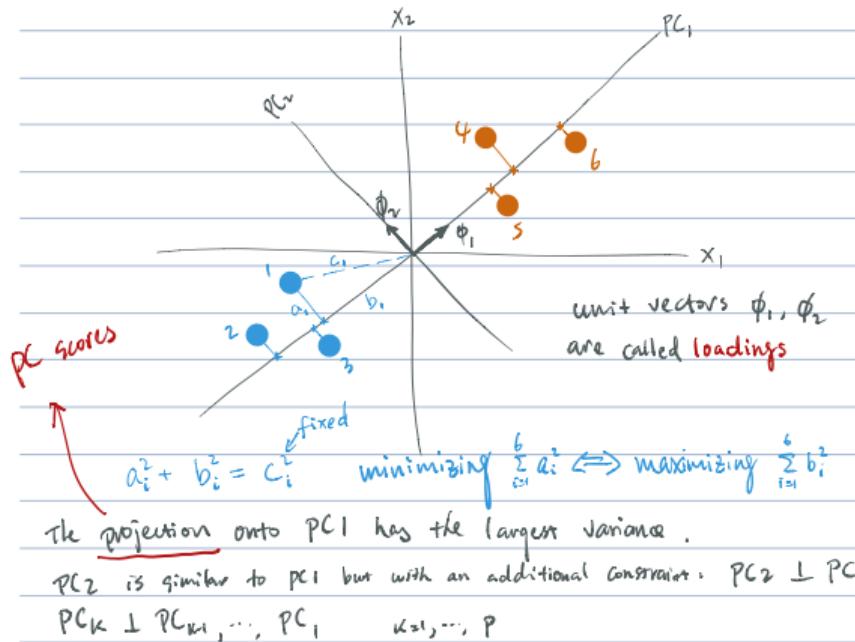
- Principal component analysis (PCA) produces a **low-dimensional** representation of a dataset. It finds a sequence of linear combinations of the variables that have **maximal variance**, and are mutually **uncorrelated**.
- Apart from producing derived variables for use in supervised learning problems, PCA also serves as a tool for data **visualization**.

PCA: continued



PCA: continued

PC_1 : line "closest" to all data points



PCA: details

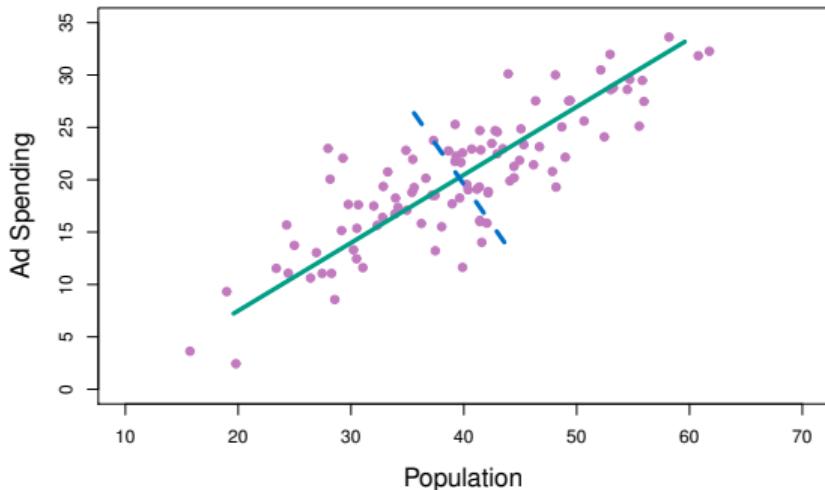
- The **first principal component** (PC_1) of a set of features X_1, \dots, X_p is the normalized linear combination of the features

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

that has the largest variance. By normalized, we mean that $\sum_{j=1}^p \phi_{j1}^2 = 1$.

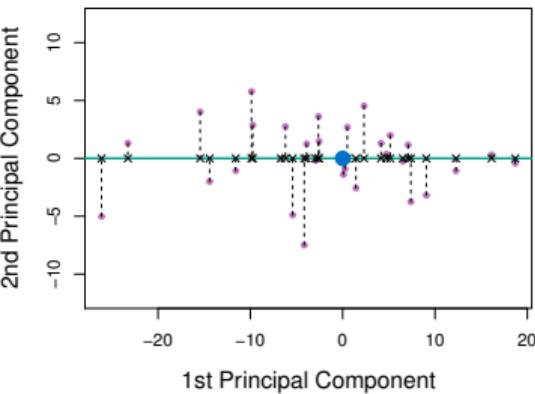
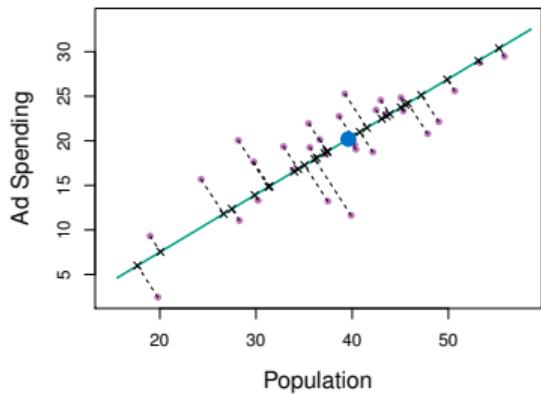
- We refer to the elements $\phi_{11}, \dots, \phi_{p1}$ as the **loadings** of the first principal component; together, the loadings make up the principal component loading vector,
 $\phi_1 = (\phi_{11}, \dots, \phi_{p1})^T$.
- We constrain the loadings so that their sum of squares is equal to one, since otherwise setting these elements to be arbitrarily large in absolute value could result in an arbitrarily large variance.

Example



The population size (pop) and ad spending (ad) for 100 different cities are shown as purple circles. The green solid line indicates the first principal component direction, and the blue dashed line indicates the second principal component direction.

Example: continued



A subset of the advertising data. The mean pop and ad budgets are indicated with a blue circle.

Left: The first principal component direction is shown in green. It is the dimension along which the data vary the most, and it also defines the line that is closest to all n of the observations. The distances from each observation to the principal component are represented using the black dashed line segments.

Right: The left-hand panel has been rotated so that the first principal component direction coincides with the x-axis.

Computation of Principal Components

- Suppose we have a $n \times p$ data set \mathbf{X} . Since we are only interested in variance, we assume that each of the variables in \mathbf{X} has been centered to have mean zero (that is, the column means of \mathbf{X} are zero).
- We then look for the linear combination of the sample feature values of the form

$$z_{i1} = \phi_{11}x_{i1} + \phi_{21}x_{i2} + \cdots + \phi_{p1}x_{ip} \quad (1)$$

for $i = 1, \dots, n$ that has largest sample variance, subject to the constraint that $\sum_{j=1}^p \phi_{j1}^2 = 1$.

- Since each of the x_{ij} has mean zero, then so does z_{i1} (for any values of ϕ_{j1}). Hence the sample variance of z_{i1} can be written as $\frac{1}{n} \sum_{i=1}^n z_{i1}^2$.

Computation: continued

- Plugging in (1) the first principal component loading vector solves the optimization problem

$$\max_{\phi_{11}, \dots, \phi_{p1}} \frac{1}{n} \sum_{i=1}^n \left(\sum_{j=1}^p \phi_{j1} x_{ij} \right)^2 \text{ subject to } \sum_{j=1}^p \phi_{j1}^2 = 1$$

- This problem can be solved via a singular-value decomposition of the matrix \mathbf{X} , a standard technique in linear algebra.
- We refer to Z_1 as the first principal component, with realized values z_{11}, \dots, z_{n1} .

Geometry of PCA

- The **loading vector** ϕ_1 with elements $\phi_{11}, \phi_{21}, \dots, \phi_{p1}$ defines a direction in feature space along which the data vary the most.
- If we project the n data points x_1, \dots, x_n onto this direction, the projected values are the **principal component scores** z_{11}, \dots, z_{n1} themselves.

Further principal components

- The second principal component is the linear combination of X_1, \dots, X_p that has maximal variance among all linear combinations that are **uncorrelated** with Z_1 .
- The second principal component scores $z_{12}, z_{22}, \dots, z_{n2}$ take the form

$$z_{i2} = \phi_{12}x_{i1} + \phi_{22}x_{i2} + \cdots + \phi_{p2}x_{ip},$$

where ϕ_2 is the second principal component loading vector, with elements $\phi_{12}, \phi_{22}, \dots, \phi_{p2}$.

Further principal components: continued

- It turns out that constraining Z_2 to be uncorrelated with Z_1 is equivalent to constraining the direction ϕ_2 to be orthogonal (perpendicular) to the direction ϕ_1 . And so on.
- The principal component directions $\phi_1, \phi_2, \phi_3, \dots$ are the ordered sequence of eigenvectors of the matrix $\mathbf{X}^T \mathbf{X}$, and the variances of the components are the eigenvalues.

Illustration

- USAarrests data: For each of the fifty states in the United States, the data set contains the number of arrests per 100,000 residents for each of three crimes: Assault, Murder, and Rape. We also record UrbanPop (the percent of the population in each state living in urban areas).
- The principal component score vectors have length $n = 50$, and the principal component loading vectors have length $p = 4$.
- PCA was performed after standardizing each variable to have mean zero and standard deviation one.

USAarrests data: PCA plot

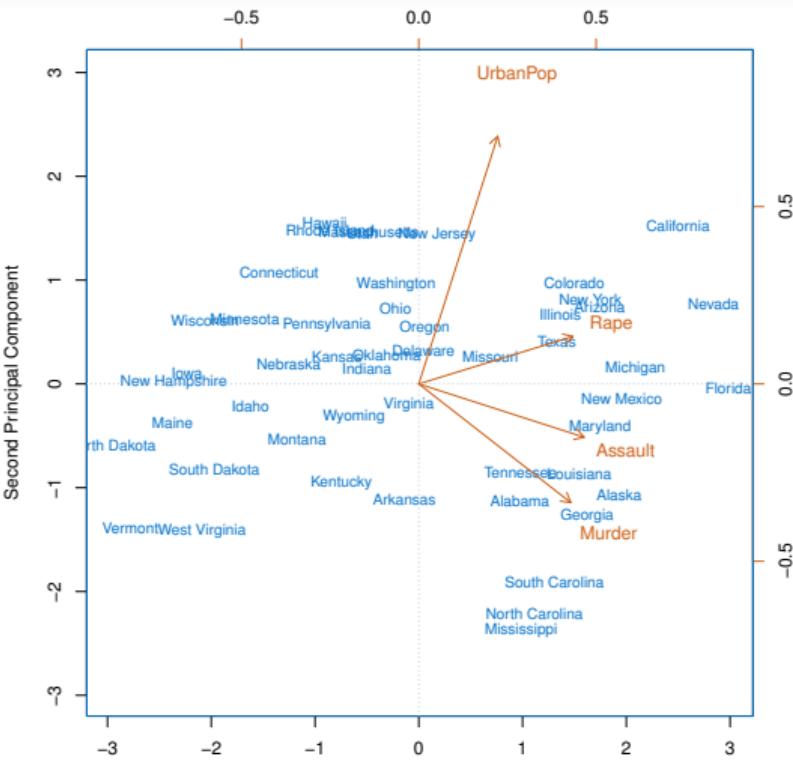


Figure details

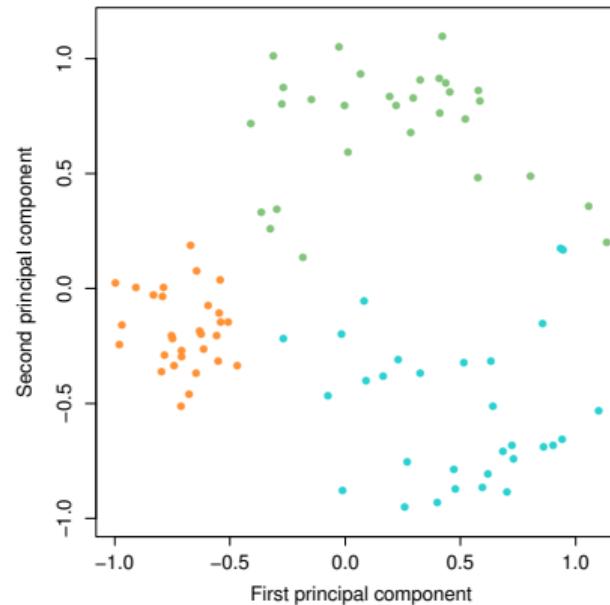
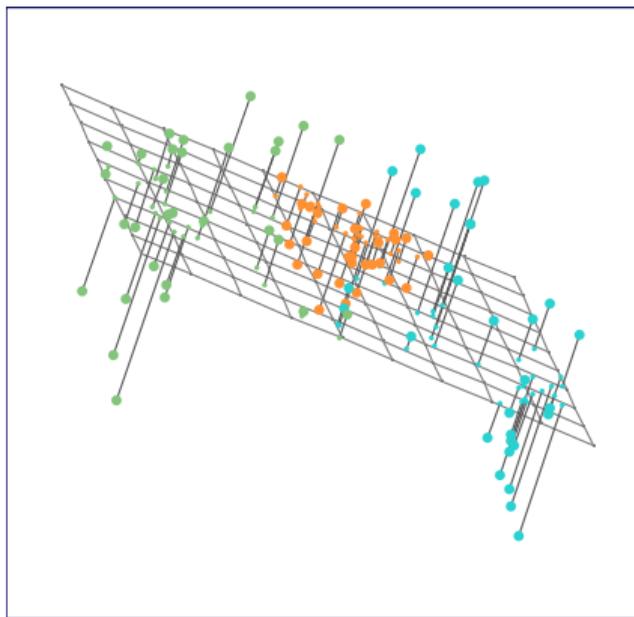
The first two principal components for the USArrests data.

- The blue state names represent the score (z_{i1}, z_{i2}) of each observation i for the first two principal components.
- The orange arrows indicate loading vectors (ϕ_{j1}, ϕ_{j2}) of each variable j for the first two principal component with axes on the top and right. For example, the loading for Rape on the first component is 0.54, and its loading on the second principal component 0.17 [the word Rape is centered at the point $(0.54, 0.17)$].
- This figure is known as a **biplot**, because it displays both the principal component scores and the principal component loadings.

PCA loadings

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

Another Interpretation of Principal Components

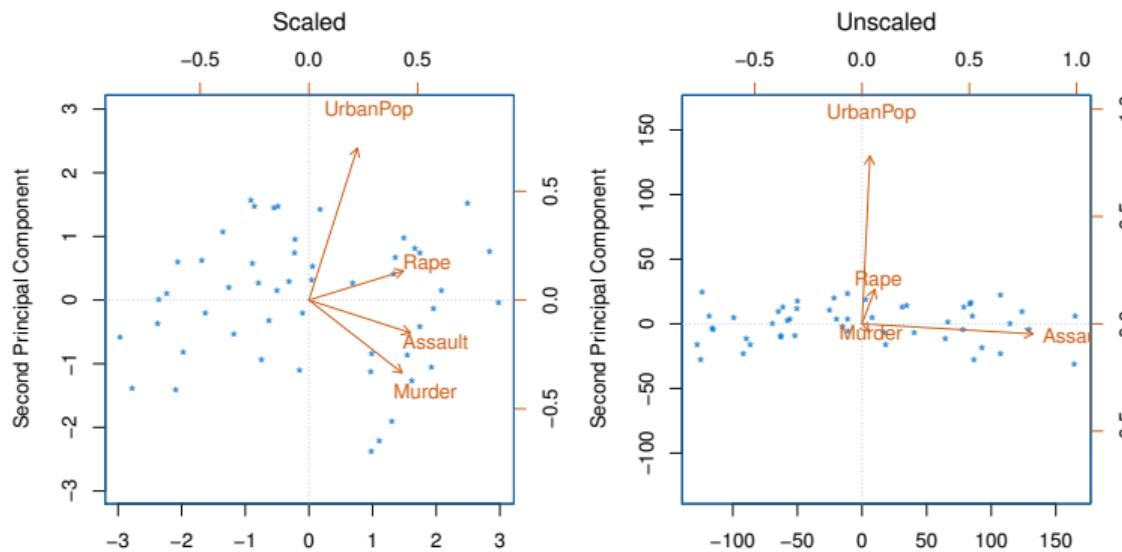


PCA find the hyperplane closest to the observations

- The first principal component loading vector has a very special property: it defines the line in p -dimensional space that is **closest** to the n observations (using average squared Euclidean distance as a measure of closeness)
- The notion of principal components as the dimensions that are closest to the n observations extends beyond just the first principal component.
- For instance, the first two principal components of a data set span the plane that is closest to the n observations, in terms of average squared Euclidean distance.

Scaling of the variables matters

- If the variables are in different units, scaling each to have standard deviation equal to one is recommended.
- If they are in the same units, you might or might not scale the variables.



Proportion Variance Explained

- To understand the strength of each component, we are interested in knowing the proportion of variance explained (PVE) by each one.
- The **total variance** present in a data set (assuming that the variables have been centered to have mean zero) is defined as

$$\sum_{j=1}^p \text{Var}(X_j) = \sum_{j=1}^p \frac{1}{n} \sum_{i=1}^n x_{ij}^2$$

and the variance explained by the m th principal component is

$$\text{Var}(Z_m) = \frac{1}{n} \sum_{i=1}^n z_{im}^2$$

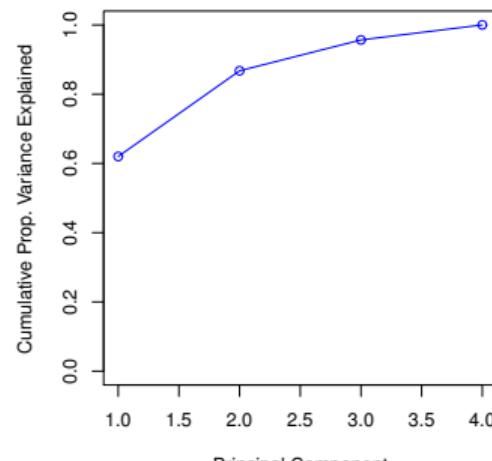
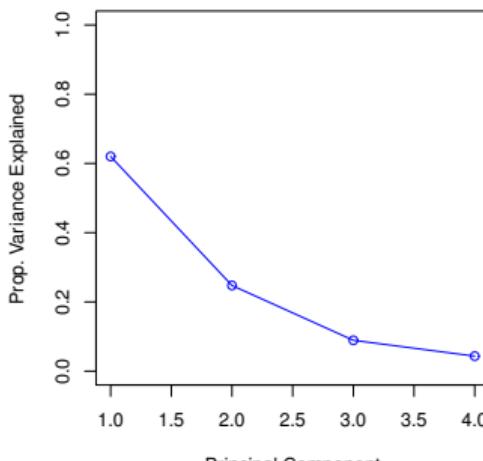
- It can be shown that $\sum_{j=1}^p \text{Var}(X_j) = \sum_{m=1}^M \text{Var}(Z_m)$, with $M = \min(n - 1, p)$.

Proportion Variance Explained: continued

- Therefore, the PVE of the m th principal component is given by the positive quantity between 0 and 1

$$\frac{\sum_{i=1}^n z_{im}^2}{\sum_{j=1}^p \sum_{i=1}^n z_{ij}^2}$$

- The PVEs sum to one. We sometimes display the cumulative PVEs.



How many principal components should we use?

If we use principal components as a summary of our data, how many components are sufficient?

- No simple answer to this question, as cross-validation is not available for this purpose.
 - Why not?
 - When could we use cross-validation to select the number of components?
- The scree plot on the previous slide can be used as a guide: we look for an elbow.

What's next?

How to use the principle components?

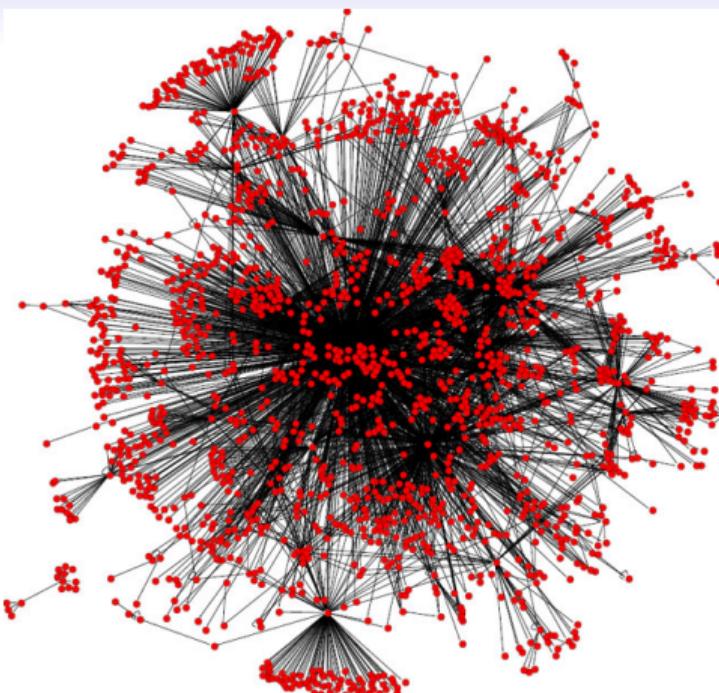
- Short answer: replacing the original data by the principal component scores. We can, for example,
 - regress response variable y on the scores (also known as principle component regression);
 - classify observations based on scores;
 - cluster the scores;
 - etc...

Other dimension reduction tools

We've only discussed one linear dimension reduction technique. There are many other dimension reduction tools:

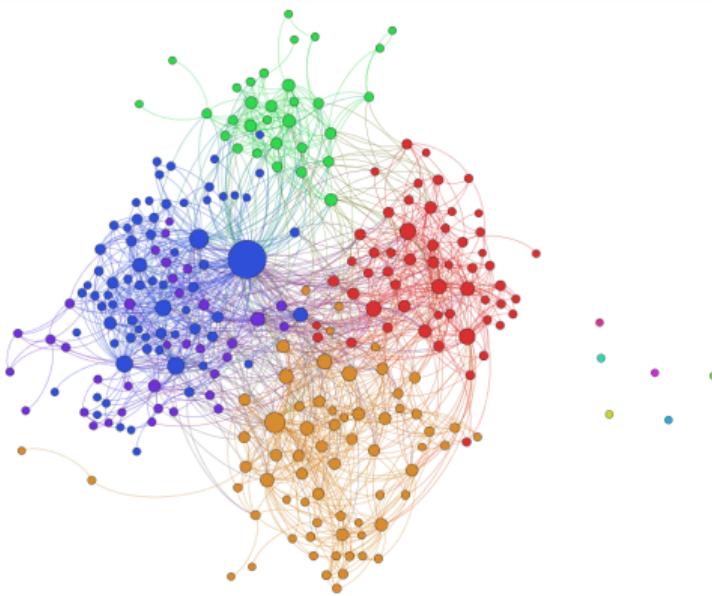
- CUR matrix approximation
- Non-negative matrix factorization (NMF)
- Kernel PCA
- T-distributed stochastic neighbor embedding (t-SNE)
- Isomap
- Autoencoder
- Self-Organizing Maps (SOM)
- Sammon mapping
- Locally Linear Embedding (LLE)

Motivation: gene regulatory networks



Nodes are genes, and edges represent regulatory interactions.

Motivation: friendship networks



Nodes are Facebook users, and edges represent friendship ties.

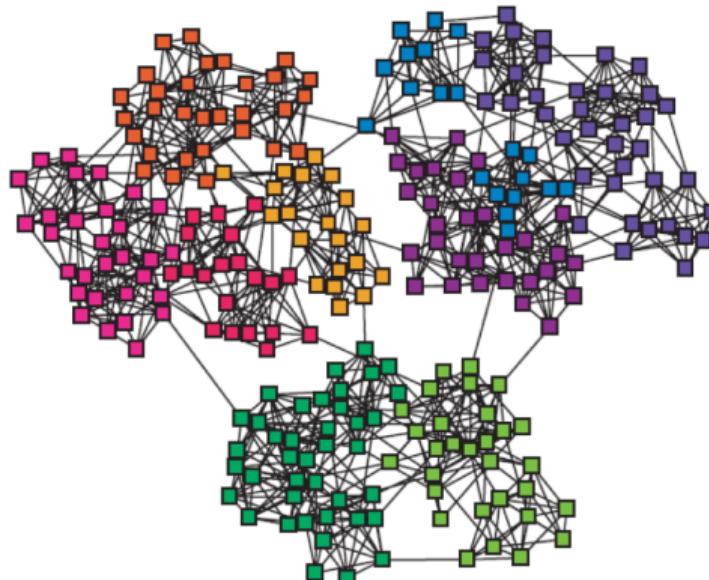
Motivation: co-authorship networks for statisticians



Nodes are statisticians and edges represent coauthorships. Names are shown for nodes with degree of 18 or larger.

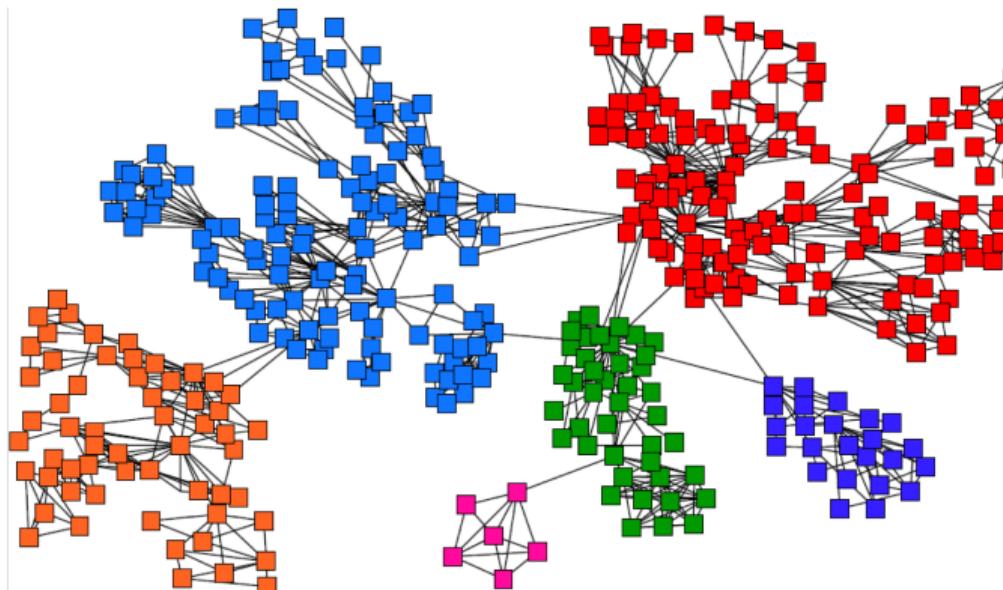
Communities

We often think of networks being organized into **cluster or communities**:



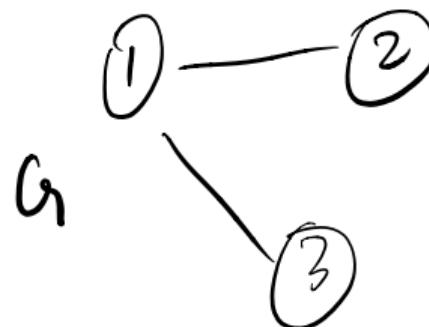
Communities

Goal: find clusters/communities such that nodes are densely connected within each cluster but sparsely connected across clusters.



Graphs

- $G = (V, E)$ is a **graph** or **network** which consists of
 - a set of vertices or nodes $V = \{1, \dots, n\}$
 - a set of edges or links $E \subseteq V \times V$
- We only consider undirected graphs i.e. edges have no directions.



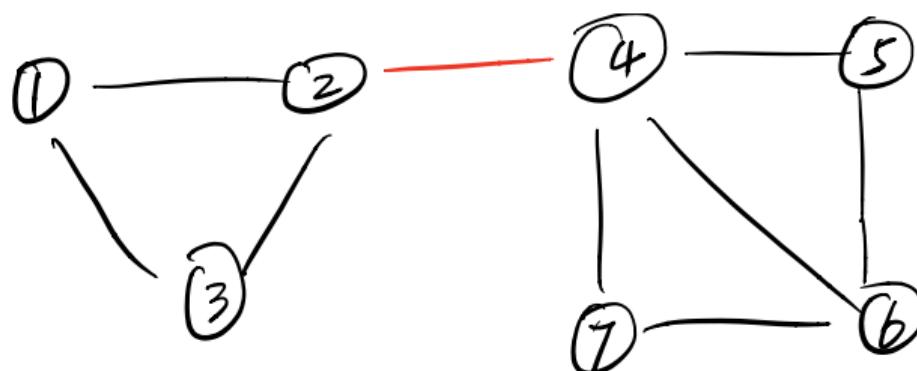
- $V = \{1, 2, 3\}$
- $E = \{\{1, 2\}, \{1, 3\}\}$

Community Detection Methods

- **Girvan-Newman algorithm** clusters nodes based on the notion of [edge betweenness](#). Similarly to hierarchical clustering, the result can be viewed as a dendrogram.
- **Spectral clustering** cuts a graph into several components. Using spectral theorem, we essentially embed nodes onto low dimensional real space and cluster the embedded features.
- **Stochastic block model** is a [generative model](#) which generates random graphs given model parameters. The learning process is the reverse: given an observed graph, estimate the model parameters.

Girvan-Newman algorithm

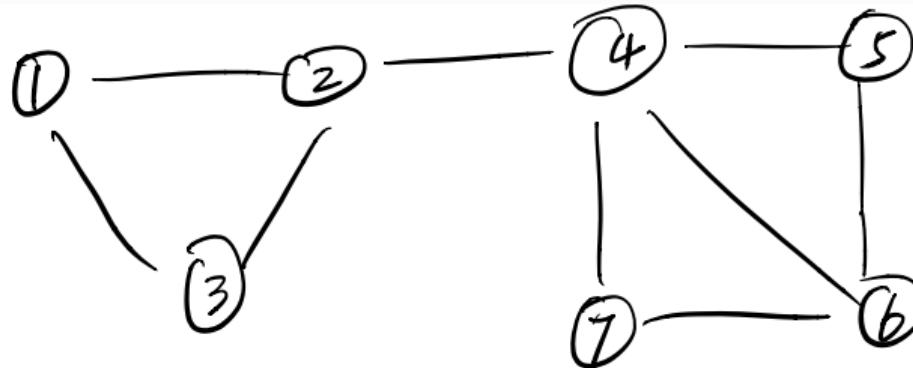
Intuition: if an edge is “heavily traveled”, it’s likely to be an edge between two communities. Girvan-Newman algorithm builds on this intuition.



Edge betweenness

- A **path of length K** is a sequence of $K + 1$ distinctive nodes i_0, \dots, i_K (except possibly $i_0 = i_K$) such that $\{i_{k-1}, i_k\} \in E$ for $k = 1, \dots, K$.
- The path between two nodes may not be unique. The **shortest path** is crucial here.
- **Edge betweenness:** the number of shortest paths passing through the edge

Example

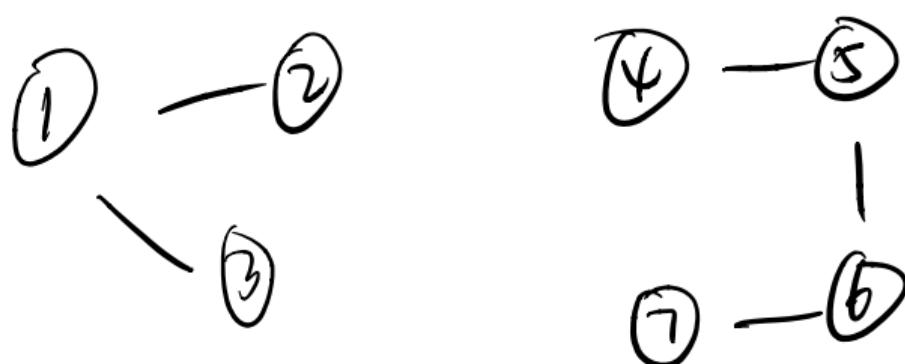


Edge $\{2, 4\}$ has the highest betweenness. In fact, this edge is on every shortest path between any of 1, 2, and 3 to any of 4, 5, 6, and 7. Its betweenness is therefore $3 \times 4 = 12$.

In contrast, the edge $\{4, 6\}$ is on only four shortest paths: those from any of 1, 2, 3, and 4 to 6.

Connect components

A **(connected) component** is a maximal set of nodes such that each pair of nodes is connected by a path.

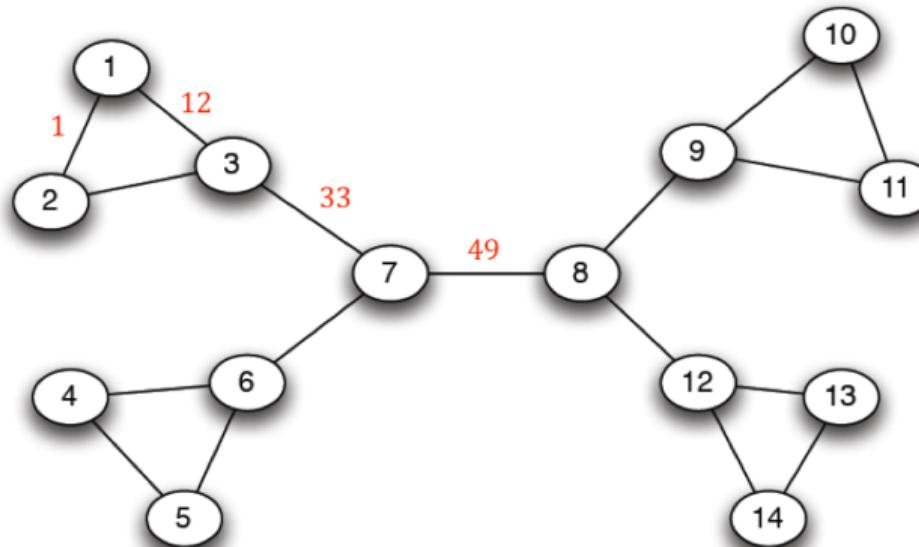


Two connected components: $\{1, 2, 3\}$ and $\{4, 5, 6, 7\}$

Girvan-Newman algorithm

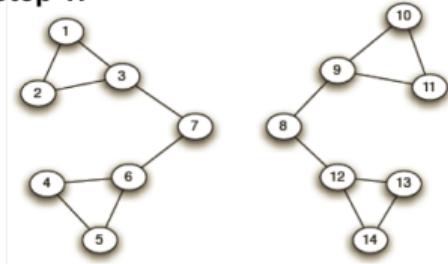
- We build a hierarchy in a “top-down” fashion. (Recall: we have discussed building a hierarchical clustering dendrogram in a “bottom-up” fashion.)
- Start with all points in one cluster. Repeat until no edges are left:
 - 1 Calculate betweenness of edges
 - 2 Remove edges with highest betweenness
- Connected components are communities
- Gives a hierarchical decomposition of the network

Example

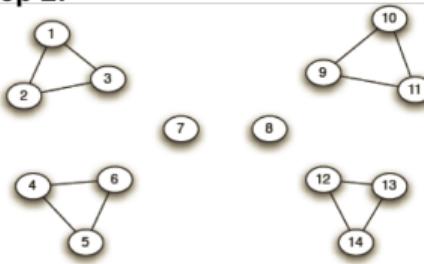


Example

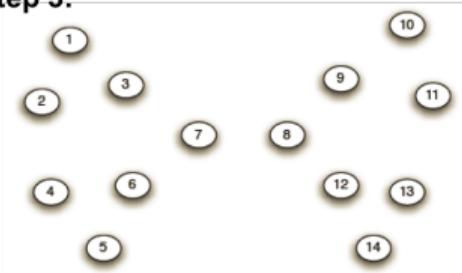
Step 1:



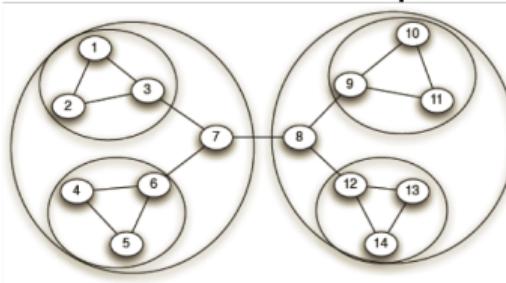
Step 2:



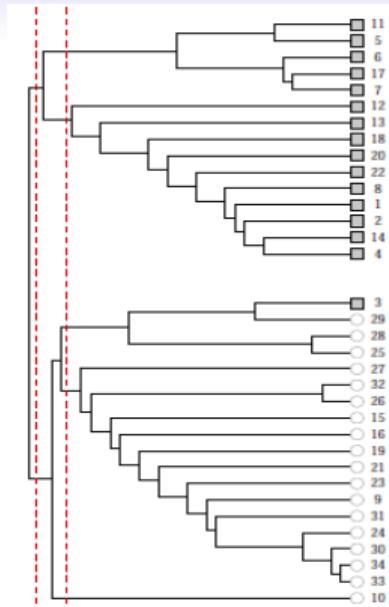
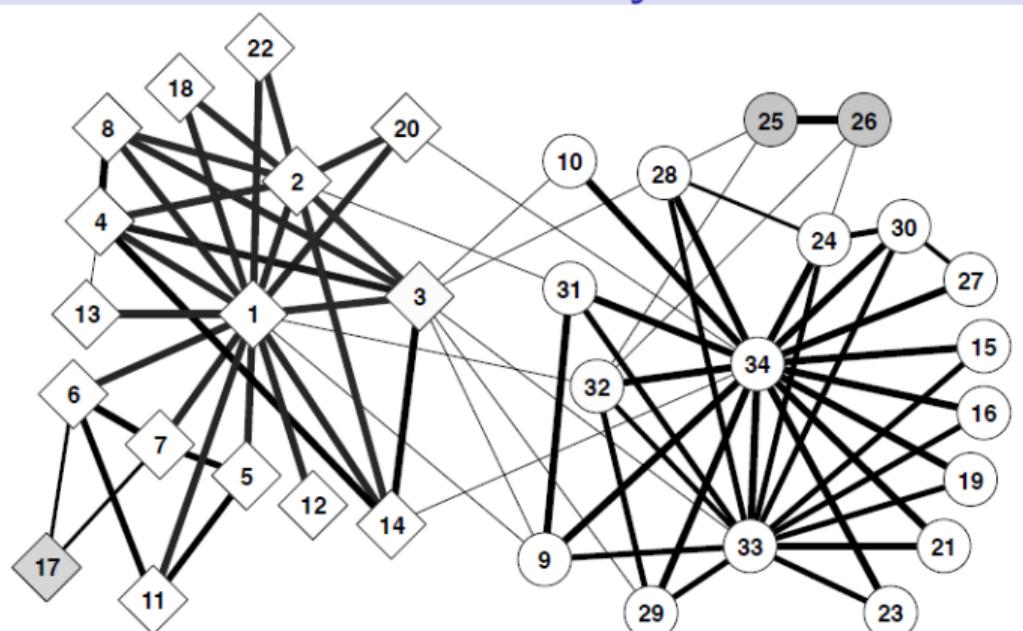
Step 3:



Hierarchical network decomposition:



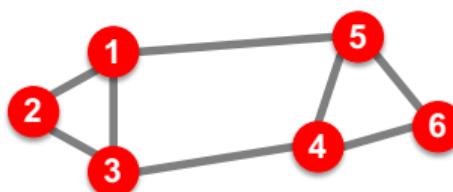
Zachary's Karate club



34 members of a karate club with links indicating interactions outside the club. A conflict arose between the administrator (node 34) and instructor (node 1), which led to the split of the club. Half of the members formed a new club around the instructor; members from the other part found a new instructor or gave up karate. Can algorithm

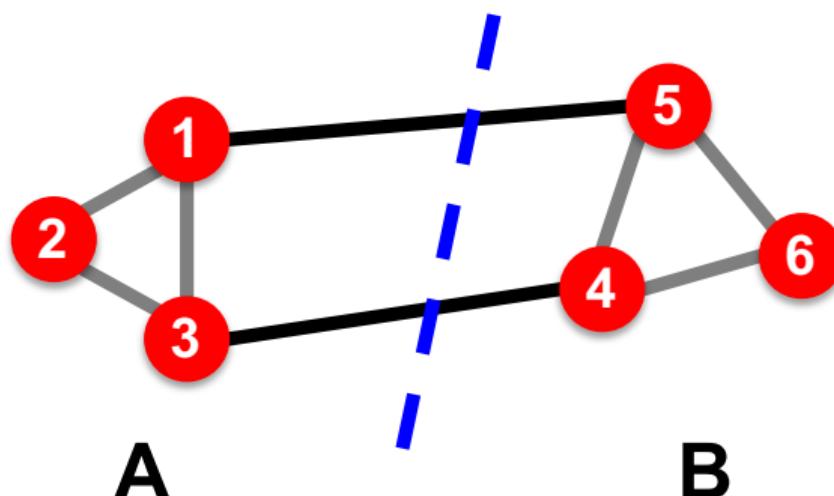
Spectral clustering

- Partition the network into two components by cutting edges.
- What makes a good partition?
- How can we efficiently identify such a partition?
- What if we want to partition the network into more than two components?



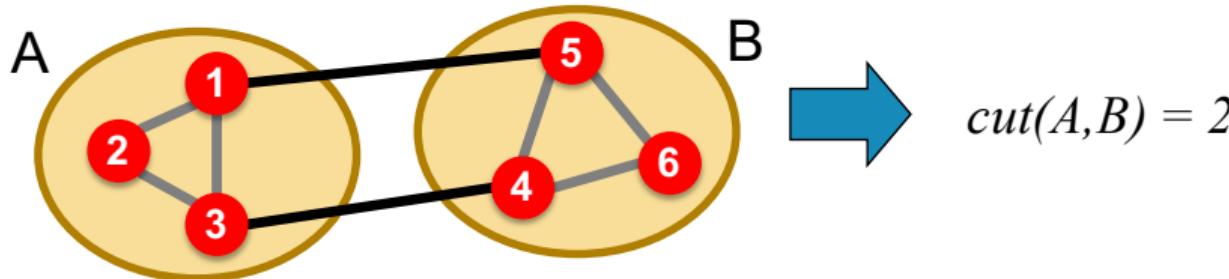
What makes a good partition?

- Maximize the number of within-group connections
- Minimize the number of between-group connections



Graph Cuts

- Express partitioning objectives as a function of the edge cut of the partition
- Cut:** the number of edges between two groups, denoted by $cut(A, B)$

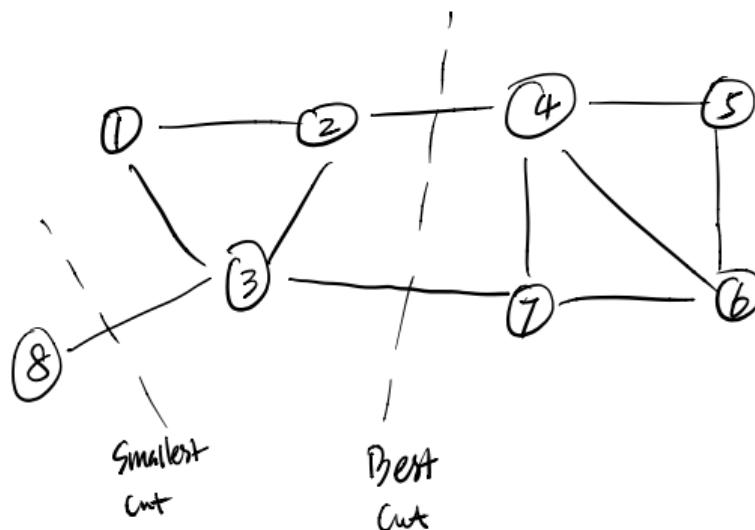


Minimize cut

- Find a partition $\{A, B\}$ of nodes that minimizes the graph cut

$$\arg \min_{A,B} \text{cut}(A, B)$$

- Degenerate case:

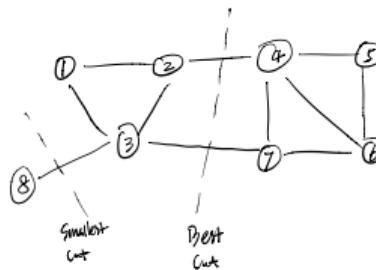


Normalized cut

- Connectivity between groups relative to the density of each group

$$ncut(A, B) = \frac{cut(A, B)}{vol(A)} + \frac{cut(A, B)}{vol(B)}$$

where $vol(A)$ is the total number of edges in A



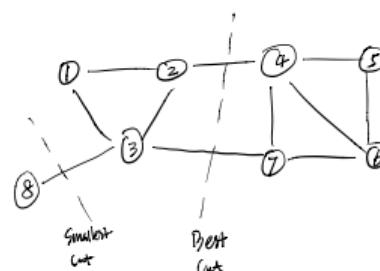
If we choose $S = \{8\}$ and $T = \{1, 2, 3, 4, 5, 6, 7\}$ (the smallest cut), then $cut(S, T) = 1$. $vol(S) = 1$, because there is only one edge connected to 8. On the other hand, $vol(T) = 11$, because all the edges have at least one end at a node of T . Thus, the normalized cut for this partition is $1/1 + 1/11 = 1.09$.

Normalized cut

- Connectivity between groups relative to the density of each group

$$ncut(A, B) = \frac{cut(A, B)}{vol(A)} + \frac{cut(A, B)}{vol(B)}$$

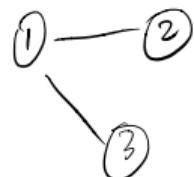
where $vol(A)$ is the total number of edges in A



Now if $S = \{1, 2, 3, 8\}$ and $T = \{4, 5, 6, 7\}$ (the best cut), then $cut(S, T) = 2$, $vol(S) = 6$, and $vol(T) = 7$. The normalized cut for this partition is thus only $2/6 + 2/7 = 0.62$.

How do we efficiently find a good partition?

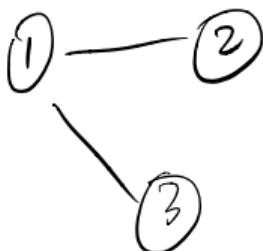
- In principle, we can compute $ncut(A, B)$ for all pairs of (A, B) and pick the pair with the smallest $ncut(A, B)$.
- But how many pairs in a network of n nodes? 2^{n-1}
- So can't solve the graph cut problem exactly.
- To find a good partition, we need a bit of matrix theory.
- How to represent a graph by a matrix?
- A graph can be represented by an **adjacency matrix** A with $A_{ij} = 1$ if there is an edge between i and j , and $A_{ij} = 0$ otherwise. A is binary and symmetric.



$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Graph Laplacian

- The **degree** d_i of a node i is the number of edges connected to it.
- The **degree matrix** D is a diagonal matrix with the diagonal entries being d_i .
- The **Laplacian matrix** is $L = D - A$.



$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, D = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$L = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

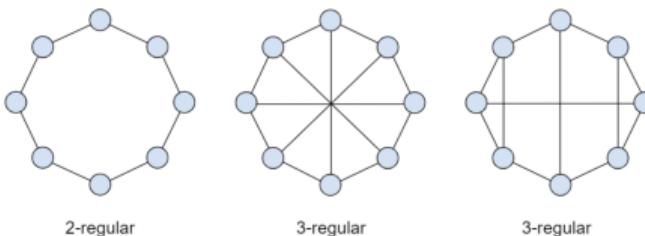
Spectral Graph Theory

- Let $\mathbf{x} = (x_1, \dots, x_n)^T$ be a vector in \mathbb{R}^n .
- Think of it as a label/value of each node.

Spectral Graph Theory

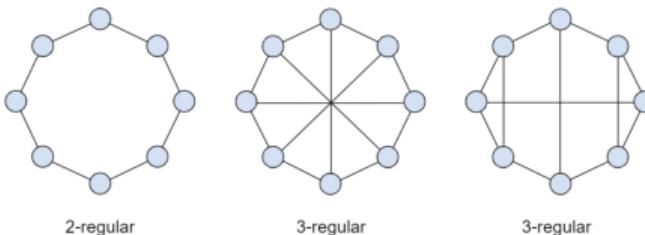
- Let $\mathbf{x} = (x_1, \dots, x_n)^T$ be a vector in \mathbb{R}^n .
- Think of it as a label/value of each node.
- We are going to use spectral theory which studies eigenpairs $(\mathbf{x}_i, \lambda_i)$ of the Laplacian matrix L , ordered by the magnitude (strength) of the eigenvalues $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$.

Example: d-regular graph



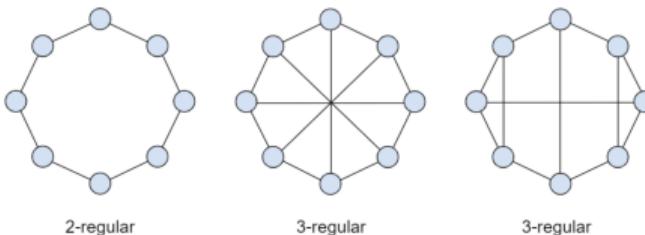
- Suppose all nodes have degree d and the graph G is connected. This is also known as d -regular graph.
- What are some eigenvalues/vectors of L ?
 $L\mathbf{x} = \lambda\mathbf{x}$ What's λ ? What's \mathbf{x} ?

Example: d-regular graph



- Suppose all nodes have degree d and the graph G is connected. This is also known as d -regular graph.
- What are some eigenvalues/vectors of L ?
 $L\mathbf{x} = \lambda\mathbf{x}$ What's λ ? What's \mathbf{x} ?
- Let's try $\mathbf{x} = \mathbf{1} = (1, 1, \dots, 1)^T$.
- Then $L\mathbf{x} = \mathbf{0} = 0 \cdot \mathbf{x}$. So $\lambda = 0$.

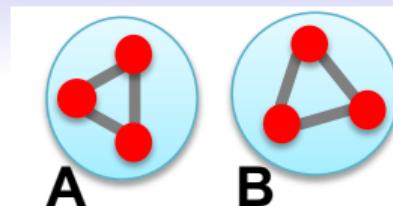
Example: d-regular graph



- Suppose all nodes have degree d and the graph G is connected. This is also known as d -regular graph.
- What are some eigenvalues/vectors of L ?
 $L\mathbf{x} = \lambda\mathbf{x}$ What's λ ? What's \mathbf{x} ?
- Let's try $\mathbf{x} = \mathbf{1} = (1, 1, \dots, 1)^T$.
- Then $L\mathbf{x} = \mathbf{0} = 0 \cdot \mathbf{x}$. So $\lambda = 0$.
- We found an eigenpair of L : $\mathbf{x} = \mathbf{1}$ and $\lambda = 0$.

Graph with 2 components

- What if G is not connected?
- G has 2 components, each d -regular
- What are some eigenvectors?

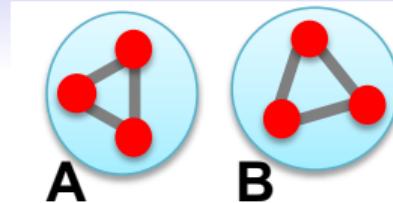


Graph with 2 components

- What if G is not connected?
- G has 2 components, each d -regular

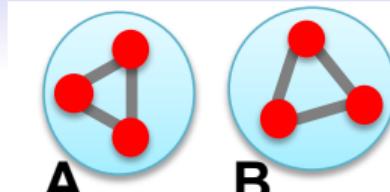
- What are some eigenvectors?

- $\mathbf{x} = (\underbrace{1, 1, \dots, 1}_A, \underbrace{0, \dots, 0}_B)^T \implies L\mathbf{x} = \mathbf{0} = 0 \cdot \mathbf{x}.$



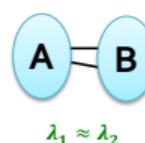
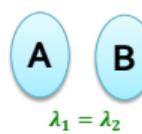
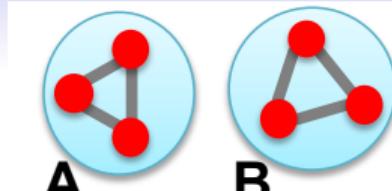
Graph with 2 components

- What if G is not connected?
- G has 2 components, each d -regular
 - What are some eigenvectors?
 - $\mathbf{x} = (\underbrace{1, 1, \dots, 1}_A, \underbrace{0, \dots, 0}_B)^T \implies L\mathbf{x} = \mathbf{0} = 0 \cdot \mathbf{x}.$
 - $\mathbf{x} = (0, \dots, 0, 1, 1, \dots, 1)^T \implies L\mathbf{x} = \mathbf{0} = 0 \cdot \mathbf{x}.$
 - And so in both cases the corresponding $\lambda = 0$ and \mathbf{x} indicates the right cluster membership.



Graph with 2 components

- What if G is not connected?
- G has 2 components, each d -regular
- What are some eigenvectors?
 - $\mathbf{x} = (\underbrace{1, 1, \dots, 1}_A, \underbrace{0, \dots, 0}_B)^T \implies L\mathbf{x} = \mathbf{0} = 0 \cdot \mathbf{x}.$
 - $\mathbf{x} = (0, \dots, 0, 1, 1, \dots, 1)^T \implies L\mathbf{x} = \mathbf{0} = 0 \cdot \mathbf{x}.$
- And so in both cases the corresponding $\lambda = 0$ and \mathbf{x} indicates the right cluster membership.
- A bit of intuition



2nd small e-val.
 λ_2 now has
 value very close
 to λ_1

Properties of Laplacian Matrix

- $L = D - A$ is symmetric.
- Eigenvalues are **non-negative** numbers.
- Eigenvectors are **orthogonal**, i.e. $\mathbf{x}_i^T \mathbf{x}_j = 0$ for any $i \neq j$.
- The trivial eigenpair ($\mathbf{x} = \mathbf{1}$, $\lambda = 0$)
- Obviously, $\lambda_1 = \lambda = 0$ is the smallest eigenvalue.

Second smallest eigenvalue λ_2

- Second smallest eigenvalue is critical for spectral clustering.
- It turns out

$$\lambda_2 = \min_{\mathbf{x}} \mathbf{x}^T L \mathbf{x} \text{ subject to } \|\mathbf{x}\|_2 = 1$$

- What is the meaning of $\min_{\mathbf{x}} \mathbf{x}^T L \mathbf{x}$ for a graph G ?

$$\begin{aligned}\mathbf{x}^T L \mathbf{x} &= \sum_{i=1}^n \sum_{j=1}^n L_{ij} x_i x_j = \sum_{i=1}^n \sum_{j=1}^n (D_{ij} - A_{ij}) x_i x_j \\ &= \sum_{i=1}^n D_{ii} x_i^2 - \sum_{\{i,j\} \in E} 2x_i x_j \\ &= \sum_{\{i,j\} \in E} (x_i^2 + x_j^2 - 2x_i x_j) = \sum_{\{i,j\} \in E} (x_i - x_j)^2\end{aligned}$$

λ_2 as optimization problem

- What else do we know about x ?

① x is a unit vector: $\sum_{i=1}^n x_i^2 = 1$

② x is orthogonal to the eigenvector $\mathbf{1} = (1, \dots, 1)^T$:

$$x^T \mathbf{1} = \sum_{i=1}^n x_i = 0$$

λ_2 as optimization problem

- What else do we know about x ?

① x is a unit vector: $\sum_{i=1}^n x_i^2 = 1$

② x is orthogonal to the eigenvector $\mathbf{1} = (1, \dots, 1)^T$:

$$x^T \mathbf{1} = \sum_{i=1}^n x_i = 0$$

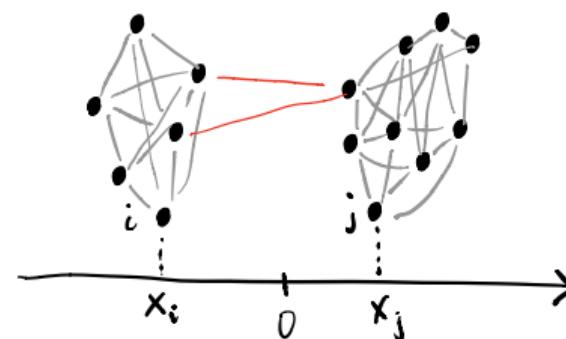
- The above implies some x_i are positive and some are negative.

λ_2 as optimization problem

- What else do we know about x ?
 - ① x is a unit vector: $\sum_{i=1}^n x_i^2 = 1$
 - ② x is orthogonal to the eigenvector $\mathbf{1} = (1, \dots, 1)^T$:

$$x^T \mathbf{1} = \sum_{i=1}^n x_i = 0$$

- The above implies some x_i are positive and some are negative.
- Recall: $\lambda_2 = \min_x \sum_{\{i,j\} \in E} (x_i - x_j)^2$.
- We want x_i and x_j to have the same sign if node i and node j are connected.



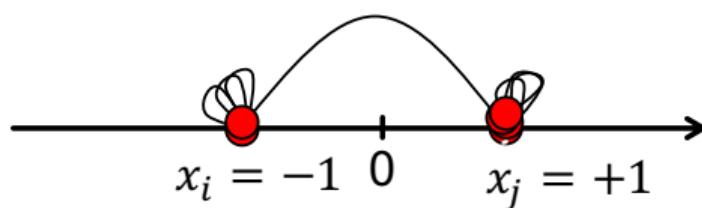
Find Optimal Cut

- Back to finding the optimal cut
- Express partition (A, B) as a vector

$$x_i = \begin{cases} +1 & \text{if } i \in A \\ -1 & \text{if } i \in B \end{cases}$$

- We can minimize the cut of the partition by finding a non-trivial vector \mathbf{x} that minimizes:

$$\arg \min_{\mathbf{x} \in \{-1, +1\}^n} \sum_{\{i, j\} \in E} (x_i - x_j)^2$$



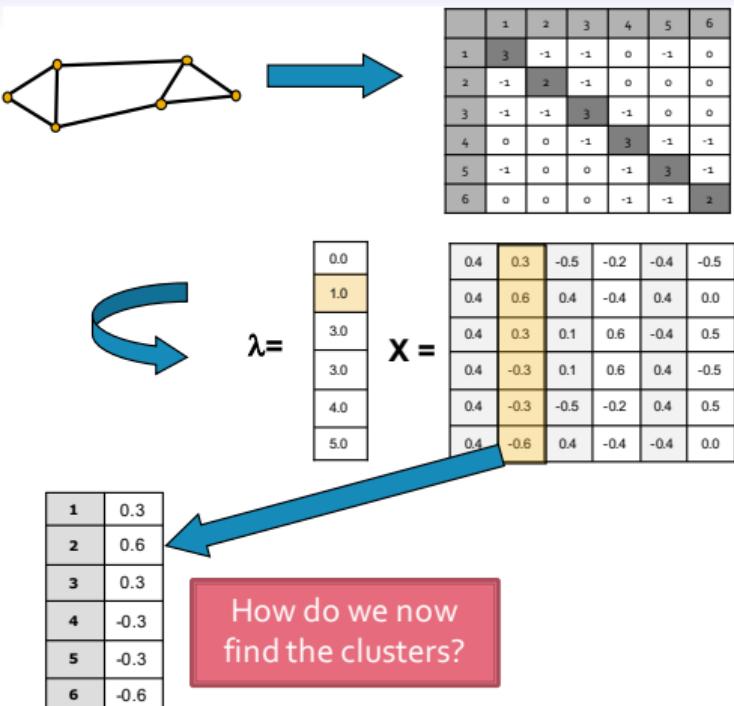
- Can't solve exactly. Let's relax \mathbf{x} and allow it to take any real values.

Continuous relaxation

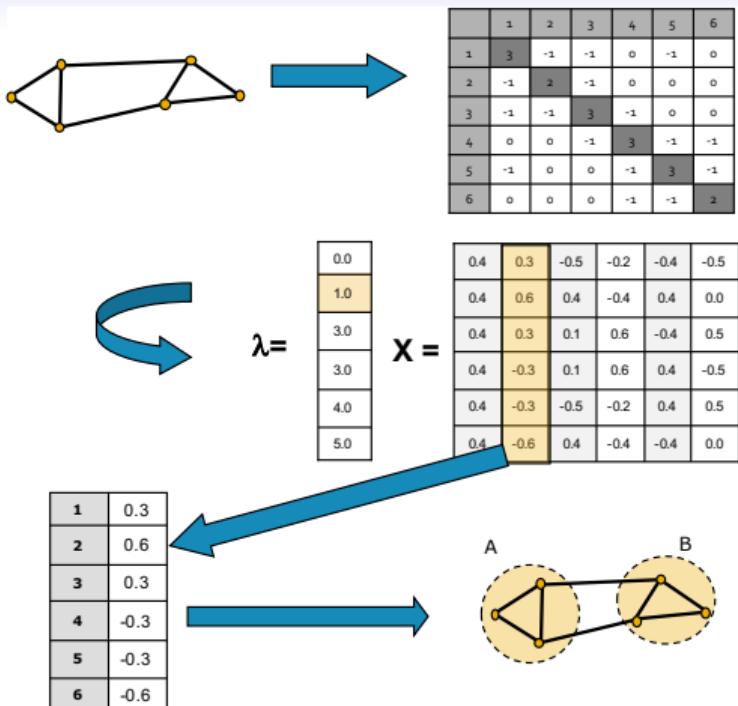
$$\min_{\mathbf{x} \in \mathbb{R}^n} \sum_{\{i,j\} \in E} (x_i - x_j)^2 = \mathbf{x}^T L \mathbf{x}$$

- As stated earlier, the minimum is the second smallest eigenvalue λ_2 of the Laplacian matrix L and its corresponding eigenvector \mathbf{x} is the solution.
- Assign nodes with positive x_i to one cluster and nodes with negative x_i to another.

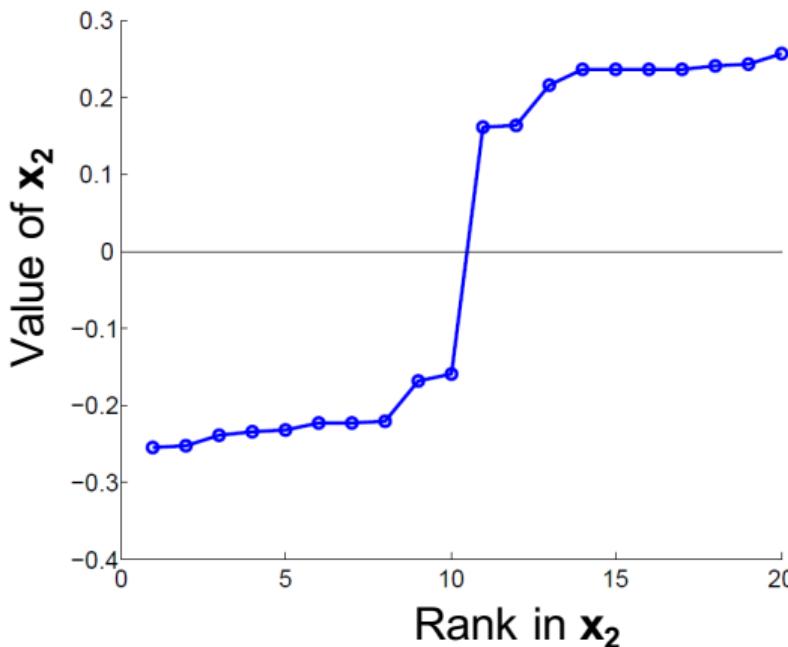
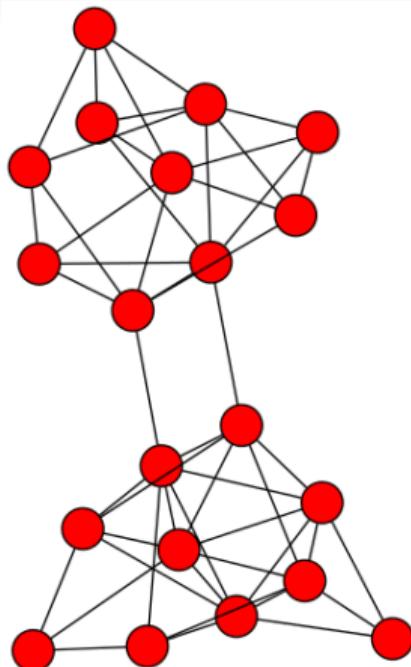
Illustration



Illustration



Example



So far...

- How to define a “good” partition of a graph?
 - Minimize a given graph cut criterion
- How to efficiently identify such a partition?
 - Approximate using information provided by the eigenvalues and eigenvectors of a graph

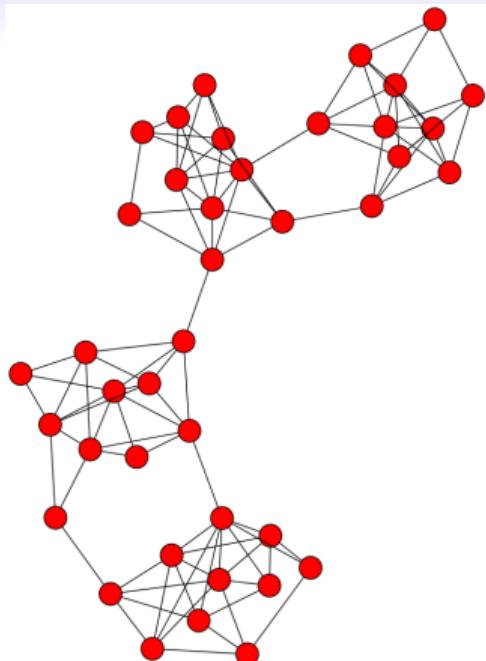
So far...

- How to define a “good” partition of a graph?
 - Minimize a given graph cut criterion
- How to efficiently identify such a partition?
 - Approximate using information provided by the eigenvalues and eigenvectors of a graph
- We are not quite done yet...
 - ① Graph can have weights. We replace adjacency matrix, degree matrix and Laplacian matrix by their weighted versions.
 - ② There are other types of Laplacian matrix which are more commonly used than the one we introduced. E.g. symmetric normalized Laplacian and random walk normalized Laplacian.
 - ③ It's applicable for $K > 2$ clusters as well.
 - ④ It's useful beyond clustering network data.

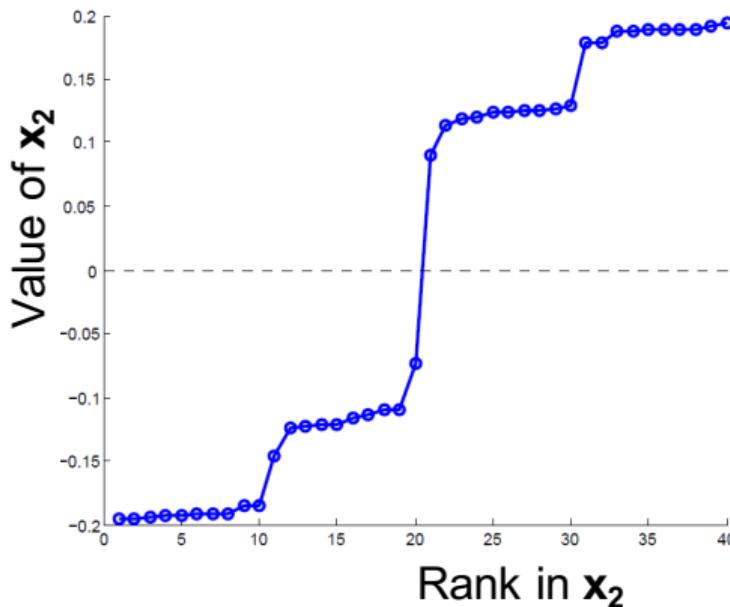
K-way Spectral Clustering

- How do we partition a graph into K clusters?
- Two basic approaches:
 - ① Recursive bi-partitioning: recursively apply bi-partitioning algorithm in a hierarchical manner. Disadvantages: Inefficient, unstable
 - ② Cluster multiple eigenvectors. Commonly used in recent papers. See next two slides for intuitions.

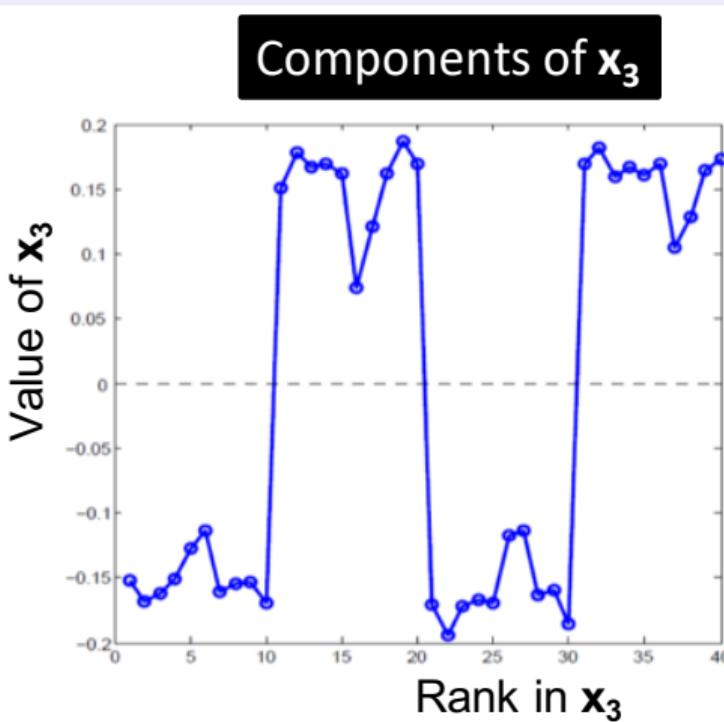
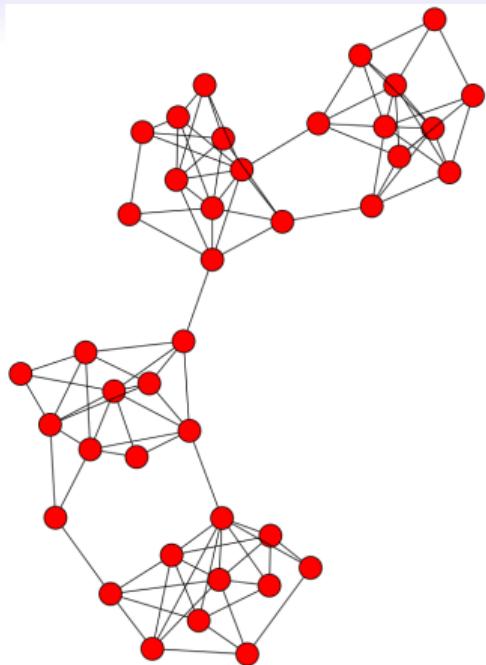
Some intuitions



Components of x_2



Some intuitions



Graph Spectral Clustering Algorithm

Input: a graph \mathcal{G} and the number K of clusters

- ① Compute the Laplacian matrix L from \mathcal{G} and the first K eigenvectors x_1, \dots, x_K of L . Let $X = [x_1, \dots, x_K]$ be an $n \times K$ matrix of eigenvectors. **Why K ?**
- ② Use any clustering algorithm such as K-means to cluster the rows of X .

Output: clusters C_1, \dots, C_K

General Spectral Clustering Algorithm

Input: data $Y \in \mathbb{R}^{n \times p}$ and the number K of clusters

- ① Construct a similarity graph \mathcal{G} from Y . (see next slide)
- ② Compute the Laplacian matrix L from \mathcal{G} and the first K eigenvectors x_1, \dots, x_K of L . Let $X = [x_1, \dots, x_K]$ be an $n \times K$ matrix of eigenvectors.
- ③ Use any clustering algorithm such as K-means to cluster the rows of X .

Output: clusters C_1, \dots, C_K

Ways to construct a similarity graph

Let s_{ij} denote a similarity measure between \mathbf{x}_i and \mathbf{x}_j , the i th and j th rows of X . E.g. Gaussian similarity

$$s_{ij} = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$

- k-nearest neighbor graphs: connect nodes i and j if i is among the k -nearest neighbors of j or i is among the k -nearest neighbors of i . Weight all edges by s_{ij} .
- ϵ -neighborhood graph: connect nodes i and j if $\|\mathbf{x}_i - \mathbf{x}_j\|^2 < \epsilon$. For small ϵ , no need to weight the edges. [Why?](#)
- Fully connected graph: connect all nodes and weight all edges by s_{ij} .

Advantage of spectral clustering

Advantages:

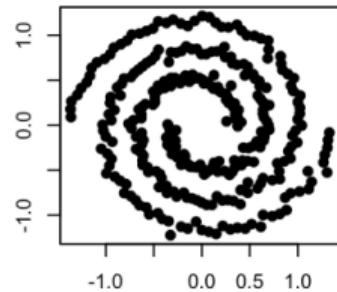
- Spectral clustering reduces the dimension from p to K .
- Capable of finding non-convex clusters.

Disadvantages:

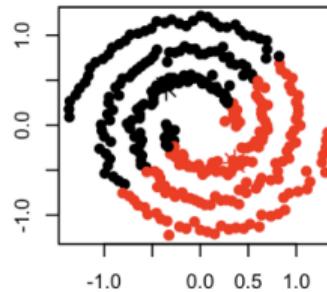
- Similarity function and its associated parameters matter.

Examples

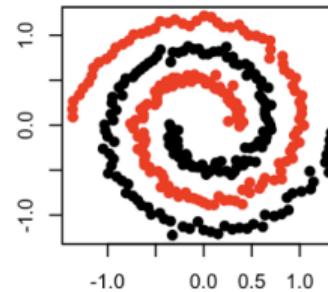
Non-covexity



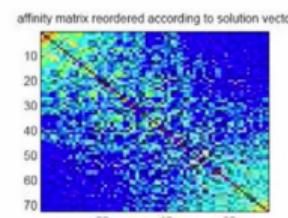
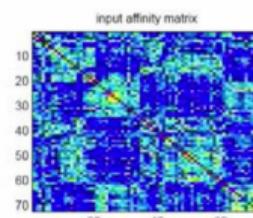
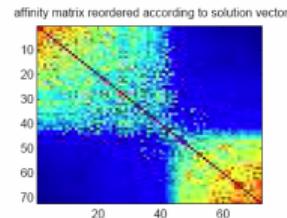
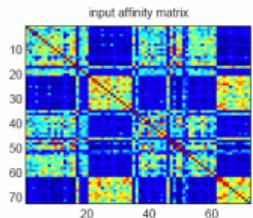
K-means



Spectral clustering



Good v.s. poor similarity measure



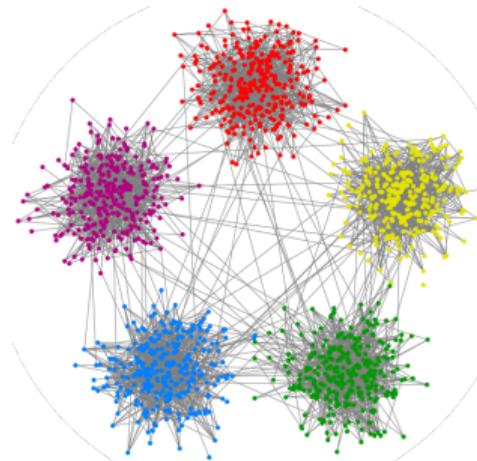
Stochastic Block Model

- Stochastic Block Model (SBM) is a **generative model** which generates random graphs with clusters/communities. An SBM depends on three sets of parameters:
 - ① Number K of clusters.
 - ② Cluster label $s_i \in \{1, \dots, K\}$. Node i belongs to cluster k if $s_i = k$.
 - ③ Connectivity probabilities matrix $Q \in [0, 1]^{K \times K}$. Q_{rs} is the probability of an edge between any node in cluster r and any node in cluster s .
- Given these parameters, SBM generates a random graph by

$$A_{ij} | Q, K, s_i, s_j \stackrel{\text{ind}}{\sim} \text{Bernoulli}(Q_{s_i s_j})$$

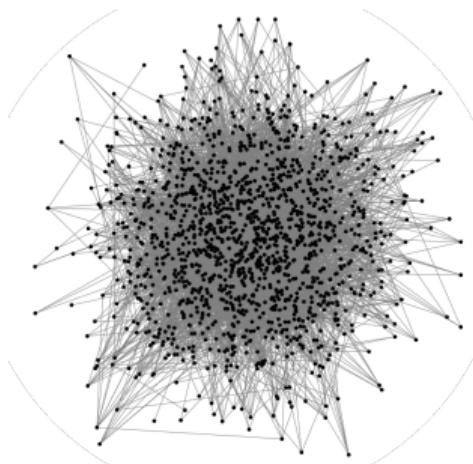
Simulated Example – generating a network with communities

- $n = 1000$ nodes
- $K = 5$ balanced clusters
- Within-cluster probability $Q_{rr} = 1/50$
- Across-cluster probability $Q_{rs} = 1/1000$ for $r \neq s$



Learning SBM

- We are given a messy network data without cluster labels and connectivity probabilities matrix.



- We need to learn $s = \{s_1, \dots, s_n\}$ and Q from the data.

Likelihood

- It's easy to write down the joint likelihood

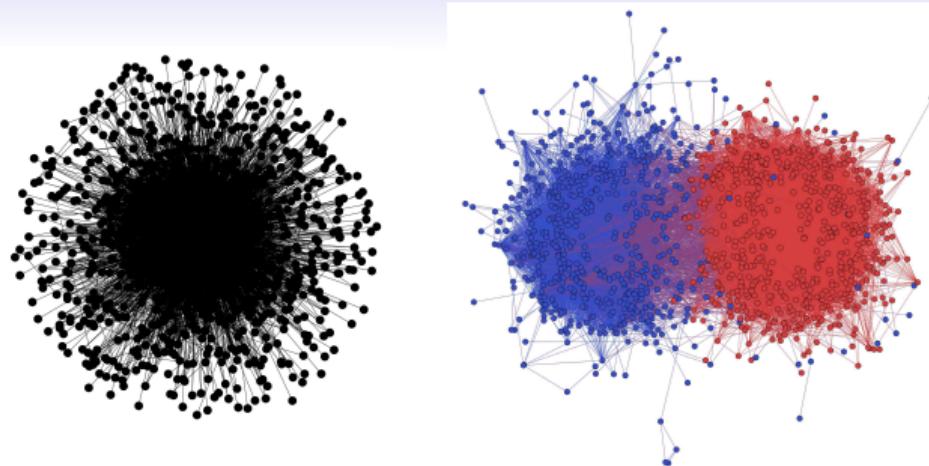
$$p(A|s, Q, K) = \prod_{i < j} Q_{s_i s_j}^{A_{ij}} (1 - Q_{s_i s_j})^{1 - A_{ij}}$$

- In principle, for a given K , we can then

$$\max_{s, Q} p(A|s, Q, K)$$

- However, it is very hard to solve it exactly.
- Instead, a lot of approximated solutions were proposed such as semidefinite programming, variational EM algorithm, Markov chain Monte Carlo, belief propagation, spectral methods, etc.

Example



The above graphs represent the real data set of the political blogs from Adamic and Glance (2005). Each vertex represents a blog and each edge represents the fact that one of the blogs refers to the other. The left graph is plotted with a random arrangement of the vertices, and the right graph is the output of SBM (using belief propagation), which gives 95% accuracy on the reconstruction of the political inclination of the blogs (blue and red colors correspond to left and right leaning blogs).

References

The content of these slides is based on the references listed below:

1. The Elements of Statistical Learning: Data mining, Inference, and Prediction (2009). Springer.
2. Applied Predictive Modeling (2013). Springer.
3. An Introduction to Statistical Learning with Applications in R (2021). Springer.
4. Pattern Recognition and Machine Learning (2006). Springer.
5. Data Mining and Analysis: Fundamental Concepts and Algorithms (2014). Cambridge University Press.
6. Stanford University CS231n: Deep Learning for Computer Vision
7. Texas A&M University STA 639: Data Mining and Analysis
8. Rice University STAT 640: Data Mining and Statistical Learning
9. Carnegie Mellon University 36-708: Statistical Methods for Machine Learning
10. University of Minnesota Twin Cities: Stat 8931 Statistical Learning and Data Mining
11. <http://www.mmmds.org/>
12. <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets>
13. <https://www.analyticsvidhya.com/blog/2023/01/gradient-descent-vs-backpropagation-whats-the-difference>
14. <https://www.geeksforgeeks.org/boosting-in-machine-learning-boosting-and-adaboost>
15. <https://speech.ee.ntu.edu.tw/~tlkagk/slides/Tutorial%20HYLee%20Share.pdf>
16. <https://kratzert.github.io/2016/02/12/understanding-the-gradient-flow-through-the-batch-normalization-layer.html>
17. <https://towardsdatascience.com/the-problem-of-vanishing-gradients-68cea05e2625>
18. <https://kharshit.github.io/blog/2018/12/28/why-batch-normalization>
19. Deep Residual Learning for Image Recognition (2015).