

# Lecture 1

block 2

Splines

Generalized additive models

# Basis function expansion

If  $y = w_0 + w_1x_1 + w_2x_1^2 + w_3e^{-x_2} + \epsilon$ ,

Model becomes linear if to recompute:

$$\begin{aligned}\phi_1(x_1) &= x_1 \\ \phi_2(x_1) &= x_1^2 \\ \phi_3(x_1) &= e^{-x_2}\end{aligned}$$

- Any model of the type  $Ey = \sum_i w_i \phi_i(x)$  can be fit by linear regression!

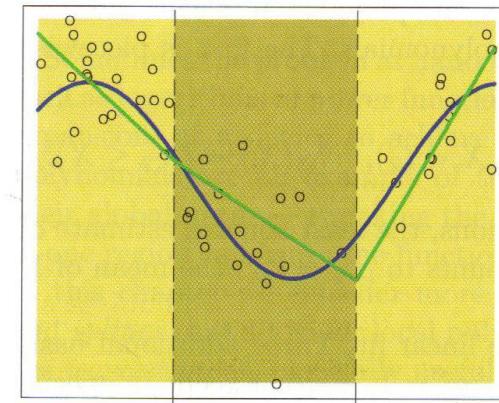
# Constructing a piecewise linear function

**Method A.** Introduce linear functions on each interval and a set of constraints

(4 free parameters)

$$\begin{cases} y_1 = \alpha_1 x + \beta_1 \\ y_2 = \alpha_2 x + \beta_2 \\ y_3 = \alpha_3 x + \beta_3 \\ \begin{cases} y_1(\xi_1) = y_2(\xi_1) \\ y_2(\xi_2) = y_3(\xi_2) \end{cases} \end{cases}$$

Continuous Piecewise Linear



**Method B.** Use a basis expansion (4 free parameters)

$$h_1(X) = 1, h_2(X) = X, h_3(X) = (X - \xi_1)_+, h_4(X) = (X - \xi_2)_+$$

**Theorem.** The two methods are equivalent.

# Splines

- A piecewise polynomial is called an **order-M** (or degree  $M-1$ ) **spline** if it is continuous and has continuous derivatives up to order  $M-2$  at the knots.
- **Equivalent:** An order- $M$  spline with  $K$  knots:

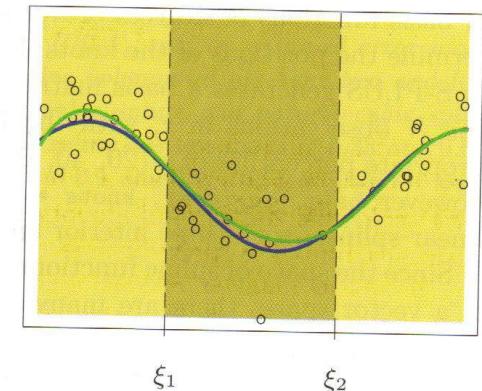
$$h_j(X) = X^{j-1}, j = 1, \dots, M$$

$$h_{M+l}(X) = (X - \xi_l)^{M-1}_+, l = 1, \dots, K$$

- An order-4 (degree-3) spline is called a **cubic spline**

In cubic splines, knot discontinuity is not visible

Continuous Second Derivative



# Natural cubic spline

- A cubic spline  $f$  is called **natural cubic spline** if its 2<sup>nd</sup> and 3<sup>rd</sup> derivatives are zero at  $a$  and  $b$

**Note** that  $f$  is linear on extreme intervals

Basis functions of natural cubic splines

$$N_1(X) = 1, N_2(X) = X, N_{k+2} = d_k(X) - d_{K-1}(X), \quad k = 1, \dots, K-2$$

where  $d_k(X) = \frac{(X - \xi_k)_+^3 - (X - \xi_K)_+^3}{\xi_K - \xi_k}$

# Fitting smooth functions to data

- Minimize

$$RSS(f, \lambda) = \sum_{i=1}^N (y_i - f(x_i))^2 + \lambda \int \{f''(t)\}^2 dt$$

where  $\lambda$  is **smoothing parameter**.

$\lambda=0$  : any function interpolating data

$\lambda=+\infty$  : least squares line fit

# Optimality of smoothing splines

- The function  $f$  minimizing  $RSS$  for a given  $\lambda$  is a natural cubic spline with knots at all unique values of  $x_i$  (NOTE:  $N$  knots!)
- Minimizing sum of squares:

$$f(x) = \sum_{j=1}^N N_j(x) \theta_j = N(x)^T \Theta$$

$$RSS(\Theta, \lambda) = (\mathbf{y} - \mathbf{N}\Theta)^T (\mathbf{y} - \mathbf{N}\Theta) + \lambda \Theta^T \Omega_N \Theta$$

$$\{\mathbf{N}\}_{ij} = N_j(x_i) \quad \{\Omega_N\}_{ij} = \int N_i''(t) N_j''(t) dt$$

$$\hat{\Theta} = (\mathbf{N}^T \mathbf{N} + \lambda \Omega_N)^{-1} \mathbf{N}^T \mathbf{y}$$

# A smoothing spline is a linear smoother

- Smoothing spline

$$\hat{f} = \mathbf{N}(\mathbf{N}^T \mathbf{N} + \lambda \Omega_N)^{-1} \mathbf{N}^T \mathbf{y} = \mathbf{S}_\lambda \mathbf{y}$$

is a **linear smoother**.

- Compare with other smoothers, such as linear regression.

# Degrees of freedom

- It can be shown that

$$\mathbf{S}_\lambda = (\mathbf{I} + \lambda \mathbf{K})^{-1}$$

where  $\mathbf{K}$  is **penalty matrix**

- Eigenvalue decomposition of  $\mathbf{K}$  :

$$\mathbf{S}_\lambda = \sum_{k=1}^N \rho_k(\lambda) \mathbf{u}_k \mathbf{u}_k^T$$

$$\rho_k(\lambda) = \frac{1}{1 + \lambda d_k}$$

- $d_k$  and  $\mathbf{u}_k$  are eigenvalues and eigenvectors

# Smoothing splines and shrinkage

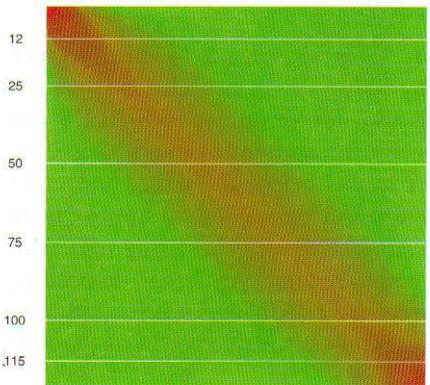
$$\mathbf{S}_\lambda \mathbf{y} = \sum_{k=1}^N \mathbf{u}_k \rho_k(\lambda) \langle \mathbf{u}_k^T, \mathbf{y} \rangle$$

- Smoothing spline decomposes vector  $\mathbf{y}$  with respect to basis of eigenvectors and shrinks respective contributions
- The eigenvectors ordered by  $\rho$  increase in complexity. The higher the complexity, the more the contribution is shrunk.

# Penalty and degrees of freedom

- $df_\lambda = \text{trace}(\mathbf{S}_\lambda) \rightarrow df_\lambda = \sum_{k=1}^N \frac{1}{1 + \lambda d_k}$
- $\lambda$  increase  $\rightarrow df_\lambda$  decrease
- higher  $\lambda \rightarrow$  higher penalization.
- Smoother matrix is has banded nature  $\rightarrow$  local fitting method

Smoother Matrix



# Automated selection of smoothing parameters

## What can be selected:

### Regression splines

- Degree of spline
- Placement of knots

### Smoothing spline

- Penalization parameter

# Automated selection of smoothing parameters

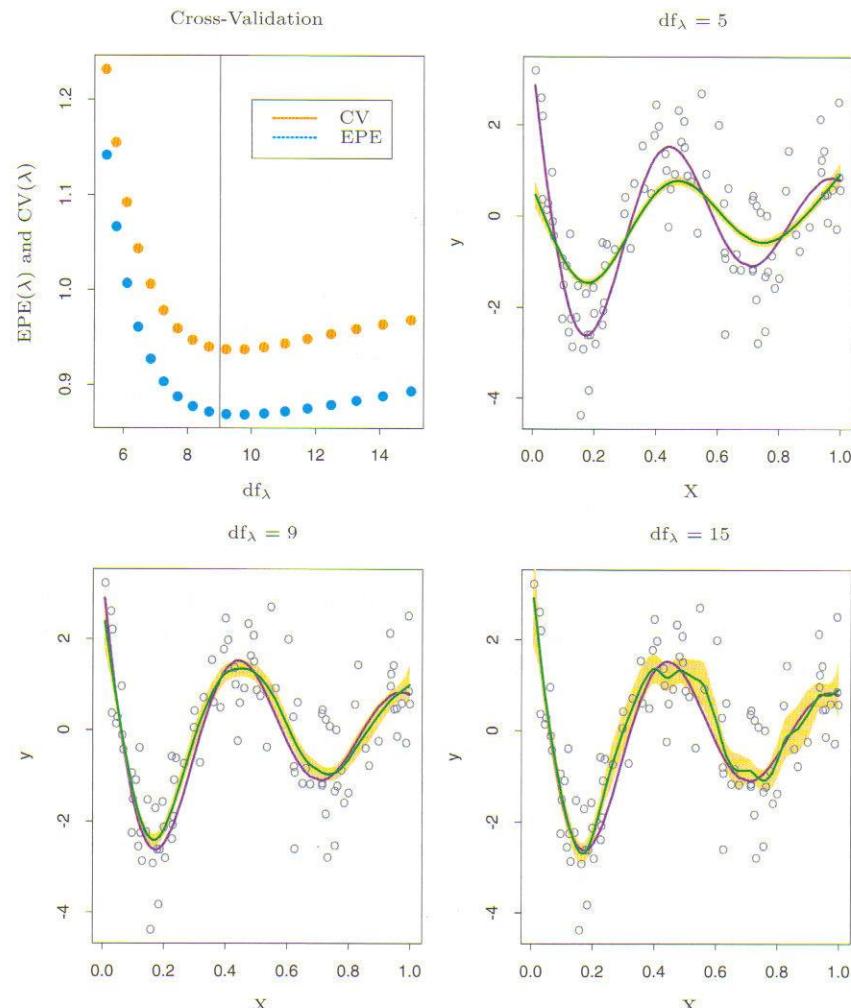
$$df_\lambda = \text{trace}(\mathbf{S}_\lambda) = \sum_{k=1}^N \frac{1}{1 + \lambda d_k}$$

- Use either  $df_\lambda$  or  $\lambda$ 
  - Given  $df_\lambda \rightarrow$  solve equation  $\rightarrow$  find  $\lambda$
- Use holdout principle or cross validation for parameter tuning

# Automated selection of smoothing parameters

- Bias-variance tradeoff

EPE – integrated squared prediction error,  
CV- cross validation



# Multidimensional splines

How to fit data smoothly in higher dimensions?

- Formulate a new problem

$$\min \sum_i (y_i - f(x_i))^2 + \lambda J[f]$$

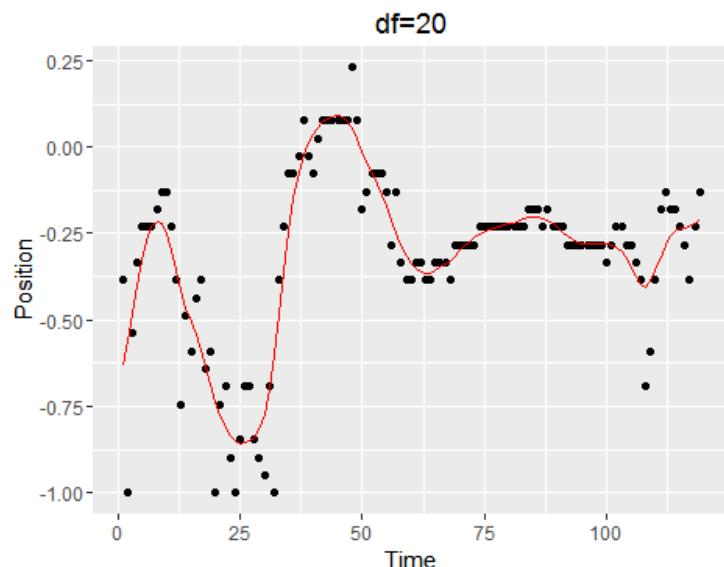
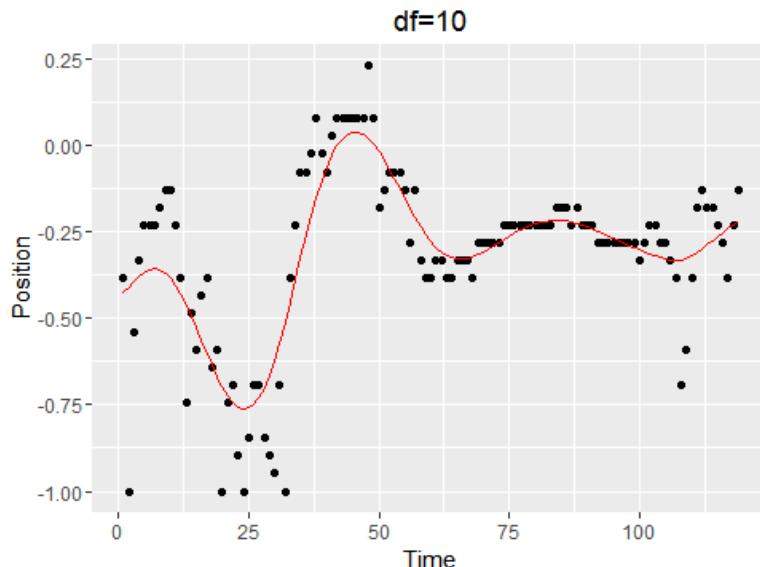
- The solution is **thin-plate splines**
- The solution in 2 dimensions is essentially sum of radial basis functions

$$f(x) = \beta_0 + \beta^T x + \sum \alpha_j \eta(\|x - x_j\|)$$

# Splines: R code

- Smoothing splines : `smooth.spline()`
- Natural cubic splines: `ns()` in **splines**
- Thin plate splines: `Tps()` in **fields**

```
res1=smooth.spline(data$Time,data$RSS_anchor2,df=10)
predict(res1,x=data$Time)$y
```



# Generalized additive models

- Model

$$Y \sim EF(\mu, \dots)$$

where

- $g(\mu) = \alpha + s_1(X_1) + s_2(X_2) + s_p(X_p)$
- $s_i(X)$  - smoothers, normally splines
- EF – distribution from exponential family
- $g$  – Link function
- Often linear terms are often included separately

$$EY = \alpha + s_1(X_1) + \dots + s_p(X_p) + \sum_{j=1}^q \beta_j X_{p+j}$$

**Example:** EF= normal, EF=Bernoulli (logistic)

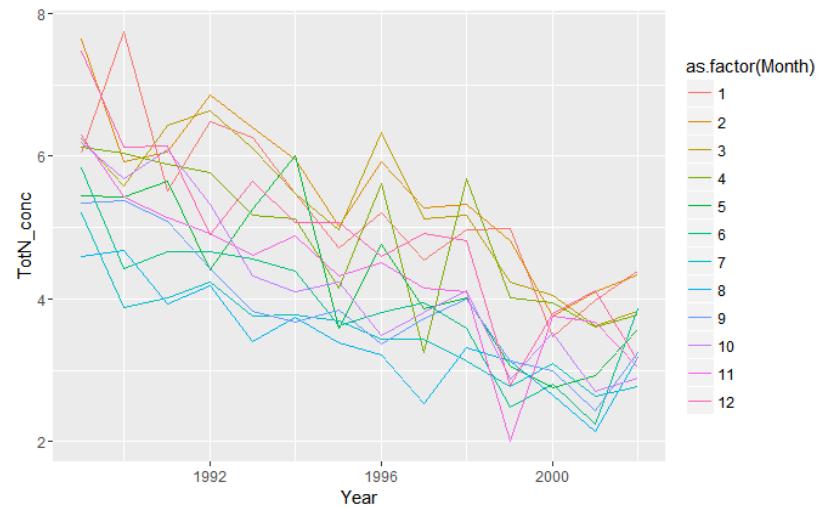
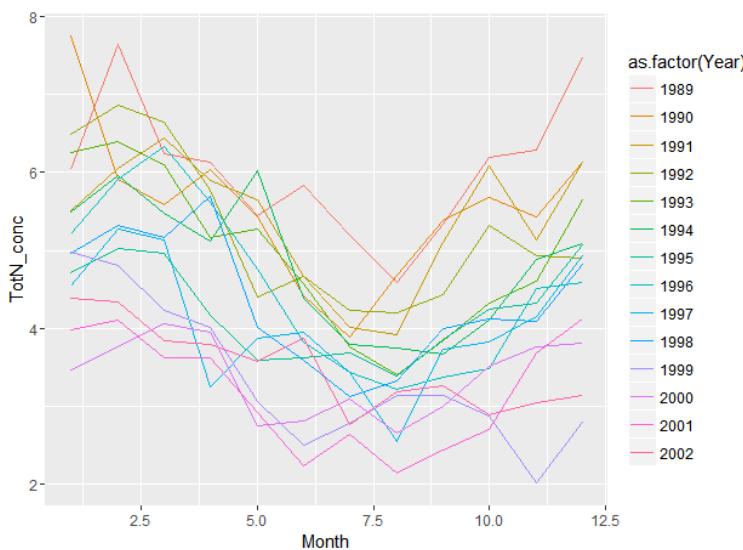
# Generalized additive models

- Sometimes even higher orders are included (thin-plate splines)

$$g(\mu) = \alpha + s_1(X_1) + \dots + s_p(X_p) + \sum_{j=1}^q \beta_j X_{p+j} + s_{12}(X_1, X_2)$$

- Method is reasonable to apply when additivity is observed or admissible

**Example:** Total Nitrogen level in Rhine river



# Estimation of additive models

## Estimation by MLE

$$g(\mu) = \alpha + f_1(x_1) + \dots + f_p(x_p)$$

## The backfitting algorithm for Normal model

1. Initialize :  $\hat{\alpha} = \frac{1}{N} \sum_{i=1}^N y_i$ ,  $\hat{f}_j \equiv 0$ ,  $j = 1, \dots, p$

2. Cycle :  $j = 1, \dots, p, 1, \dots, p, \dots, 1, \dots, p$

$$\hat{f}_j \leftarrow s_j \left[ \left\{ (y_i - \hat{\alpha} - \sum_{k \neq j} \hat{f}_k(x_{ik})) \right\} \right]$$

$$\hat{f}_j \leftarrow \hat{f}_j - \frac{1}{N} \sum_{i=1}^N \hat{f}_j(x_{ij})$$

$\lambda$  in each term  
can be  
estimated by  
CV

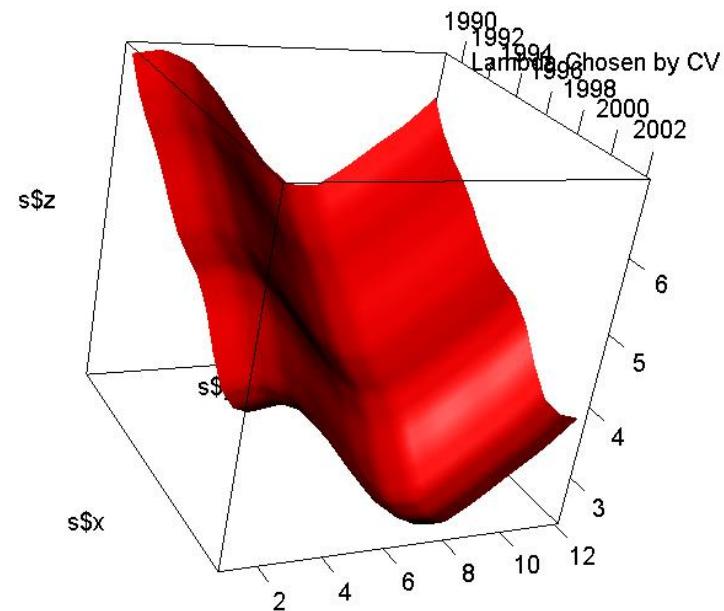
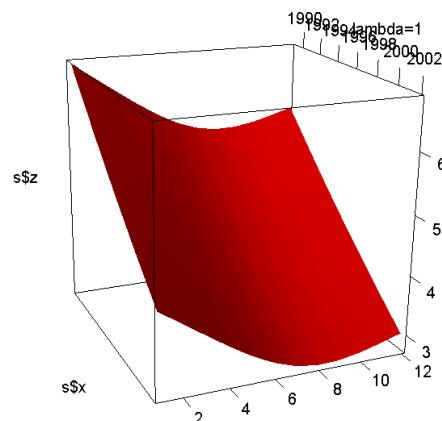
# Generalized additive models

- **Example:** Modelling the concentration of total nitrogen at Lobith on the Rhine
  - There are seasonal trends (GAM reasonable)
  - Variables
    - Nitrogen level
    - Year
    - Month
- R: package **mgcv** (also package **gam**)
  - `gam(formula, family,data,select, method)`
    - Select allows for term (variable) selection
  - `predict()`, `plot()`, `summary()`...
  - `s(k, sp)`
    - k should be the same as the amount of **unique values** of this variable in **smoothing splines**
    - sp - smoothing penalty.

# Generalized additive models

- R code

```
river=read.csv2("Rhine.csv")
res=gam(TotN_conc~Year+Month+s(Year)
         +s(Month), data=river)
library(rgl)
library(akima)
s=interp(river$Year,river$Month,
          fitted(res))
persp3d(s$x, s$y, s$z, col="red")
```



# Generalized additive models

```
> summary(res)
```

- Seeing trend and seasonal pattern

Family: gaussian  
Link function: identity

Formula:

TotN\_conc ~ Year + Month + s(Year) + s(Month)

Parametric coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	0.0008852	0.0009512	0.931	0.3535
Year	0.0014169	0.0003421	4.142	5.63e-05 ***
Month	0.2517641	0.1048467	2.401	0.0175 *

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Approximate significance of smooth terms:

	edf	Ref.df	F	p-value
s(Year)	6.049	7.206	66.72	<2e-16 ***
s(Month)	4.476	5.611	35.45	<2e-16 ***

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

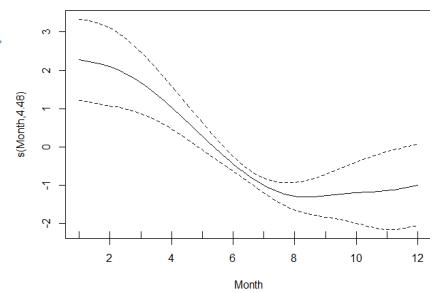
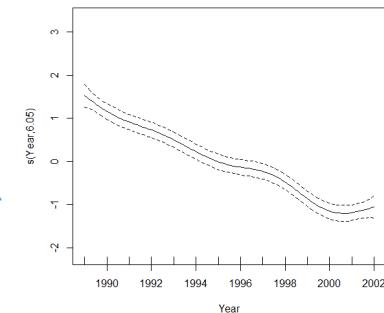
Rank: 19/21

R-sq.(adj) = 0.819 Deviance explained = 83.2%  
GCV = 0.27689 scale est. = 0.25638 n = 168

```
> res$sp
```

s(Year)	s(Month)
0.003342167	0.007087835

plot(res)



# Basic concepts

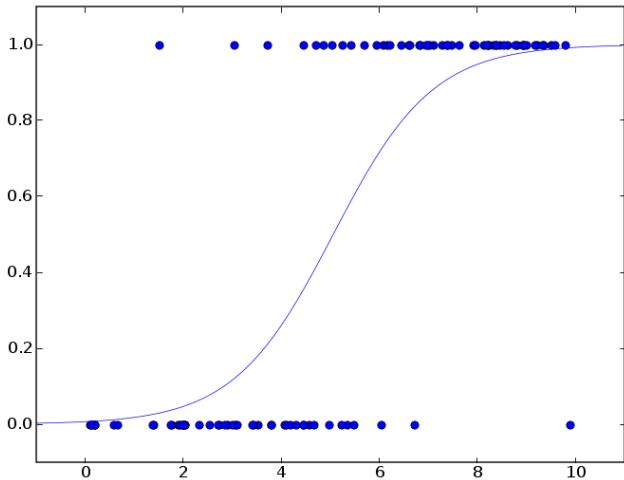
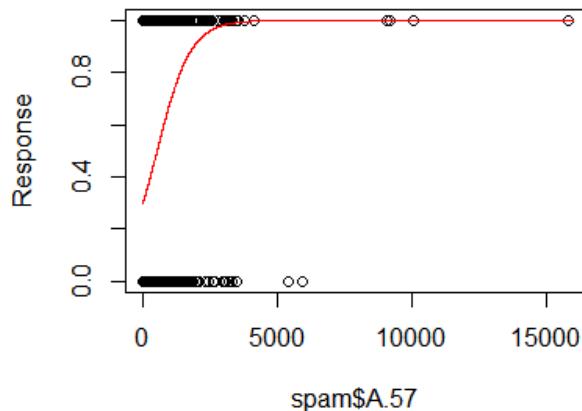
Lecture 1a

Course leader: Oleg Sysoev

# Logistic regression

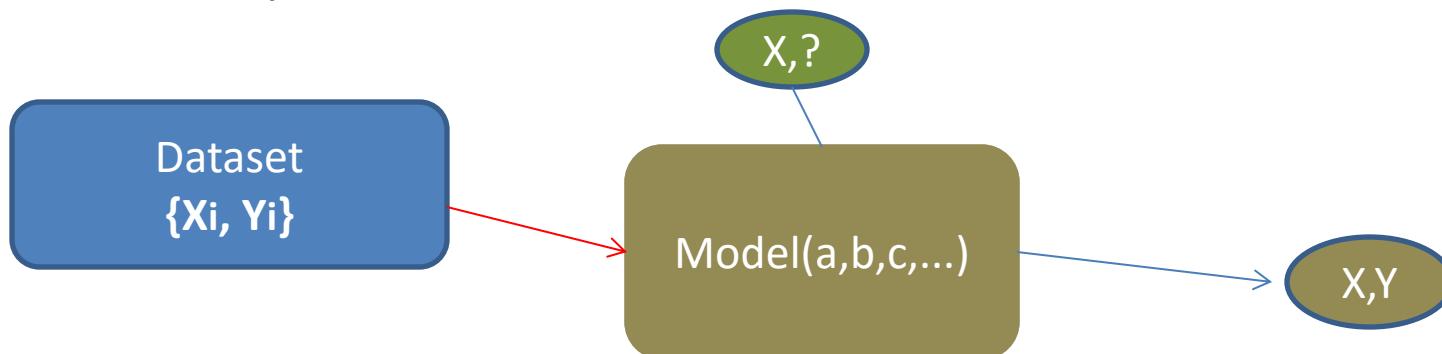
- Data  $Y_i \in \{Spam, Not\ Spam\}, X_i = \#of\ a\ word$
- Model:  $p(Y = Spam|w, x) = \frac{1}{1+e^{-w_0-w_1x}}$
- Fitting: maximum likelihood
- Prediction :  $p(spam) = p(Y = spam|x)$

We can also make point predictions  
-how?



# Types of learning

- **Supervised learning** (classification, regression)
  - Compute parameters from data
  - Given features of a new object, predict target
  - **Classification** ( $Y$ =categorical), **Regression** ( $Y$ =continuous)
- Most of ML models: Neural Nets, Decision Trees, Support Vector Machines, Bayesian nets



# Types of learning

- **Unsupervised learning** ( $\rightarrow$ Data Mining)
  - No target
  - Aim is to extract interesting information about
    - Relations of parameters to each other
    - Grouping of objects

**Ex:** clustering, density estimation, association analysis

X1<-> X2<-> X3...

# Types of learning

- **Semi-supervised**: targets are known only for some observations.
- **Active learning**. Strategies for deciding which observations to label
- **Reinforcement learning**. Find suitable actions to maximize the reward. True targets are discovered by trial and error.

# Basic ML ingredients

- **Data**  $D$ : observations
  - Features  $X_1, \dots, X_p$
  - Targets  $Y_1, \dots, Y_r$
- **Model**  $P(x | w_1, \dots, w_k)$  or  $P(y | x, w_1, \dots, w_k)$ 
  - Example: Linear regression  $p(y | x, w) = N(w_0 + w_1 x, \sigma^2)$
- **Learning procedure** (data → get parameters  $\hat{w}$  or  $p(w | D)$ )
  - Maximum likelihood, MAP, Bayes rule...
- **Prediction** of new data  $X^{new}$  by using the fitted model

Case	$X_1$	$X_2$	$Y$
1			
2			
...			

# K-nearest neighbor density estimation

- Data: Fish length  $X_1, \dots X_N$
- Model  $p(x|\Delta) = \frac{K}{N \cdot \Delta}$ 
  - $K$ : #neighbors in training data
  - $\Delta$ : length of the interval containing  $K$  neighbors
- Learning: Fix some  $K$  or find an appropriate  $K$
- Prediction: predict  $p(x|K)$

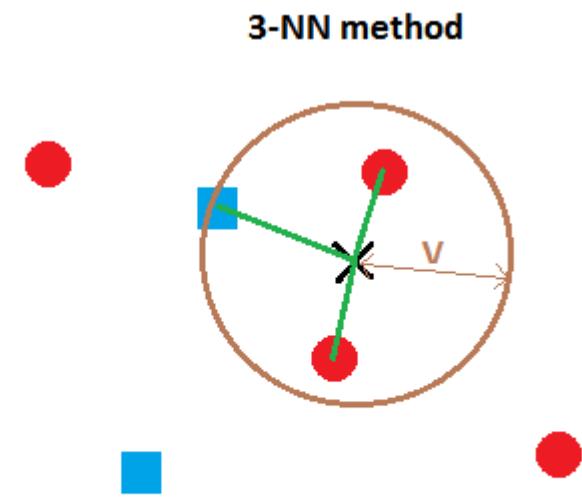
# K-nearest neighbor classification

- Given  $N$  observations  $(X_j, Y_j)$ 
  - $Y_j = C_i$ , where  $C_1, \dots, C_m$  are possible class values

- Model assumptions
  - Apply K-NN density estimation:

$$p(X = x | Y = C_i) = \frac{K_i}{N_i V}, p(C_i) = \frac{N_i}{N}$$

- $V$ : volume of the sphere
- $K_i$ : #obs from training data of  $Y = C_i$  in the sphere
- $N_i$ : #obs from training data of  $Y = C_i$



# Bayesian classification

- Prediction  $\hat{Y}(\mathbf{x}) = C_l$

$$l = \arg \max_{i \in \{1, \dots, m\}} p(C_i | \mathbf{x})$$

- Bayes theorem

$$p(C_i | \mathbf{x}) = \frac{p(\mathbf{x} | C_i) p(C_i)}{p(\mathbf{x})}$$

- We get

$$p(C_i | \mathbf{x}) \propto \frac{K_i}{K}$$

# K-nearest neighbor classification

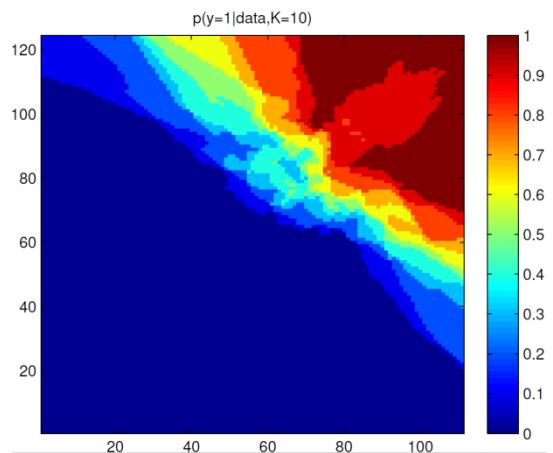
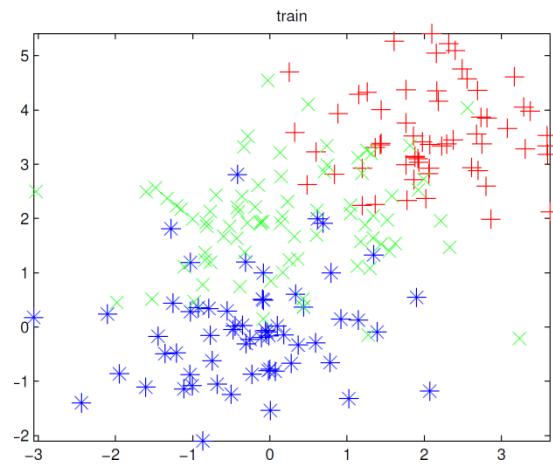
## Algorithm

1. Given training set  $D$ , number  $K$ , and test set  $T$
2. For each  $x \in T$ 
  1. For each  $i = 1, \dots, M$ 
    1.  $p'(C_i|x) = \frac{K_i}{K}$
  2. Compute  $l = \arg \max_{i \in \{1, \dots, m\}} p'(C_i|x)$
  3. Predict  $\hat{Y}(x) = C_l$

**Majority voting:** prediction for  $x$  is defined by majority voting of  $K$  neighbors

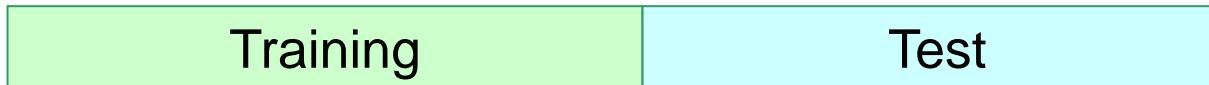
# Model types

- Parametric models
  - Have certain number of parameters independently of the size of training data
  - Assumption about of the data distribution
  - Ex: logistic regression
- Nonparametric models
  - Number of parameters (complexity) grows with training data
    - Example: K-NN classifier



# Model selection

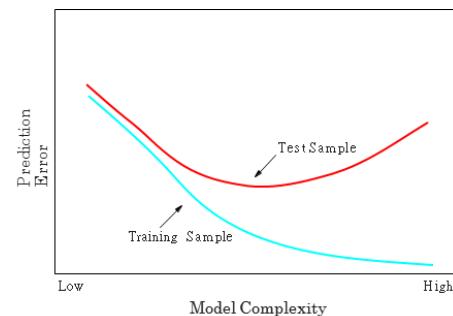
- Given several models  $M_1, \dots M_m$
- Divide data set into **training** and **test** data



- Fit models  $M_i$  to training data → get parameter values
- Use fitted models to predict test data and compare **test errors**  $R(M_1), \dots R(M_m)$
- Model with lowest prediction error is best

**Comment:**

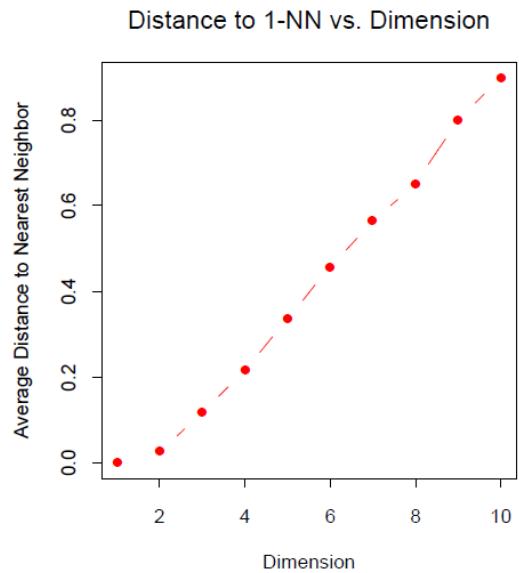
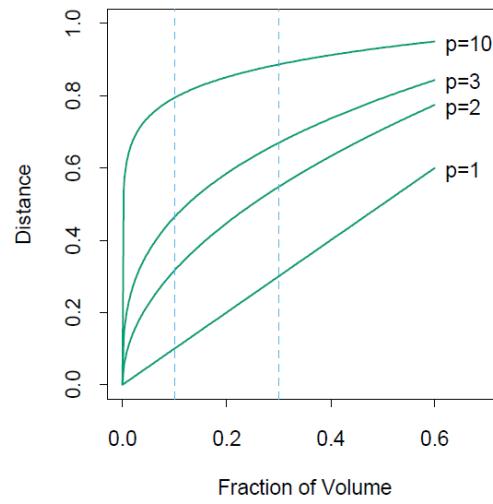
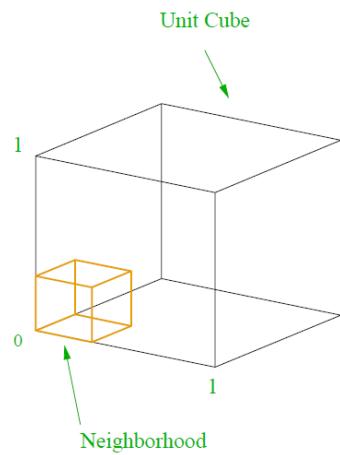
- Approach works well for moderate/large data. The figure is about overfitting



# Curse of dimensionality

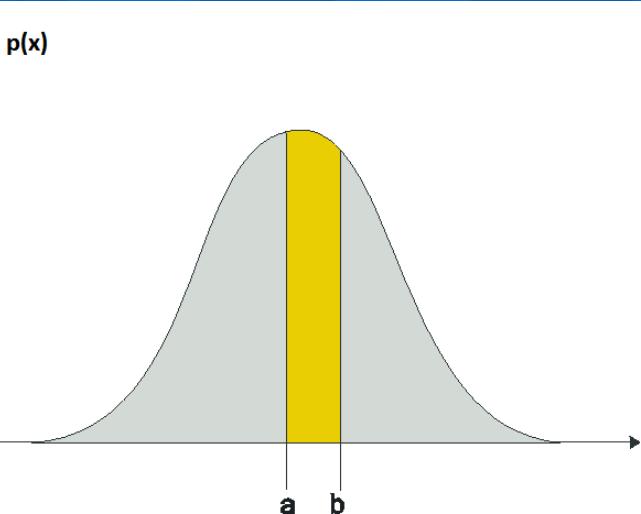
- Given data  $D$ :
  - Features  $X_1, \dots, X_p$
  - Targets  $Y_1, \dots, Y_r$
- When  $p$  increases models using "proximity" measures work badly
- **Curse of dimensionality**: A point has no "near neighbors" in high dimensions → using class labels of a neighbor can be misleading
  - Distance-based methods affected

# Curse of dimensionality



# Curse of dimensionality

- Hopeless? No!
- Real data normally has much lower effective dimension
  - Dimensionality reduction techniques
- Smoothness assumption
  - small change in one of Xs should lead to small change in Y → interpolation



# Basic concepts Introduction to Bayesian methods

## Lecture 1c

- $p(x \in [a, b]) = \int_a^b p(x)dx$
- $p(x) \geq 0, \int_{-\infty}^{+\infty} p(x)dx = 1$

# Probabilities

- **Laws of probabilities**

- Sum rule (compute **marginal** probability)

$$p(X) = \sum_Y p(X, Y)$$

$$p(X) = \int p(X, Y) dY$$

- Product rule

$$p(X, Y) = p(X|Y)p(Y)$$

Combination 1:

$$p(X) = \sum_Y p(X|Y)p(Y)$$

$$p(X) = \int p(X|Y)p(Y)dY$$

# Bayes theorem

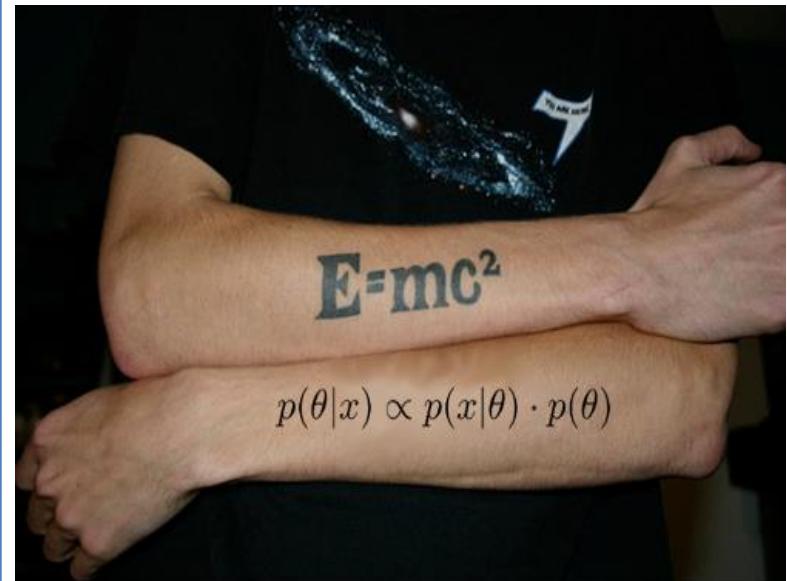
- Combination 2:

## Bayes Theorem

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)}$$

$$p(Y|X) \propto p(X|Y)p(Y)$$

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X|Y)p(Y)dY}$$



# Bayesian probabilities

- Probability reflects your knowledge (uncertainty) about a phenomenon → **subjective probabilities**
  - **Prior probability**  $p(w)$ , can be uninformative  $p(w) \propto 1$
  - Formulate a model, compute **likelihood**  $p(D|w)$
  - **Posterior probability**  $p(w|D)$ , after observing data
    - $p(w|D) \propto p(D|w)p(w)$
- Model parameters are considered as random variables
  - In real life, do not need to be random, but we model as random

# Basic ML ingredients

- Data  $D$ : observations
  - Features  $X_1, \dots, X_p$
  - Targets  $Y_1, \dots, Y_r$
- Model  $P(x|w_1, \dots, w_k)$  or  $P(y|x, w_1, \dots, w_k)$ 
  - Example: Linear regression  $p(y|x, w) = N(w_0 + w_1 x, \sigma^2)$
- Learning procedure (data → get parameters  $\hat{w}$  or  $p(w|D)$ )
  - Maximum likelihood, MAP, Bayes rule...
- Predict new data  $X^{new}$  by using the fitted model

Case	$X_1$	$X_2$	$Y$
1			
2			
...			

# Probabilistic models

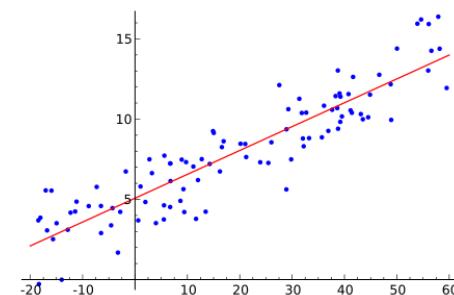
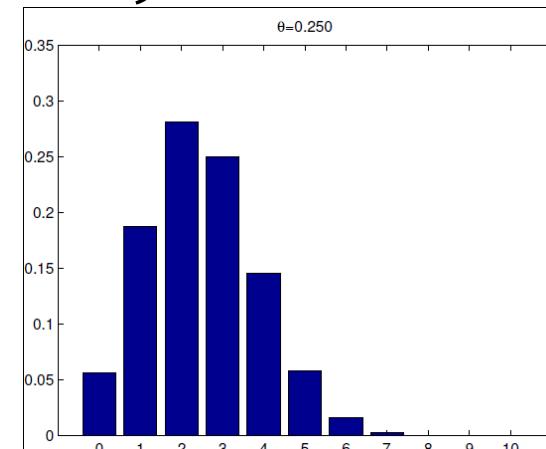
- A distribution  $p(x|w)$  or  $p(y|x, w)$

- Example:

- $x \sim Bin(n, \theta)$

$$p(x = k|n, \theta) = \binom{n}{k} \theta^k (1 - \theta)^{n-k}$$

- $y \sim N(\alpha_0 + \alpha_1 x, \sigma^2)$



Learn basic distributions and their properties → PRML, chapter 2!

Source: Wikipedia

# Fitting a model

- Frequentist principle: Maximum likelihood principle
  - Compute likelihood  $p(\mathbf{D} | w)$   
$$p(\mathbf{D} | w) = \prod_{i=1}^n p(X_i | w) \text{ or } p(\mathbf{D} | w) = \prod_{i=1}^n p(Y_i | X_i, w)$$
  - Maximize the likelihood and find the optimal  $w^* \rightarrow$  they are the fitted values

## Remarks:

- Likelihood shows how much the chosen parameter value is proper for a specific model and the given data
- Normally **log-likelihood** is used in computations instead
- Other alternatives to ML exist...

# Fitting a model

- Bayesian principle
  - Compute  $p(w|D)$  and then decide yourself what to do with this (for ex. MAP, mean, median)
- Use bayes theorem

$$p(w|D) = \frac{p(D|w)p(w)}{p(D)} \propto p(D|w)p(w)$$
- $p(D)$  is **marginal likelihood**
  - $p(D) = \int p(D|w)p(w)dw$  or
  - $p(D) = \sum_i p(D|w_i)p(w_i)$

**Example:** tossing a coin. Find  $p(\theta|D)$ , estimate various  $\theta^*$

# Fitting a model

- How to chose the prior?
  - Expert knowledge about the phenomenon
  - Forcing a model to have a certain structure
    - Example: decision trees: prior prefers smaller trees
  - Conjugacy [http://en.wikipedia.org/wiki/Conjugate\\_prior](http://en.wikipedia.org/wiki/Conjugate_prior)
    - Distribution of the posterior is the same type as the distribution of the likelihood or prior
- Prior is the most controversial about Bayesian methods, but
  - When  $N \rightarrow \infty$ , data overwhelms the prior

# Prediction

- **Plug-in estimation** (Frequentist and Bayesian)
  - Substitute the estimated  $w^*$  into  $p(\mathbf{x}|w)$  or  $p(y|\mathbf{x}^{new}, w)$
- **Bayesian model averaging**
  - Posterior predictive distribution:
    - $p(\mathbf{x}^{new}|D) = \int p(\mathbf{x}^{new}|w)p(w|D)dw$
    - $p(\mathbf{y}|D) = \int p(\mathbf{y}|w, \mathbf{x}^{new})p(w|D, \mathbf{x}^{new})dw$

# Black swan paradox

- In the coin example,  $p(x^{new} = 1) = \frac{k}{n}$  if MLE used
- If we made 3 attempts, no successes  $\rightarrow k=0$
- Does this mean  $p(x^{new} = 1) = 0$  ??
- Problem does not appear in Bayesian setting (posterior mean)

# Types of supervised models

- **Generative models:** model  $p(X|Y, w)$  and  $p(Y|w)$ 
  - Example: k-NN classification

$$p(X = x|Y = C_i, K) = \frac{K_i}{N_i V}, p(C_i|K) = \frac{N_i}{N}$$

From Bayes Theorem,

$$p(Y = C_i|x, K) \propto \frac{K_i}{K}$$

- **Discriminative models:** model  $p(Y|X, w)$ ,  $X$  constant
  - Example: logistic regression
  - $p(Y = 1|w, x) = \frac{1}{1+e^{-w^T x}}$

# Generative vs Discriminative

- Generative can be used to generate new data
- Generative normally easier to fit (check Logistic vs K-NN)
- Generative: each class estimated separately → do not need to retrain when a new class added
- Discriminative models: can replace  $X$  with  $\phi(X)$  (preprocessing), method will still work
  - Not generative, distribution will change
- Generative: often make too strong assumptions about  $p(X|Y, w) \rightarrow$  bad performance

# Bayesian decision theory

- Machine learning models estimate  $p(y|x)$  or  $p(y|x, \hat{w})$
- Transform probability into action → which value to predict? → decision step
  - $p(Y = \text{Spam}|x) = 0.83 \rightarrow$  do we move the mail to Junk?
  - What is more dangerous: deleting 1 non-spam mail or letting 1 spam mail enter Inbox?
- → **Loss function** or **Loss matrix**

# Loss matrix

- Costs of classifying  $Y = C_k$  to  $C_j$ :
  - Rows: true, columns: predicted

$$L = \|L_{ij}\|, i = 1, \dots, n, j = 1, \dots, n$$

- Example 1: 0/1-loss

$$L = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

- Example 2: Spam

$$L = \begin{pmatrix} 0 & 100 \\ 1 & 0 \end{pmatrix}$$

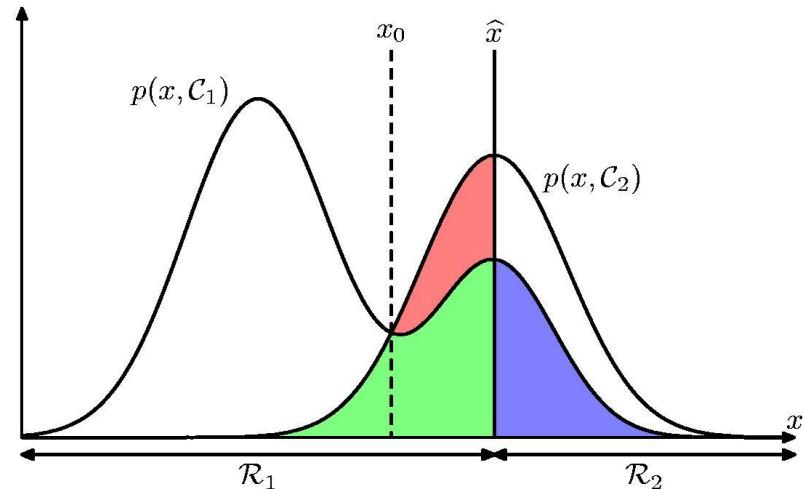
# Loss and decision

- Expected loss minimization

- $R_j$  : classify to  $C_j$

$$EL = \sum_k \sum_j \int_{R_j} L_{kj} p(x, C_k) dx$$

- Choose such  $R_j$  that  $EL$  is minimized
- Two classes



$$EL = \int_{R_1} L_{21} p(x, C_2) dx + \int_{R_2} L_{12} p(x, C_1) dx$$

# Loss and decision

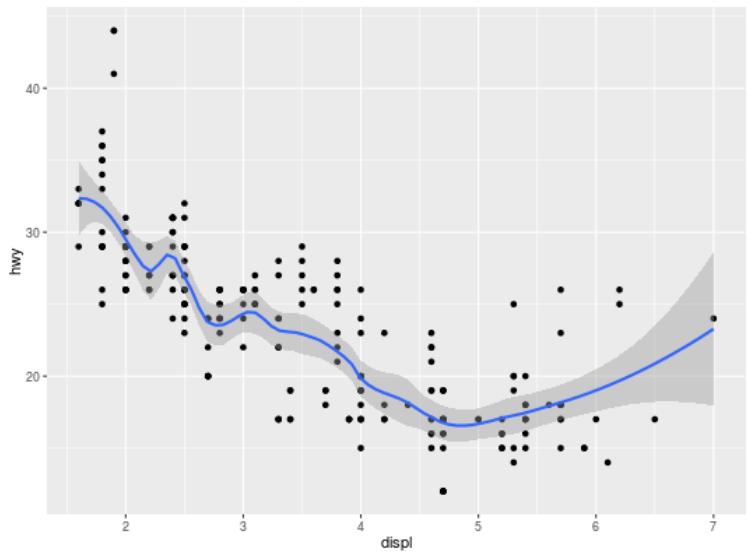
- How to minimize  $EL$ ?
  - We free to assign  $x$  to either  $R_1$  or  $R_2$
  - Assigning  $x$  to region with smallest  $L_{ij}p(x, C_i)$  will make  $EL$  smaller
- → Rule:
  - $L_{21}p(x, C_1) > L_{12}p(x, C_2) \rightarrow$  predict  $y$  as  $C_1$
- 0/1 Loss: classify to the class which is more probable!

$$\frac{p(C_1|x)}{p(C_2|x)} > \frac{L_{12}}{L_{21}} \rightarrow \text{predict } y \text{ as } C_1$$

# Loss and decision

- Continuous targets: squared loss
  - Given a model  $p(x, y)$ , minimize
$$EL = \int L(y, \hat{Y}(x)) p(x, y) dx dy$$
- Using **square loss**, the optimal is posterior mean

$$\hat{Y}(x) = \int y p(y|x) dy$$



# ROC curves

- Binary classification
- The choice of the threshold  $\hat{x} = \frac{L_{12}}{L_{21}}$  affects prediction → what if we don't know the loss? Which classifier is better?
- **Confusion matrix**

		PREDICTED		
		1	0	Total
T R U E	1	TP	FN	$N_+$
	0	FP	TN	$N_-$

# ROC curves

- **True Positive Rates (TPR) = sensitivity = recall**
  - Probability of detection of positives: TPR=1 positives are correctly detected
- **False Positive Rates (FPR)**
  - Probability of false alarm: system alarms (1) when nothing happens (true=0)

$$TPR = TP/N_+$$

**(FPR)**

- Probability of false alarm: system alarms (1) when nothing happens (true=0)

$$FPR = FP/N_-$$

- **Specificity**

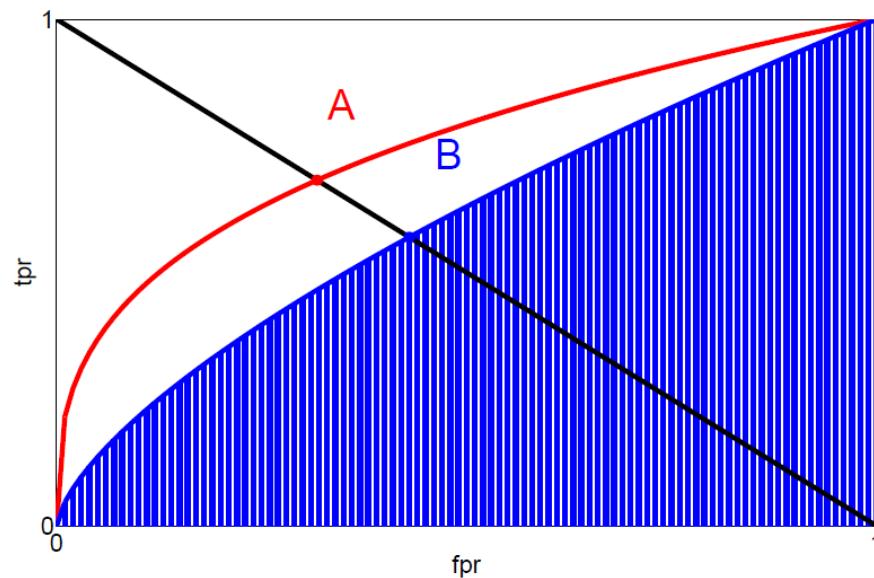
$$\text{Specificity} = 1 - FPR$$

- **Precision**

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

# ROC curves

- **ROC**=Receiver operating characteristics
- Use various thresholds, measure TPR and FPR
- Same FPR, higher TPR → better classifier
- Best classifier = greatest Area Under Curve (**AUC**)



# Regression and regularization

- Linear regression
- Ridge Regression Lecture 2a
- Lasso
- Variable selection

# Simple linear regression

## Model:

$$y \sim N(w_0 + w_1 x, \sigma^2)$$

or

$$\begin{aligned}y &= w_0 + w_1 x + \epsilon, \\ \epsilon &\sim N(0, \sigma^2)\end{aligned}$$

or

$$p(y|x, w) = N(w_0 + w_1 x, \sigma^2)$$

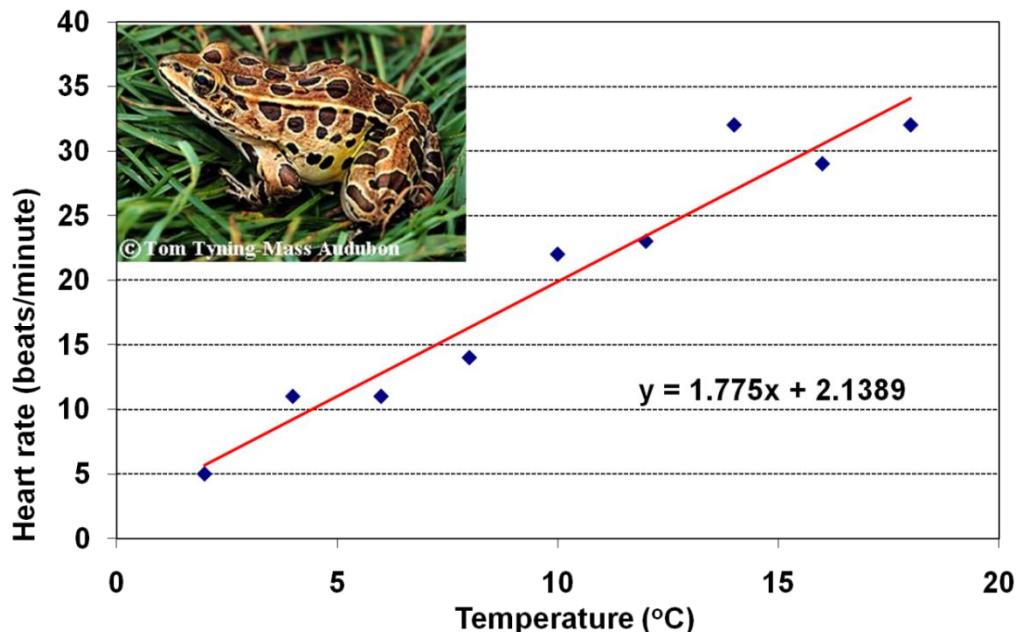
## Terminology:

$w_0$ : intercept (or bias)

$w_1$ : regression coefficient

## Response

The target responds directly and linearly to changes in the feature



# Ordinary least squares regression (OLS)

Model:

$$y \sim N(w^T x, \sigma^2)$$

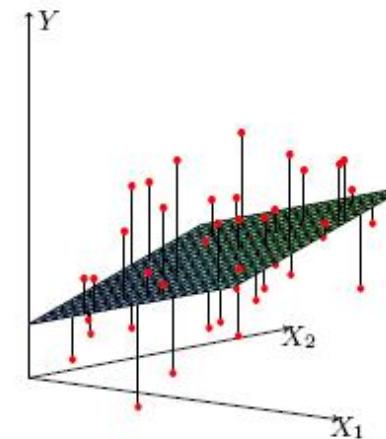
where

$$w = \{w_0, \dots, w_d\}$$

$$x = \{1, x_1, \dots, x_d\}$$



Why is "1" here?



The response variable responds directly and linearly to changes in each of the inputs

# Ordinary least squares regression

Given data set  $D$

Case	$X_1$	$X_2$			$X_p$	$Y$
1	$x_{11}$	$x_{21}$			$x_{p1}$	$y_1$
2	$x_{12}$	$x_{22}$			$x_{p2}$	$y_2$
3	$x_{13}$	$x_{23}$			$x_{p3}$	$y_3$
$N$	$x_{1N}$	$x_{2N}$			$x_{pN}$	$y_N$

**Estimation:** maximizing the likelihood

$$\hat{w} = \max_w p(D|w)$$

Is equivalent to minimizing

$$RSS(w) = \sum_{i=1}^n (Y_i - \mathbf{w}^T \mathbf{X}_i)^2$$

# Matrix formulation of OLS regression

Optimality condition:

where

$$\mathbf{X}^T (\mathbf{y} - \mathbf{X}\mathbf{w}) = 0$$

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{21} & & x_{p1} \\ 1 & x_{12} & x_{22} & & x_{p2} \\ & & & & \\ 1 & x_{1N} & x_{2N} & & x_{pN} \end{pmatrix} \quad \text{and} \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

# Parameter estimates and predictions

- Least squares estimates of the parameters

$$\hat{w} = (X^T X)^{-1} X^T y$$

- Predicted values

$$\hat{y} = X\hat{w} = X(X^T X)^{-1} X^T y = Py$$



Hat matrix

- Linear regression belongs to the class of **linear smoothers**

Why is it called so?

# Degrees of freedom

Definition:

$$df(\hat{y}) = \frac{1}{\sigma^2} \sum_{i=1}^N Cov(\hat{y}_i, y_i)$$

- Larger covariance → stronger connection → model can approximate data better → model more flexible (complex)
- For linear smoothers  $\hat{Y} = S(X)Y$

$$df = \text{trace}(S)$$

- For linear regression, degrees of freedom is

$$df = \text{trace}(P) = p$$

# Different types of features

- Interval variables
- Numerically coded ordinal variables
  - (small=1, medium=2, large=3)
- Dummy coded qualitative variables

**Example of dummy coding:**

$$x_1 = \begin{cases} 1, & \text{if Jan} \\ 0, & \text{otherwise} \end{cases}$$

$$x_2 = \begin{cases} 1, & \text{if Feb} \\ 0, & \text{otherwise} \end{cases}$$

.

.

.

$$x_{11} = \begin{cases} 1, & \text{if Nov} \\ 0, & \text{otherwise} \end{cases}$$

**Basis function expansion:**

$$\text{If } y = w_0 + w_1 x_1 + w_2 x_1^2 + w_3 e^{-x_2} + \epsilon,$$

Model becomes linear if to recompute:

$$\phi_1(x_1) = x_1$$

$$\phi_2(x_1) = x_1^2$$

$$\phi_3(x_1) = e^{-x_2}$$

# Basis function expansion

- In general  $\phi_1(\dots)$  may be a function of several  $x$  components
- Having data given by  $\mathbf{X}$ , compute new data

$$\Phi = \begin{pmatrix} 1 & \phi_1(x_{11}, \dots, x_{1p}) & \dots & \phi_p(x_{11}, \dots, x_{1p}) \\ & \dots & \dots & \dots \\ 1 & \phi_1(x_{n1}, \dots, x_{np}) & \dots & \phi_p(x_{n1}, \dots, x_{np}) \end{pmatrix}$$

- If doing a basis function in a model, replace  $\mathbf{X}$  by  $\Phi$  everywhere where  $\mathbf{X}$  is used:

$$\hat{\mathbf{y}} = \Phi(\Phi^T \Phi)^{-1} \Phi^T \mathbf{y}$$

# Linear regression in R

- `fit=lm(formula, data, subset, weights,...)`
  - **data** is the data frame containing the predictors and response values
  - **formula** is expression for the model
  - **subset** which observations to use (training data)?
  - **weights** should weights be used?

**fit** is object of class **lm** containing various regression results.

- Useful functions (many are generic, used in many other models)
  - Get details about the particular function by ":", for ex. `predict.lm`

```
summary(fit)
predict(fit, newdata, se.fit, interval)
coefficients(fit) # model coefficients
confint(fit, level=0.95) # CIs for model parameters
fitted(fit) # predicted values
residuals(fit) # residuals
```

# An example of ordinary least squares regression

```
mydata=read.csv2("Bilexempel.csv")
fit1=lm(Price~Year, data=mydata)
summary(fit1)
fit2=lm(Price~Year+Mileage+Equipment,
data=mydata)
summary(fit2)
```

```
> summary(fit1)

Call:
lm(formula = Price ~ Year, data = mydata)

Residuals:
    Min      1Q  Median      3Q     Max 
-167683 -14683  20056  35933  72317 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -78161027   8448038  -9.252 6.00e-13 ***
Year          39246      4226   9.288 5.25e-13 ***  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 57270 on 57 degrees of freedom
Multiple R-squared:  0.6021, Adjusted R-squared:  0.5952 
F-statistic: 86.26 on 1 and 57 DF,  p-value: 5.248e-13
```

## Response variable:

Requested price of used Porsche cars  
(1000 SEK)

## Inputs:

$X_1$  = Manufacturing year  
 $X_2$  = Milage (km)  
 $X_4$  = Equipment (0 or 1)

# An example of ordinary least squares regression

```
> summary(fit2)

call:
lm(formula = Price ~ Year + Mileage + Equipment, data = mydata)

Residuals:
    Min      1Q  Median      3Q     Max 
-66223 -10525   -739  14128  65332 

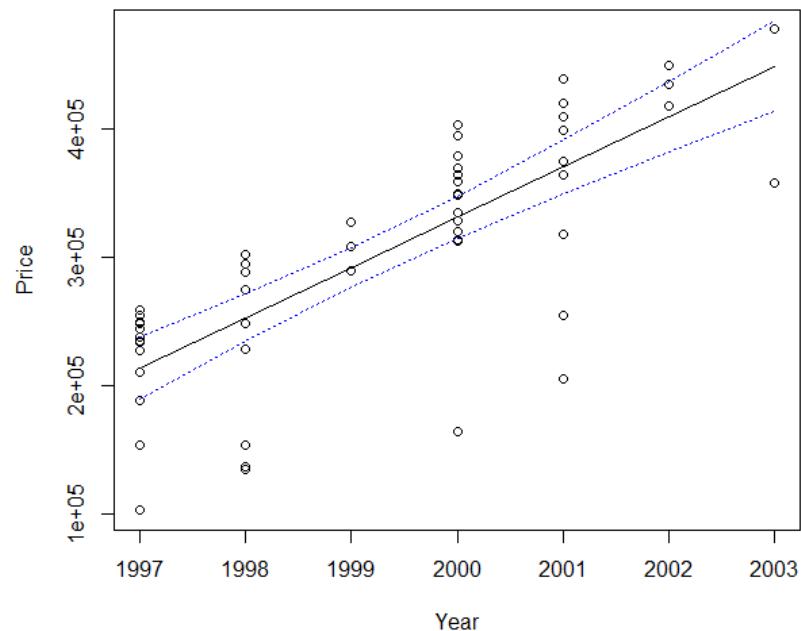
Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -2.083e+07  6.309e+06  -3.302  0.00169 ***
Year         1.062e+04  3.154e+03   3.366  0.00139 ***
Mileage      -2.077e+00  2.022e-01 -10.269 2.14e-14 ***
Equipment   5.790e+04  1.041e+04   5.563 8.08e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 29270 on 55 degrees of freedom
Multiple R-squared:  0.8997, Adjusted R-squared:  0.8942 
F-statistic: 164.5 on 3 and 55 DF,  p-value: < 2.2e-16
```

# An example of ordinary least squares regression

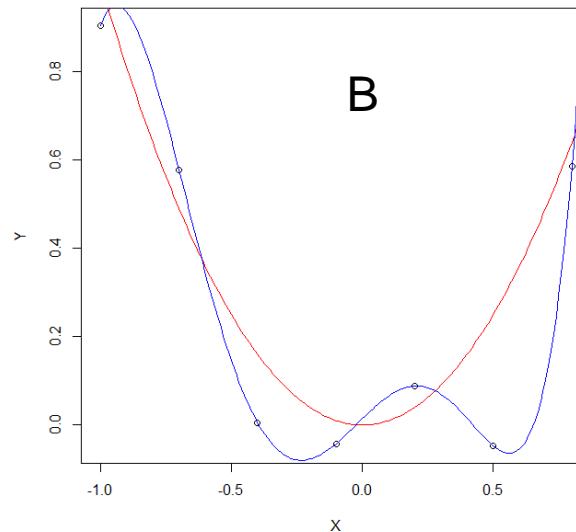
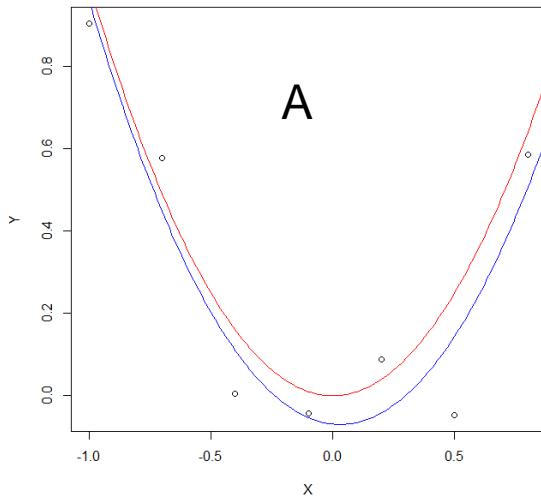
- Prediction

```
fitted <- predict(fit1, interval =  
"confidence")  
  
# plot the data and the fitted line  
attach(mydata)  
plot(Year, Price)  
lines(Year, fitted[, "fit"])  
  
# plot the confidence bands  
lines(Year, fitted[, "lwr"], lty = "dotted",  
col="blue")  
lines(Year, fitted[, "upr"], lty = "dotted",  
col="blue")  
detach(mydata)
```



# Ridge regression

- Problem: linear regression can overfit:
  - Take  $Y := Y, X_1 = X, X_2 = X^2, \dots, X_p = X^p \rightarrow$  polynomial model, fit by linear regression
  - High degree of polynomial leads to overfitting.



# Ridge regression

- **Idea:** Keep all predictors but shrink coefficients to make model less complex

$$\text{minimize } -\log \text{likelihood} + \lambda_0 \|w\|_2^2$$

→  **$L_2$  regularization**

- Given that model is Gaussian, we get **Ridge regression:**

$$\hat{w}^{ridge} = \operatorname{argmin} \left\{ \sum_{i=1}^N (y_i - w_0 - w_1 x_{1j} - \dots - w_p x_{pj})^2 + \lambda \sum_{j=1}^p w_j^2 \right\}$$

- $\lambda > 0$  is **penalty factor**

# Ridge regression

Equivalent form

$$\hat{w}^{ridge} = \operatorname{argmin} \sum_{i=1}^N (y_i - w_0 - w_1 x_{1j} - \dots - w_p x_{pj})^2$$

**subject to**  $\sum_{j=1}^p w_j^2 \leq s$

*Solution*

$$\hat{w}^{ridge} = (X^T X + \lambda I)^{-1} X^T y$$

$$\hat{y} = X \hat{w} = X (X^T X + \lambda I)^{-1} X^T y = P y$$

Hat matrix

How do we  
compute degrees  
of freedom here?

# Ridge regression

## Properties

- Extreme cases:
  - $\lambda = 0$  usual linear regression (no shrinkage)
  - $\lambda = +\infty$  fitting a constant ( $w = 0$  except of  $w_0$ )
- When input variables are orthogonal (not realistic),  $X^T X = I \rightarrow \hat{w}^{\text{ridge}} = \frac{1}{1+\lambda} w^{\text{linreg}} \rightarrow$  coefficients are equally shrunk
- **Ridge regression is particularly useful if the explanatory variables are strongly correlated to each other.**
  - Correlated variables often correspond large  $w \rightarrow$  shrunk
- Degrees of freedom decrease when  $\lambda$  increases
  - $\lambda = 0 \rightarrow d.f. = p$

# Ridge regression

## Properties

- Shrinking enables estimation of regression coefficients even if the number of parameters exceeds the number of cases!  
 $(X^T X + \lambda I)$  is always nonsingular
  - Compare with linear regression
- How to estimate  $\lambda$ ?
  - cross-validation

# Ridge regression

- Bayesian view
  - Ridge regression is just a special form of Bayesian Linear Regression with constant  $\sigma^2$ :

$$\begin{aligned} \mathbf{y} &\sim N(\mathbf{y} | \mathbf{w}_0 + \mathbf{X}\mathbf{w}, \sigma^2) \\ \mathbf{w} &\sim N\left(\mathbf{0}, \frac{\sigma^2}{\lambda} \mathbf{I}\right) \end{aligned}$$

**Theorem** MAP estimate to the Bayesian Ridge is equal to solution in frequentist Ridge

$$\hat{\mathbf{w}}_{ridge} = \mathbf{X}^T (\mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

- In Bayesian version, we can also make inference about  $\lambda$

**Example Computer Hardware Data Set** : performance measured for various processors and also

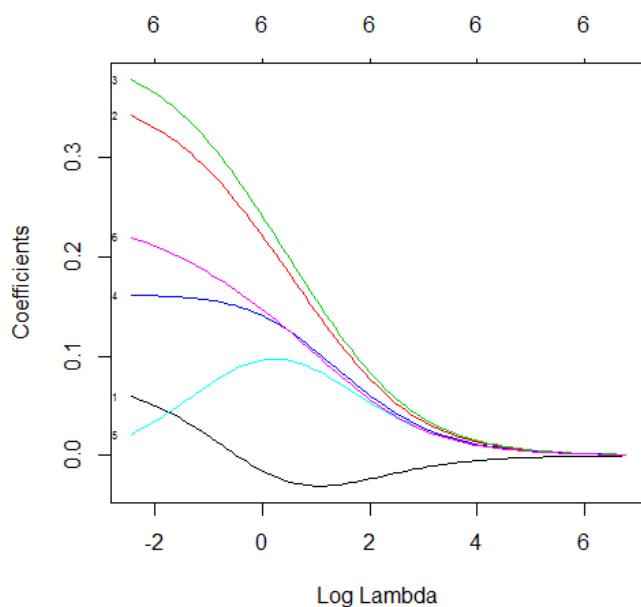
- Cycle time
- Memory
- Channels

# Ridge regression

- R code: use package **glmnet** with alpha=0 (Ridge regression)
- Seeing how Ridge converges

```
data=read.csv("machine.csv", header=F)
covariates=scale(data[,3:8])
response=scale(data[, 9])

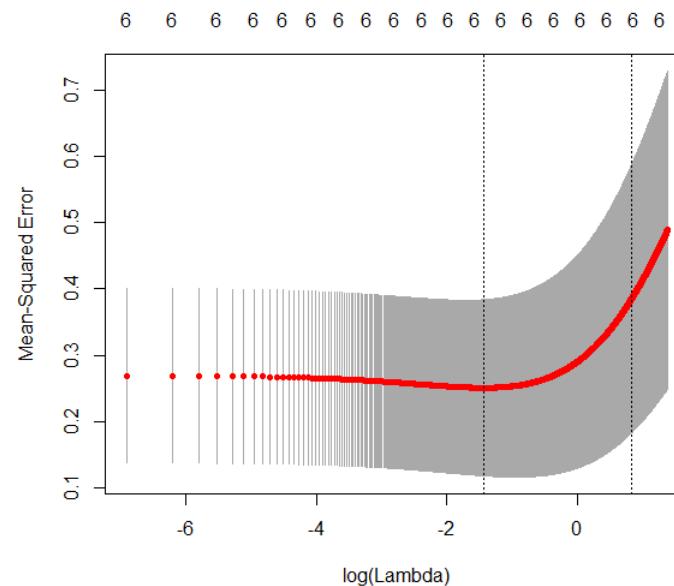
model0=glmnet(as.matrix(covariates),
response, alpha=0,family="gaussian")
plot(model0, xvar="lambda", label=TRUE)
```



# Ridge regression

- Choosing the best model by cross-validation:

```
model=cv.glmnet(as.matrix(covariates),  
response, alpha=0,family="gaussian")  
model$lambda.min  
plot(model)  
coef(model, s="lambda.min")  
  
> coef(model, s="lambda.min")  
7 x 1 sparse Matrix of class "dgCM"  
1  
(Intercept) -4.530442e-17  
v3 3.420739e-02  
v4 3.085696e-01  
v5 3.403839e-01  
v6 1.593470e-01  
v7 5.489116e-02  
v8 1.970982e-01
```



```
> model$lambda.min  
[1] 0.046
```

# Ridge regression

- How good is this model in prediction?

```
ind=sample(209, floor(209*0.5))
data1=scale(data[,3:9])
train=data1[ind,]
test=data1[-ind,]

covariates=train[,1:6]
response=train[, 7]
model=cv.glmnet(as.matrix(covariates), response, alpha=1,family="gaussian",
lambda=seq(0,1,0.001))
y=test[,7]
ynew=predict(model, newx=as.matrix(test[, 1:6]), type="response")
```

```
#Coefficient of determination
sum((ynew-mean(y))^2)/sum((y-mean(y))^2)

sum((ynew-y)^2)
```

```
> sum((ynew-mean(y))^2)/sum((y-mean(y))^2)
[1] 0.5438148
> sum((ynew-y)^2)
[1] 18.04988
> |
```

Note that data are so small so numbers change much for other train/test

# LASSO

- **Idea:** Similar idea to Ridge
- Minimize minus loglikelihood plus **linear** penalty factor  $\rightarrow \mathbf{L_1}$  regularization
  - Given that model is Gaussian, we get **LASSO** (least absolute shrinkage and selection operator):

$$\hat{w}^{lasso} = \operatorname{argmin} \left\{ \sum_{i=1}^N (y_i - w_0 - w_1 x_{1j} - \dots - w_p x_{pj})^2 + \lambda \sum_{j=1}^p |w_i| \right\}$$

- $\lambda > 0$  is **penalty factor**
- Equivalently

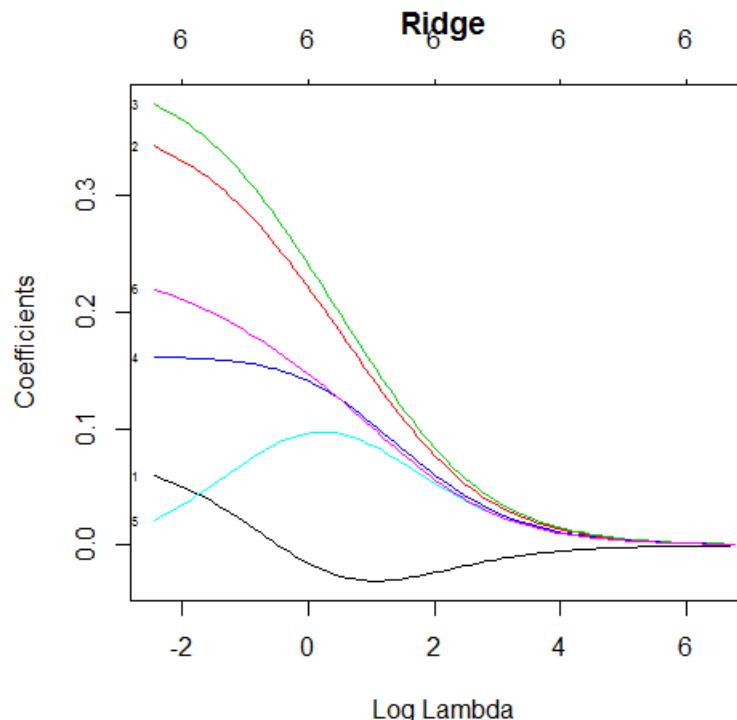
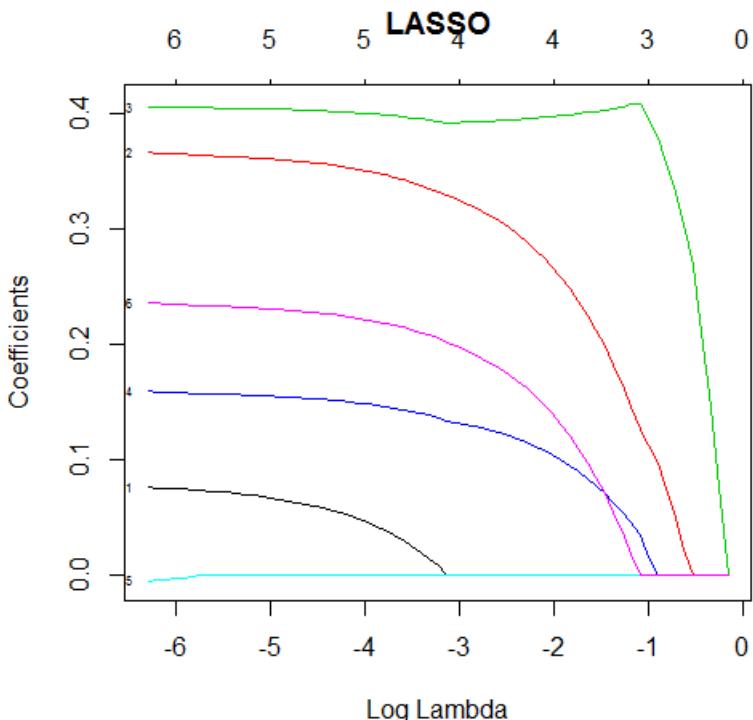
$$\hat{w}^{lasso} = \operatorname{argmin} \sum_{i=1}^N (y_i - w_0 - w_1 x_{1j} - \dots - w_p x_{pj})^2$$

**subject to**  $\sum_{j=1}^p |w_i| \leq s$

# LASSO vs Ridge

- LASSO yields sparse solutions!

**Example** Computer hardware data



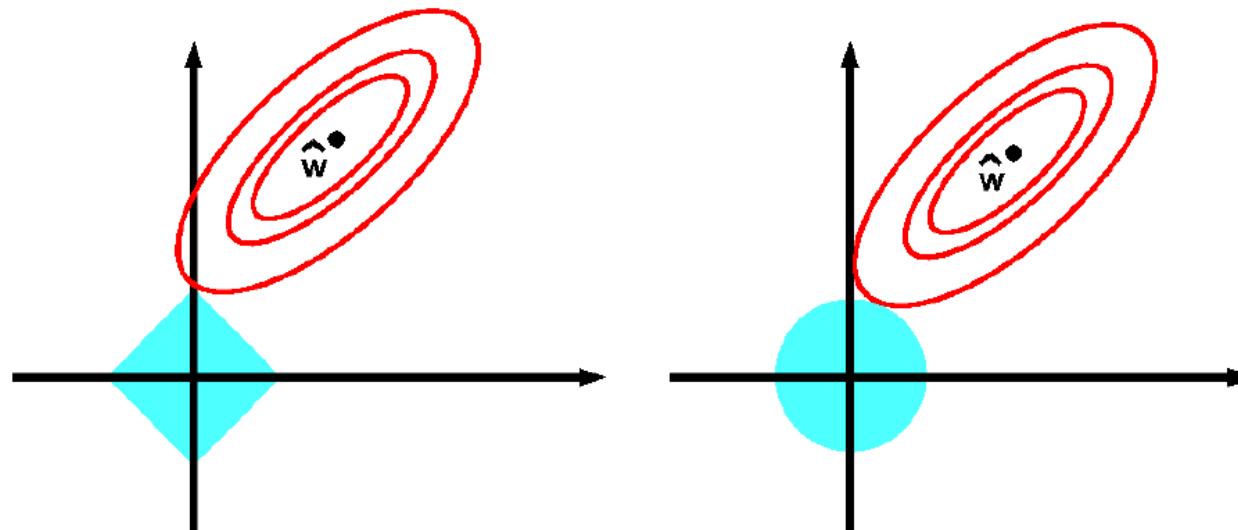
# LASSO vs Ridge

- Only 5 variables selected by LASSO

```
> coef(model, s="lambda.min")
7 x 1 sparse Matrix of class "dgCMatrix"
   [1]
(Intercept) -5.091825e-17
v3           6.350488e-02
v4           3.578607e-01
v5           4.033670e-01
v6           1.541329e-01
v7           .
v8           2.287134e-01
> 1
> sum((ynew-mean(y))^2)/sum((y-mean(y))^2)
[1] 0.5826904
> sum((ynew-y)^2)
[1] 16.63756
```

# LASSO vs Ridge

- Why Lasso leads to sparse solutions?
  - Feasible area for Ridge is a circle (2D)
  - Feasible area for LASSO is a polygon (2D)



# LASSO properties

- Lasso is widely used when  $p \gg n$ 
  - Linear regression breaks down when  $p > n$
  - Application: DNA sequence analysis, Text Prediction

- When inputs are orthonormal,

$$\hat{w}_i^{\text{lasso}} = \text{sign}(w_i^{\text{linreg}}) \left( |w_i^{\text{linreg}}| - \frac{\lambda}{2} \right)_+$$

- No explicit formula for  $\hat{w}^{\text{lasso}}$

- Optimization algorithms used

Coding in R: use  
`glmnet()` with  
`alpha=1`

# Variable selection

- .. Or “Feature selection”

Often, we do not need all features available in the data to be in the model

## Reasons:

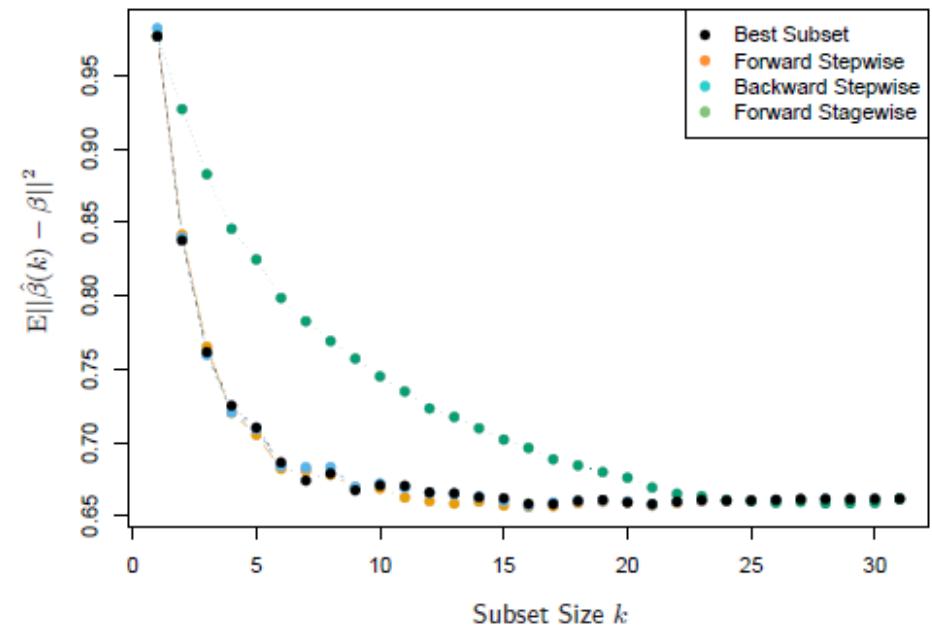
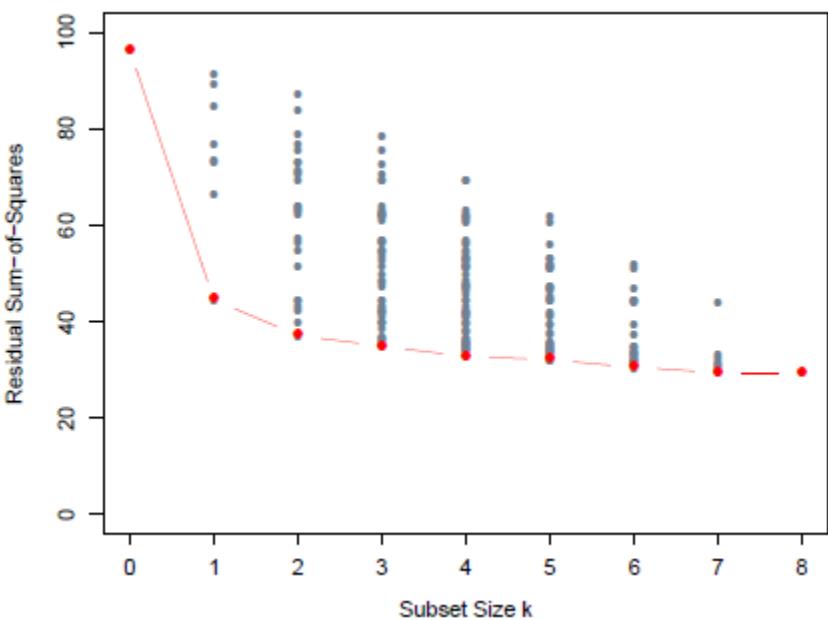
- Model can become overfitted (recall polynomial regression)
- Large number of predictors → model is difficult to use and interpret

# Variable selection

## Alternative 1: Variable subset selection

- Best subset selection:
  - Consider different subsets of the full set of features, fit models and evaluate their quality
    - Problem: computationally difficult for  $p$  around 30 or more
    - How to choose the best model size? Some measure of predictive performance normally used (ex. AIC).
- Forward and Backward stepwise selection
  - Starts with 0 features (or full set) and then adds a feature (removes feature) that most improves the measure selected.
    - Can handle large  $p$  quickly
    - Does not examine all possible subsets (not the “best”)

# RSS and MSE depend on k



# Variable selection in R

- Use stepAIC() in MASS

```
library(MASS)
fit <- lm(V9~., data=data.frame(data1))
step <- stepAIC(fit, direction="both")
step$anova
summary(step)
```

```
Call:
lm(formula = V9 ~ V3 + V4 + V5 + V6 + V8, data = data.fra
```

```
Residuals:
    Min      1Q      Median      3Q      Max 
-1.20232 -0.15512  0.03579  0.16567  2.42280
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -5.783e-17 2.574e-02  0.000  1.0000    
V3          7.948e-02 2.826e-02  2.813  0.0054 **  
V4          3.661e-01 4.312e-02  8.490 4.34e-15 *** 
V5          4.055e-01 4.664e-02  8.695 1.18e-15 *** 
V6          1.591e-01 3.394e-02  4.687 5.07e-06 *** 
V8          2.360e-01 3.356e-02  7.031 3.06e-11 ***
```

```
> step <- stepAIC(fit, direction="both")
Start:  AIC=-405.35
V9 ~ V3 + V4 + V5 + V6 + V7 + V8
```

	Df	Sum of Sq	RSS	AIC
- V7	1	0.0139	28.117	-407.25
<none>			28.103	-405.35
- V3	1	1.0819	29.185	-399.46
- V6	1	2.9385	31.041	-386.57
- V8	1	6.3150	34.418	-364.99
- V4	1	9.7492	37.852	-345.11
- V5	1	10.4837	38.586	-341.09

```
Step:  AIC=-407.25
V9 ~ V3 + V4 + V5 + V6 + V8
```

	Df	Sum of Sq	RSS	AIC
<none>			28.117	-407.25
+ V7	1	0.0139	28.103	-405.35
- V3	1	1.0958	29.212	-401.26
- V6	1	3.0431	31.160	-387.77
- V8	1	6.8472	34.964	-363.70
- V4	1	9.9840	38.101	-345.74
- V5	1	10.4713	38.588	-343.08

# Model selection

Lecture 2b

# Frequentist vs Bayesian

- Probabilistic Model  $p(y, x, w)$ 
  - Frequentists:  $w$  is a parameter that should be estimated by model fitting
  - Bayesians:  $w$  is a random variable that has a prior distribution  $p(w)$ 
    - How to set  $p(w)??$

**Example:** Linear regression, what are parameters here?

$$y \sim w_0 + \mathbf{w}x + e, e \sim N(0, \sigma^2)$$
$$y \sim N(w_0 + \mathbf{w}x, \sigma^2)$$

# An estimator

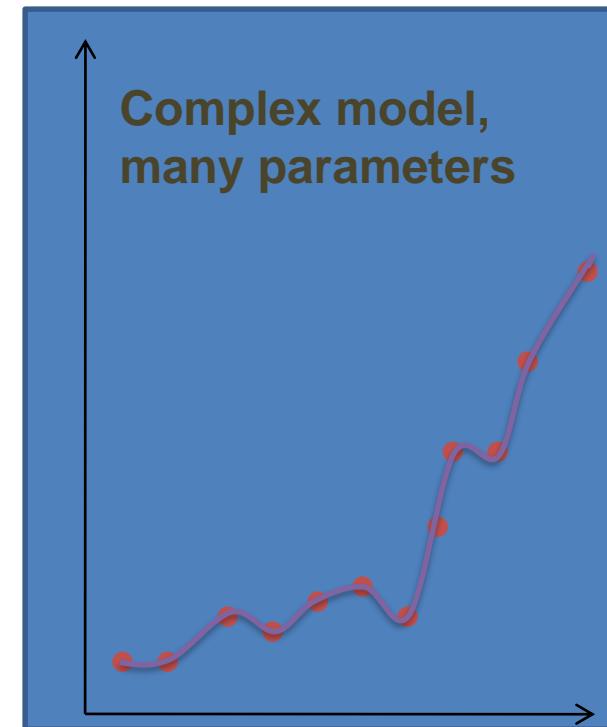
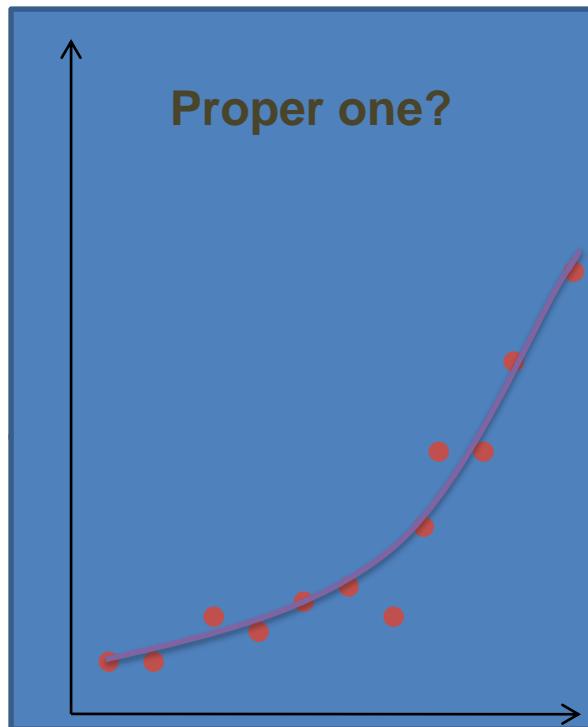
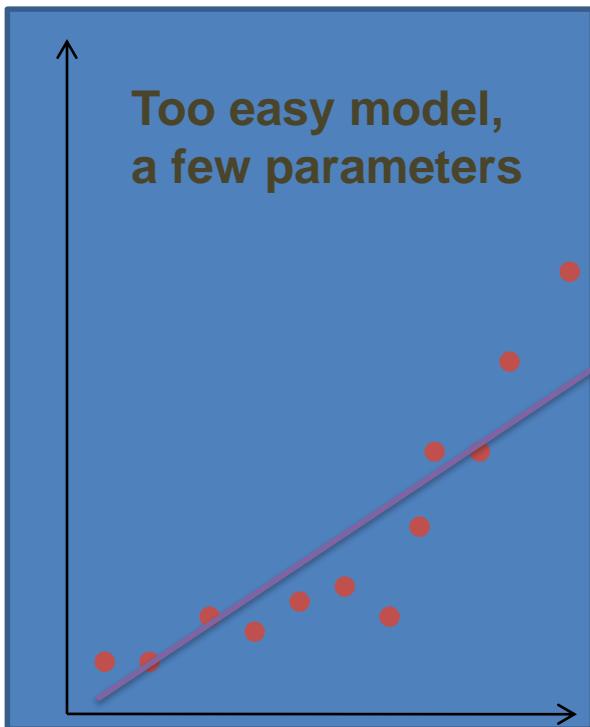
- $\hat{w} = \delta(D)$  (some function of your data) – an **estimator**
- Optimal parameter values? → there can be many ways to compute them (MLE, shrinkage...)
  - Compare Bayesian: given estimators  $w^1$  and  $w^2$ , we **can** compare them!  $p(w^1|D) > p(w^2|D)$
  - There is no easy way to compare estimators in frequentist tradition

**Example:** Linear regression

- Estimator 1:  $w = (X^T X)^{-1} X^T Y$  (maximum likelihood)
- Estimator 2:  $w = (0, \dots, 0, 1)$
- Which one is better?
  - A comparison strategy is needed!

# Overfitting

- Complex model can overfit your data



# Overfitting: solutions

- **Observed:** Maximum likelihood can lead to overfitting.
- **Solutions**
  - Selecting proper parameter values
    - Regularized risk minimization
  - Selecting proper model type, for ex. number of parameters
    - Houldout method
    - Cross-validation

# Model selection

- Given a model, choose the optimal parameter values
  - Decision theory
- Define loss  $L(Y, \hat{y})$ 
  - How much we loose in guessing true Y incorrectly
- If we know the true distribution  $p(y, x|w)$  then we choose  $\hat{y}$

$$\min_{\hat{f}} EL(y, \hat{y}) = \min_{\hat{y}} \int L(y, \hat{y}) p(y, x|w) dx dy$$

# Loss functions

- How to define loss function?
  - No unique choice, often defined by application
  - **Normal practice: Choose the loss related to minus loglikelihood**

**Example:** Predicting the amount of the product at the storage:

$$L(Y, \hat{y}) = \begin{cases} 10, & +\hat{y}/Y \geq Y \\ 1000, & \hat{y} < Y \end{cases}$$

**Example:** Compute loss function related to

- Normal distribution

Guess why such loss  
function was chosen

# Loss functions

- Classification problems

- Common loss function  $L(Y, \hat{y}) = \begin{cases} 1, & Y = \hat{y} \\ 0, & Y \neq \hat{y} \end{cases}$
- When minimizing the loss, equivalent to misclassification rate

# Model selection

- **Problem:** true model and true  $w$  are unknown → can not compute expected loss!
- How to find an optimal model?
  - Consider what expected loss (**risk**) depends on
$$R(Y, \hat{y}) = E[L(Y, \hat{y}(X, D))]$$
- Random factors:
  - $D$  – **training set**
  - $Y, X$  – data to be predicted (**validation set**)

# Holdout method

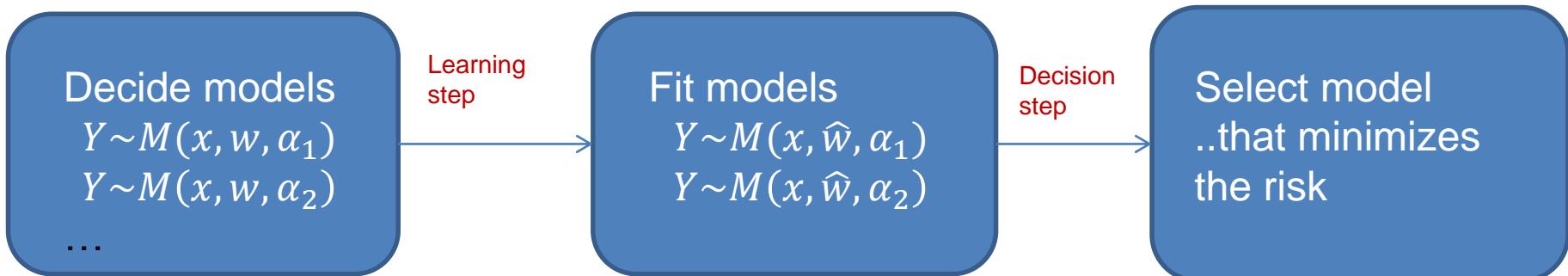
- Simplify the risk estimation:
  - Fix D as a particular training set T
  - Fix Y,X as a particular validation set V
- Risk becomes (**empirical risk**)

$$\hat{R}(y, \hat{y}) = \frac{1}{|V|} \sum_{(X,Y) \in V} L(Y, \hat{y}(X, T))$$

- Estimator is fit by Maximum Likelihood using training set
- Risk estimated by using validation set
- Model with minimum empirical risk is selected

# General model selection strategy

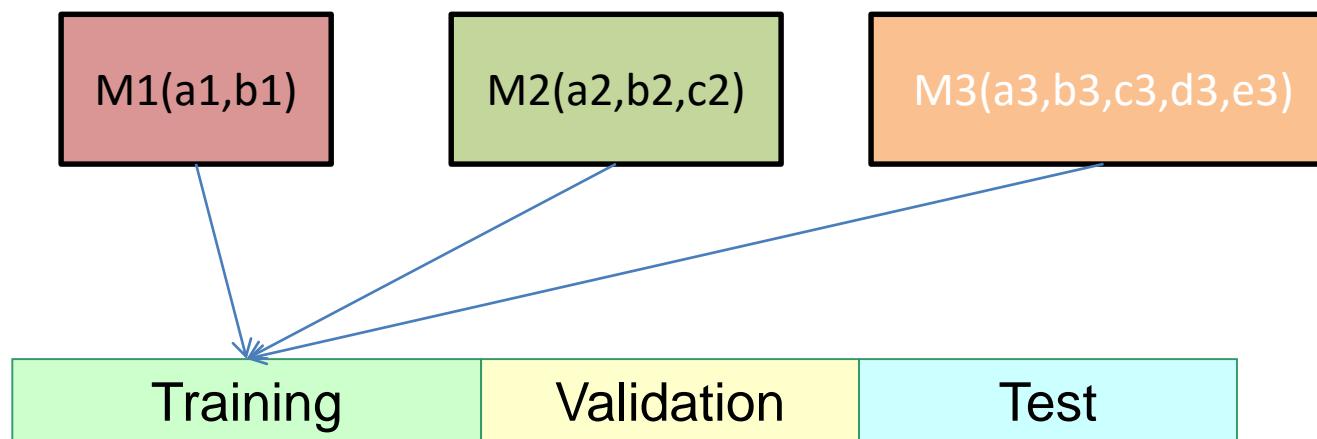
- Given data  $D = \{X_i, Y_i, i = 1 \dots n\}$



- When fitting data, Maximum Likelihood is usually used
- $\alpha_i$  can be different things:
  - Type of distribution
  - Number of variables in the model
  - Regularization parameter value
  - ...

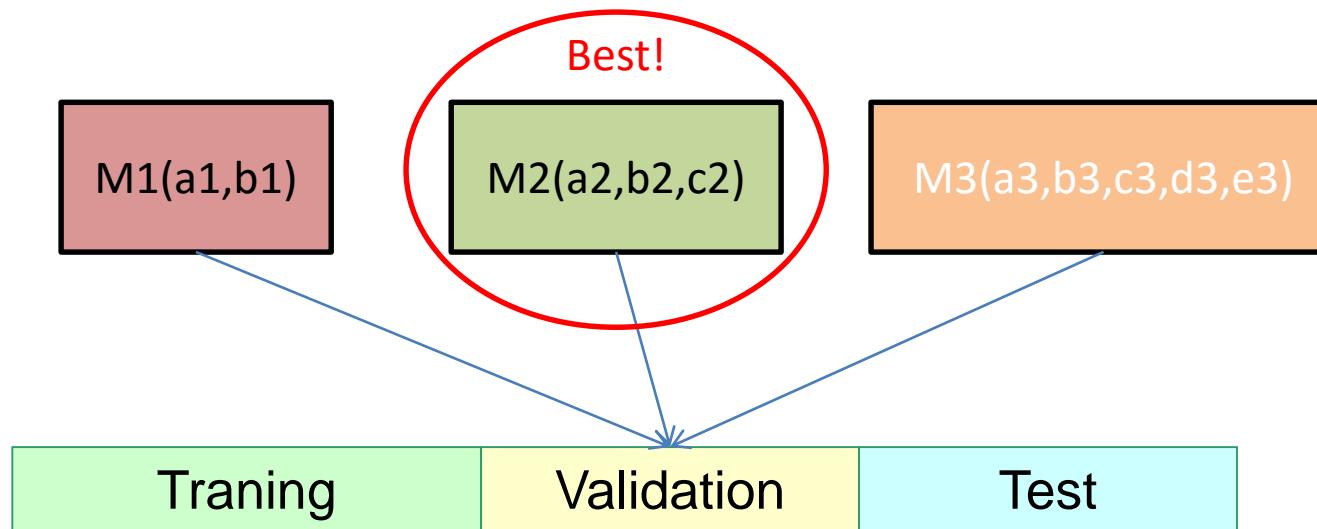
# Holdout method

- Training set is used for fitting models to the dataset by using maximum likelihood



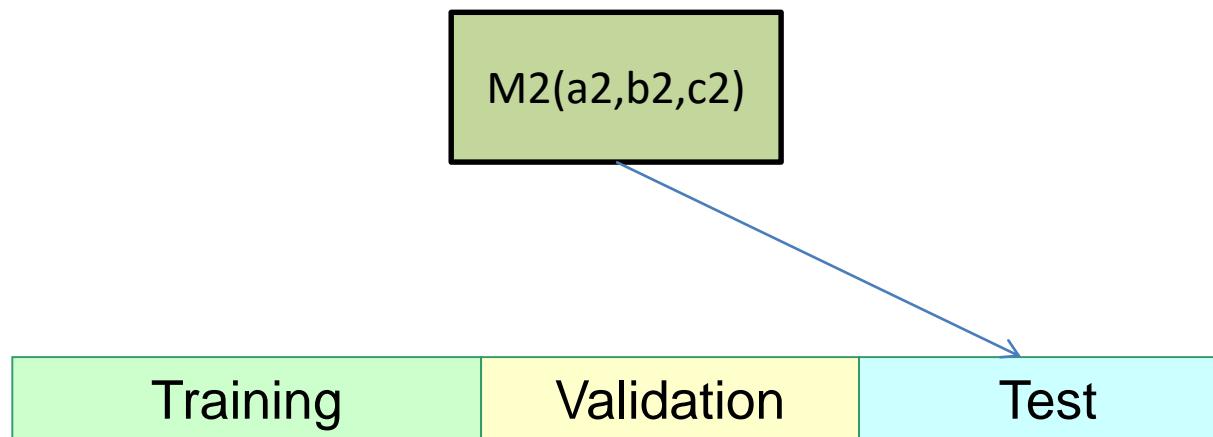
# Holdout method

- Validation set is used to choose the best model (lowest risk)

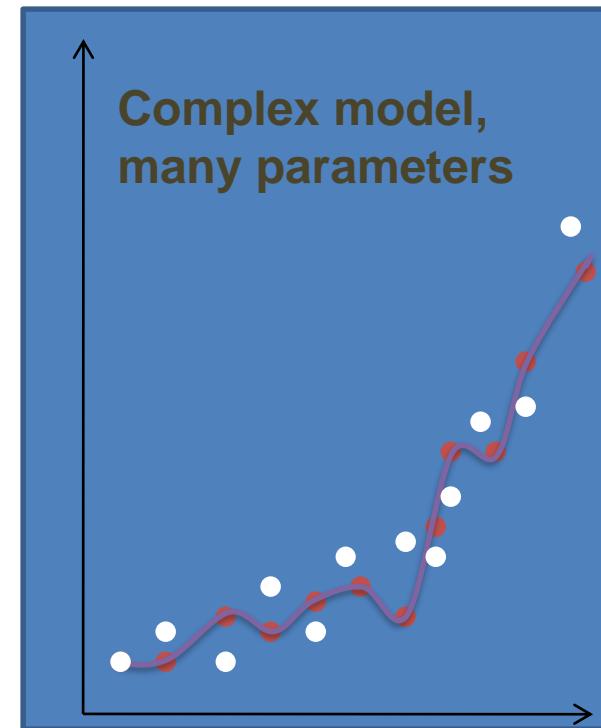
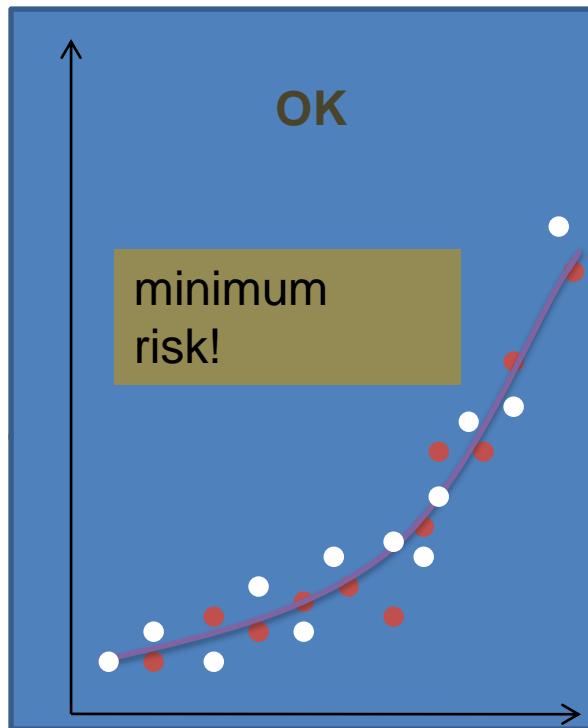
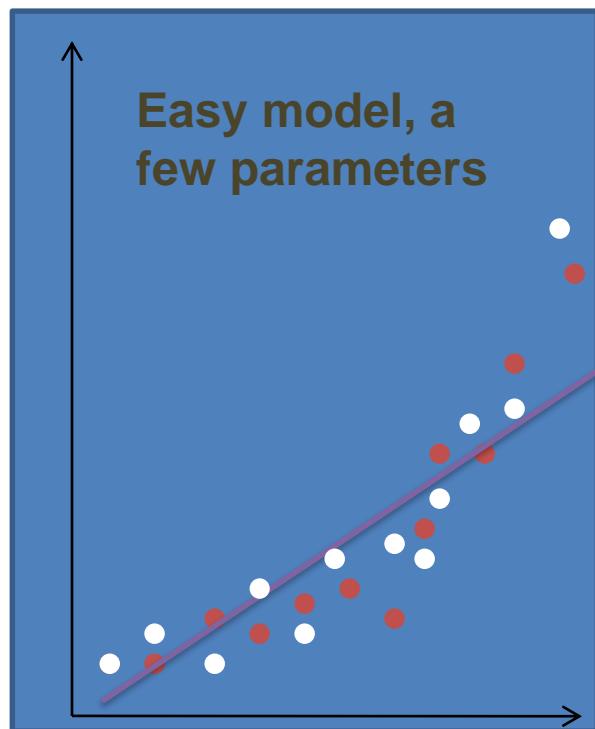


# Holdout method

- Test set is used to test a performance on a new data



# Holdout method



# Holdout in R

- How to partition into train/test?
  - Use `set.seed(12345)` in the labs to get identical results

```
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.7))
train=data[id,]
test=data[-id,]
```

- How to partition into train/valid/test?

```
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.4))
train=data[id,]

id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.3))
valid=data[id2,]

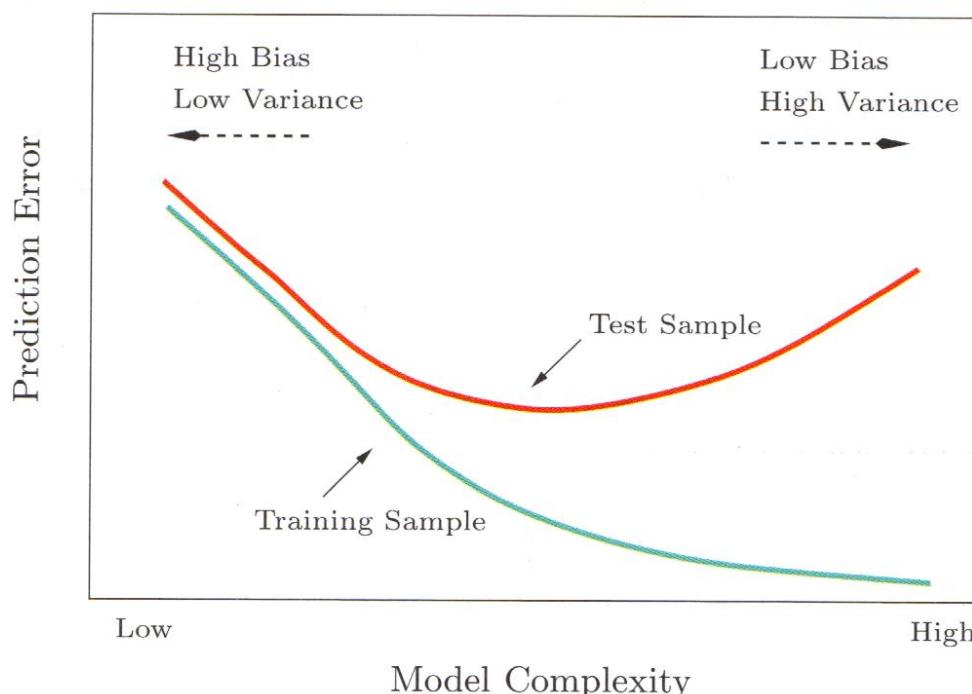
id3=setdiff(id1,id2)
test=data[id3,]
```

# Bias-variance tradeoff

- Bias of an estimator  $Bias(\hat{y}(x_0)) = E[\hat{y}(x_0)] - f(x_0)$ ,  $f(x_0)$  is expected response
  - If  $Bias(\hat{y}(x_0)) = 0$ , the estimator is **unbiased**
  - ML estimators are asymptotically unbiased if the model is enough complex
  - However, unbiasedness does not mean a good choice!

# Bias-variance tradeoff

- Assume loss is  $L(Y, \hat{y}) = (Y - \hat{y})^2$   
 $R(Y(x_0), \hat{y}(x_0)) = \sigma^2 + Bias^2(\hat{y}(x_0)) + Var(\hat{y}(x_0))$



When loss is not quadratic, no such nice formula exist

# Cross-validation

- Compared to holdout method:
  - Why do we use only some portion of data for training- can we use more (increase accuracy)?

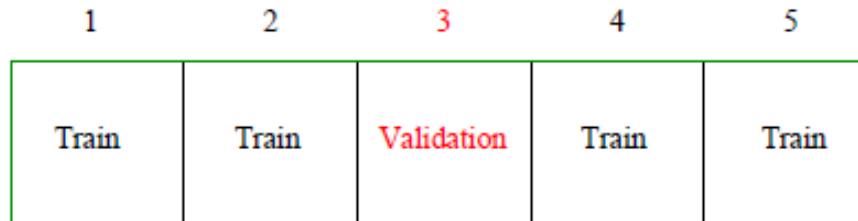
## **Cross-validation** (Estimates Err)

### **K-fold cross-validation (rough scheme, show picture):**

1. Permute the observations randomly
2. Divide data-set in K roughly equally-sized subsets
3. Remove subset #i and fit the model using remaining data.
4. Predict the function values for subset #i using the fitted model.
5. Repeat steps 3-4 for different i
6. CV= squared difference between observed values and predicted values (another function is possible)

# Cross-validation

## Cross-validation



Note: if  $K=N$  then method is *leave-one-out* cross-validation.

$$\kappa : \{1, \dots, N\} \mapsto \{1, \dots, K\}$$

**K-fold cross-validation:**  $CV =$

$$\frac{1}{N} \sum_{i=1}^N L(Y_i, \hat{y}^{-k(i)}(x_i))$$

What to do if  $N$  is not a multiple of  $K$ ?

# Cross-validation vs Holdout

- Holdout is easy to do (a few model fits to each data)
- Cross validation is computationally demanding (many model fits)
- Holdout is applicable for large data
  - Otherwise, model selection performs poorly
- Cross validation is more suitable for smaller data

# Analytical methods

- Analytical expressions to select models
  - $AIC$  (Akaike's information criterion)

**Idea:** Instead of  $R(Y, \hat{y}) = E[L(Y, \hat{y}(X, D))]$   
consider **in-sample** risk (only  $Y$  in  $D$  is random):

$$R_{in}(Y, \hat{y}) = \frac{1}{N} \sum_{i=1}^N E_{Y_i}[L(Y_i, \hat{y}(X, D)) \mid D, X \in D]$$

# Analytical methods

- One can show that

$$R_{in}(Y, \hat{y}) \approx R_{train} + \frac{2}{N} \sum_i cov(\hat{y}_i, Y_i)$$

where  $R_{train} = \sum_{X_i, Y_i \in T} L(Y_i, \hat{y}_i)$

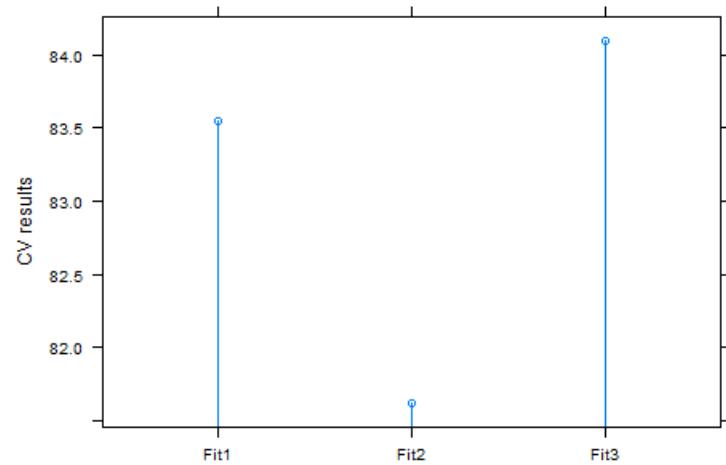
- Recall, **degrees of freedom**  $df(model) = \frac{1}{\sigma^2} \sum_i cov(\hat{y}_i, Y_i)$ 
  - When model is linear,  $df$  is the number of parameters.
- If loss is defined by minus two loglikelihood,  
$$AIC \equiv -2loglik(D) + 2df(model)$$

# Cross-validation

- Try models with different predictor sets

```
data=read.csv("machine.csv", header=F)
library(cvTools)

fit1=lm(V9~V3+V4+V5+V6+V7+V8, data=data)
fit2=lm(V9~V3+V4+V5+V6+V7, data=data)
fit3=lm(V9~V3+V4+V5+V6, data=data)
f1=cvFit(fit1, y=data$V9, data=data,K=10,
foldType="consecutive")
f2=cvFit(fit2, y=data$V9, data=data,K=10,
foldType="consecutive")
f3=cvFit(fit3, y=data$V9, data=data,K=10,
foldType="consecutive")
res=cvSelect(f1,f2,f3)
plot(res)
```



# 732A95 Introduction to Machine Learning

## Lecture 2a: Ensemble Methods

Jose M. Peña  
IDA, Linköping University, Sweden

## Contents

- ▶ Bagging
- ▶ Random Forests
- ▶ Boosting
  - ▶ AdaBoost
  - ▶ Forward Stage Additive Modeling
  - ▶ Gradient Boosting
- ▶ Summary

## Literature

- ▶ Main source
  - ▶ Hastie, T., Tibshirani, R. and Friedman, J. *The Elements of Statistical Learning*. Springer, 2009. Sections 10.1-10.10 and 15.1-15.4.
- ▶ Additional source
  - ▶ Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006. Sections 14.1-14.4.

## Bagging

- ▶ Boosting aggregation (bagging) is a technique to combine (weak) regressions and so produce a more accurate (committee) regression. It can also be applied to classification.
- ▶ Main steps:
  - ▶ Obtain  $B$  bootstrap samples of the original training data, i.e. draw  $B$  samples with replacement from the original data and of the same size as the original data.
  - ▶ Run the regression algorithm on each bootstrap sample  $b$  to obtain the regression  $f^b(x)$ .
  - ▶ Return the average of the regressions obtained, that is

$$f_{\text{bag}}(x) = \frac{1}{B} \sum_b f^b(x)$$

- ▶ Bagging (generalized) linear regressions is not particularly useful: The bagged regression will converge to the one learned from the original data as  $B$  increases.
- ▶ Then, it makes more sense to use bagging with non-linear regressions.
- ▶ For classification, we can average the individual models by averaging their posterior class probabilities, or use majority voting.

## Bagging

- Let  $h(x)$  denote the true regression. Then,  $f^b(x) = h(x) + \epsilon^b(x)$ .
- The error of  $f^b(x)$  can be expressed as

$$E_X[(f^b(x) - h(x))^2] = E_X[\epsilon^b(x)^2]$$

- The error of  $f_{bag}(x)$  can be expressed as

$$E_X\left[\left(\frac{1}{B} \sum_b f^b(x) - h(x)\right)^2\right] = E_X\left[\left(\frac{1}{B} \sum_b \epsilon^b(x)\right)^2\right]$$

- Assume** that the error terms  $\epsilon^b(x)$  have zero mean and are uncorrelated, i.e.  $E_X[\epsilon^b(x)] = 0$  and  $E_X[\epsilon^b(x)\epsilon^{b'}(x)] = 0$ . Then,

$$E_X\left[\left(\frac{1}{B} \sum_b f^b(x) - h(x)\right)^2\right] = \frac{1}{B} \left[ \frac{1}{B} \sum_b E_X[\epsilon^b(x)^2] \right]$$

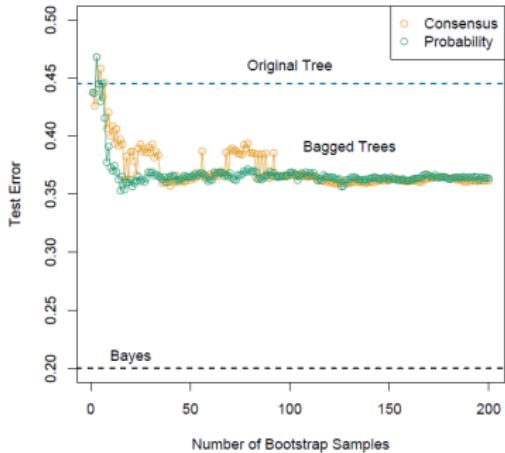
which implies that bagging reduces the average error of the individual regressions by a factor of  $B$ .

- In practice, the reduction in error is less dramatic as the individual errors are highly correlated.
- At least, the bagged error is never larger than the average individual errors:

$$\frac{1}{B} \sum_b E_X[\epsilon^b(x)^2] = E_X\left[\sum_b \frac{1}{B} \epsilon^b(x)^2\right] \geq E_X\left[\left(\frac{1}{B} \sum_b \epsilon^b(x)\right)^2\right]$$

by Jensen's inequality, i.e.  $\sum_i \lambda_i g(a_i) \geq g(\sum_i \lambda_i a_i)$ .

## Bagging



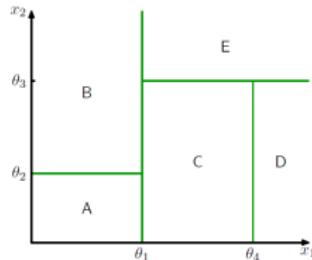
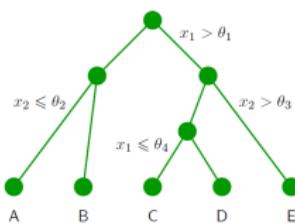
**FIGURE 8.10.** Error curves for the bagging example of Figure 8.9. Shown is the test error of the original tree and bagged trees as a function of the number of bootstrap samples. The orange points correspond to the consensus vote, while the green points average the probabilities.

- ▶ Note that bagging reduces the variance of the original algorithm (the more uncorrelated the individual errors, the larger the reduction).
- ▶ The reason is that the variance of the average of  $B$  identically distributed variables with positive pairwise correlation  $\rho$  is

$$\rho\sigma^2 + \frac{1 - \rho}{B}\sigma^2$$

## Random Forests

- Recall from previous lectures that a decision tree partitions the input space into regions. Each region has associated a predictor.
- Then, decision trees can be seen as a form of committee predictor.



- The model in each region is typically a constant, specifically
  - the result of majority voting for classification, since the best classifier under the 0-1 loss function is  $\arg \max_y p(y|x)$ , and
  - the average for regression, since the best regression function under the squared error loss function is  $E_Y[y|x]$ .
- Decision trees have many advantages (popular, fast, interpretable, etc.) and two major disadvantages: Low accuracy and high variance.
- These disadvantages make them good candidates for bagging, at the expense of compromising some of their advantages.

## Random Forests

- ▶ Random forest = bagging + decision trees + decorrelation.

---

**Algorithm 15.1** Random Forest for Regression or Classification.

---

1. For  $b = 1$  to  $B$ :
  - (a) Draw a bootstrap sample  $\mathbf{Z}^*$  of size  $N$  from the training data.
  - (b) Grow a random-forest tree  $T_b$  to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size  $n_{min}$  is reached.
    - i. Select  $m$  variables at random from the  $p$  variables.
    - ii. Pick the best variable/split-point among the  $m$ .
    - iii. Split the node into two daughter nodes.
2. Output the ensemble of trees  $\{T_b\}_1^B$ .

To make a prediction at a new point  $x$ :

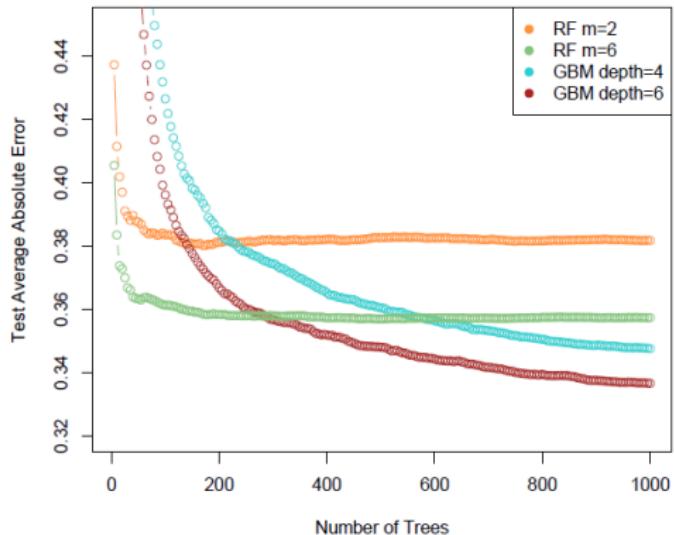
$$\text{Regression: } \hat{f}_{\text{rf}}^B(x) = \frac{1}{B} \sum_{b=1}^B T_b(x).$$

*Classification:* Let  $\hat{C}_b(x)$  be the class prediction of the  $b$ th random-forest tree. Then  $\hat{C}_{\text{rf}}^B(x) = \text{majority vote } \{\hat{C}_b(x)\}_1^B$ .

---

- ▶ Note that step 1ai aims to decorrelate the individual decision trees.

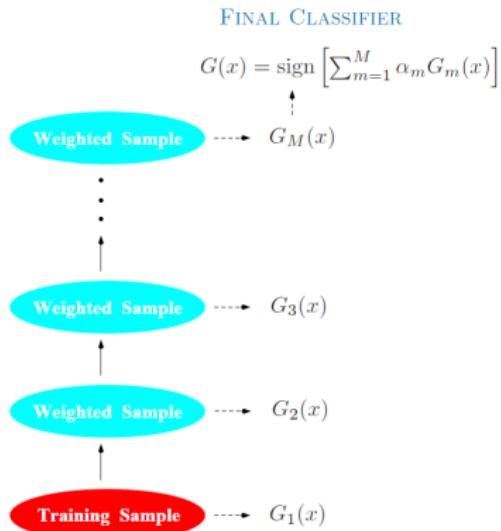
## Random Forests



**FIGURE 15.3.** Random forests compared to gradient boosting on the California housing data. The curves represent mean absolute error on the test data as a function of the number of trees in the models. Two random forests are shown, with  $m = 2$  and  $m = 6$ . The two gradient boosted models use a shrinkage parameter  $\nu = 0.05$  in (10.41), and have interaction depths of 4 and 6. The boosted models outperform random forests.

## Boosting

- ▶ Boosting is a technique to combine (weak) classifiers and so produce a more accurate (committee) classifier. It can also be applied to regression.
- ▶ Main steps:
  - ▶ Run the original classification algorithm on the original/modified training data.
  - ▶ Modify the training data by giving
    - ▶ more weight to the erroneously classified points, and
    - ▶ less weight to the correctly classified points.
  - ▶ Iterate through the previous steps a number of times.
  - ▶ Return a weighted average of the classifiers obtained.



## AdaBoost

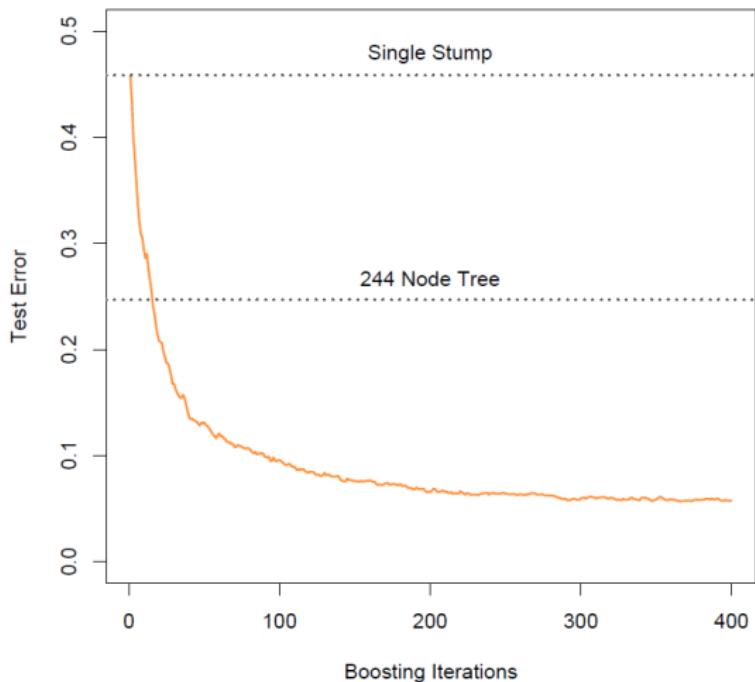
---

**Algorithm 10.1** *AdaBoost.M1.*

---

1. Initialize the observation weights  $w_i = 1/N$ ,  $i = 1, 2, \dots, N$ .
  2. For  $m = 1$  to  $M$ :
    - (a) Fit a classifier  $G_m(x)$  to the training data using weights  $w_i$ .
    - (b) Compute
$$\text{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}.$$
    - (c) Compute  $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$ .
    - (d) Set  $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$ ,  $i = 1, 2, \dots, N$ .
  3. Output  $G(x) = \text{sign} \left[ \sum_{m=1}^M \alpha_m G_m(x) \right]$ .
-

## AdaBoost



**FIGURE 10.2.** Simulated data (10.2): test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a 244-node classification tree.

## Forward Stage Additive Modeling

- ▶ Boosting can be seen as fitting an additive expansion of a set of basis functions. That is, the boosted classifier

$$G(x) = \sum_m \alpha_m G_m(x)$$

can be rewritten as

$$f(x) = \sum_m \beta_m b(x; \gamma_m)$$

and the aim of boosting can be rephrased as minimizing a loss function over the training data  $\{x_i, y_i\}$ , that is

$$\min_{\{\beta_m, \gamma_m\}} \sum_i L(y_i, \sum_m \beta_m b(x_i; \gamma_m))$$

- ▶ The problem above is challenging for most loss functions and basis functions. Heuristic solution: Add the basis functions one at a time.

## Forward Stage Additive Modeling

---

**Algorithm 10.2** *Forward Stagewise Additive Modeling.*

---

1. Initialize  $f_0(x) = 0$ .
2. For  $m = 1$  to  $M$ :
  - (a) Compute

$$(\beta_m, \gamma_m) = \arg \min_{\beta, \gamma} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$

- (b) Set  $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$ .
-

## AdaBoost

- It can be shown that AdaBoost is equivalent to forward staged additive modeling with exponential loss function, that is

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

- In each step of forward staged additive modeling, we have to solve

$$\begin{aligned}(\beta_m, \gamma_m) &= \arg \min_{\beta, \gamma} \sum_i \exp[-y_i(f_{m-1}(x_i) + \beta b(x_i; \gamma))] \\ &= \arg \min_{\beta, \gamma} \sum_i w_i^{(m)} \exp(-y_i \beta b(x_i; \gamma))\end{aligned}$$

where  $w_i^{(m)} = \exp(-y_i f_{m-1}(x_i))$  are the weights applied by AdaBoost to modify the training data.

- Note that

$$\sum_i w_i^{(m)} \exp(-y_i \beta b(x_i; \gamma)) = \exp(-\beta) \sum_{y_i=b(x_i; \gamma)} w_i^{(m)} + \exp(\beta) \sum_{y_i \neq b(x_i; \gamma)} w_i^{(m)}$$

and, thus, for any given  $\beta > 0$

$$\gamma_m = \arg \min_{\gamma} \sum_i w_i^{(m)} I(y_i \neq b(x_i; \gamma))$$

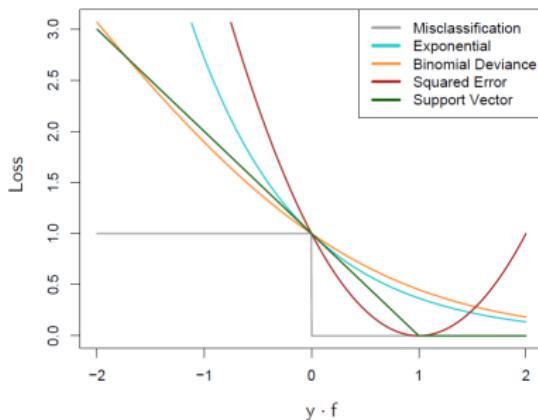
- This shows how to solve step 2a of AdaBoost: Simply, minimize the error rate on the modified **training data**.

## AdaBoost

- ▶ It can be shown that the population minimizer of the exponential loss function is

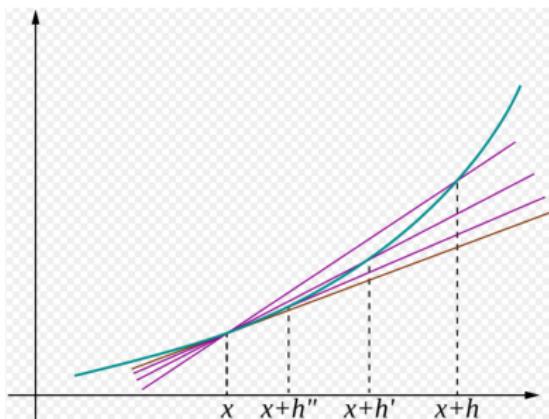
$$\arg \min_{f(x)} E_Y[\exp(-yf(x))] = \frac{1}{2} \log \frac{p(Y=1|x)}{p(Y=-1|x)}$$

- ▶ Since Adaboost's output is an approximation to the population minimizer, it makes sense to use the sign as the classification rule in step 3.
- ▶ Note that the exponential loss for a misclassified point increases exponentially with its margin  $yf(x)$ . This may make the performance of Adaboost to degrade in settings with wrongly labeled points: Adaboost may try to classify them correctly at the expense of other misclassified points that are correctly labeled.



## Gradient Boosting

- For some loss functions and/or basis functions, forward staged additive modeling can be cumbersome. In those cases, gradient boosting may be the solution.
- Recall that  $f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$



- Recall that the gradient is a vector whose components are the partial derivatives.

# Gradient Boosting

---

## Gradient Boosting

---

1. Initialize  $f_0(x) = \arg \min_{\gamma} \sum_i L(y_i, b(x_i; \gamma))$
2. For  $m = 1$  to  $M$ :
  - (a) Compute the gradient residual  $r_{im} = -\left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x)=f_{m-1}(x)}$
  - (b)  $\gamma_m = \arg \min_{\gamma} \sum_i (r_{im} - b(x_i; \gamma))^2$
  - (c)  $\beta_m = \arg \min_{\beta} \sum_i L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma_m))$
  - (d)  $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$

- 
- ▶ Note that step 2b aims for the basis function that is most parallel to the gradient.
  - ▶ Step 1a is easy in some cases, e.g. if  $L(y, f(x)) = \frac{1}{2}(y - f(x))^2$  then

$$\frac{\partial L(y, f(x))}{\partial f(x)} = y - f(x)$$

## Summary

- ▶ Bagging: Combines weak predictors to reduce error and variance.
- ▶ Random forest = bagging + decision trees + decorrelation.
- ▶ AdaBoost: Combination of weak predictors under exponential loss.
- ▶ Gradient boosting: Extension to arbitrary loss functions.
- ▶ How many individual models in bagging, boosting and random forest ?  
(Nested) cross-validation.

# 732A95 Introduction to Machine Learning

## Lecture 2b: Mixtures Models

Jose M. Peña  
IDA, Linköping University, Sweden

- ▶ Mixture Models
- ▶ Maximum Likelihood
- ▶ Expectation Maximization Algorithm
- ▶ Number of Mixture Components
- ▶ Model-Based Clustering
- ▶  $K$ -Means Algorithm
- ▶ Summary

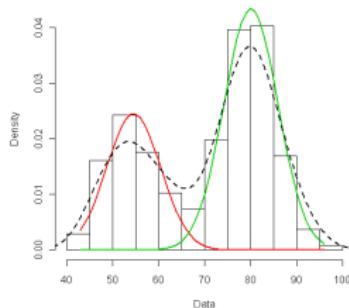
## Mixture Models

- Sometimes the data do not follow any known probability distribution but a mixture of known distributions such as

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

where  $p(\mathbf{x}|k)$  are called mixture components, and  $p(k)$  are called mixing coefficients, usually denoted by  $\pi_k$ .

- We can also see a mixture model as an ensemble model.



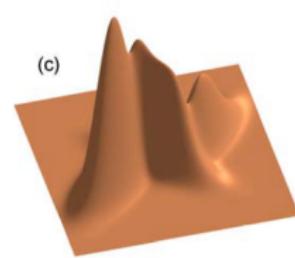
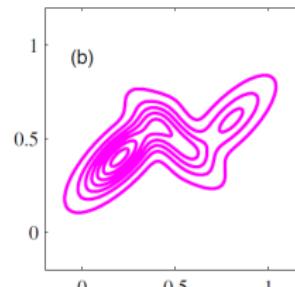
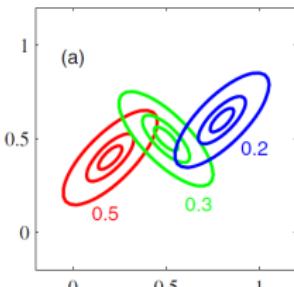
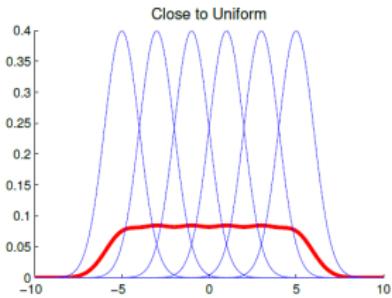
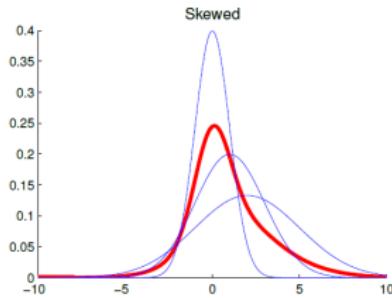
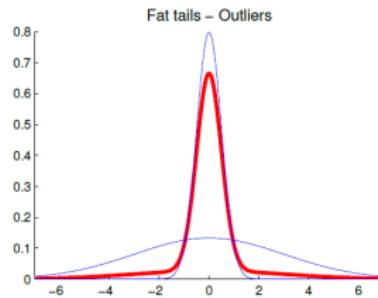
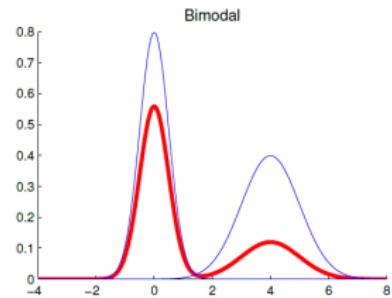
- Mixture of multivariate Gaussian distributions:

$$p(\mathbf{x}) = \sum_k \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \text{ and } \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{2\pi^{D/2}} \frac{1}{|\boldsymbol{\Sigma}_k|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}-\boldsymbol{\mu}_k)}$$

- Note that a mixture model defines a proper probability distribution:

$$0 \leq p(\mathbf{x}) \leq 1 \text{ and } \int p(\mathbf{x}) d\mathbf{x} = 1$$

# Mixture Models



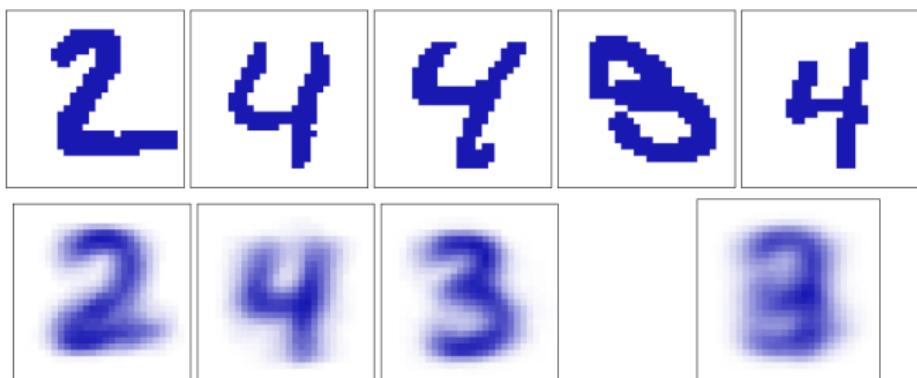
## Mixture Models

- ▶ Mixture of multivariate Bernoulli distributions:

$$p(\mathbf{x}) = \sum_k \pi_k \text{Bern}(\mathbf{x}|\boldsymbol{\mu}_k)$$

where

$$\text{Bern}(\mathbf{x}|\boldsymbol{\mu}_k) = \prod_i \mu_{ki}^{x_i} (1 - \mu_{ki})^{(1-x_i)}$$



**Figure 9.10** Illustration of the Bernoulli mixture model in which the top row shows examples from the digits data set after converting the pixel values from grey scale to binary using a threshold of 0.5. On the bottom row the first three images show the parameters  $\mu_{ki}$  for each of the three components in the mixture model. As a comparison, we also fit the same data set using a single multivariate Bernoulli distribution, again using maximum likelihood. This amounts to simply averaging the counts in each pixel and is shown by the right-most image on the bottom row.

## Maximum Likelihood

- Given a sample  $\{\mathbf{x}_n, k_n\}$  of size  $N$  from a mixture of multivariate Bernoulli distributions, rewrite it as  $\{\mathbf{x}_n, \mathbf{z}_n\}$  where  $\mathbf{z}_n$  is a  $K$ -dimensional binary vector having only the  $k_n$ -th element equal to 1.
- The log likelihood function is

$$\begin{aligned}\log p(\{\mathbf{x}_n, \mathbf{z}_n\} | \boldsymbol{\mu}, \boldsymbol{\pi}) &= \sum_n \log p(\mathbf{x}_n, \mathbf{z}_n | \boldsymbol{\mu}, \boldsymbol{\pi}) = \sum_n \log \prod_k [\pi_k \prod_i \mu_{ki}^{x_{ni}} (1 - \mu_{ki})^{(1-x_{ni})}]^{z_{nk}} \\ &= \sum_n \sum_k z_{nk} [\log \pi_k + \sum_i [x_{ni} \log \mu_{ki} + (1 - x_{ni}) \log (1 - \mu_{ki})]]\end{aligned}$$

- Let  $x'_{ni} = 1 - x_{ni}$  and  $\mu'_{ki} = 1 - \mu_{ki}$ . To maximize the log likelihood function subject to the constraints  $\sum_k \pi_k = 1$  and  $\mu_{ki} + \mu'_{ki} = 1$ , we maximize

$$\sum_n \sum_k z_{nk} [\log \pi_k + \sum_i [x_{ni} \log \mu_{ki} + x'_{ni} \log \mu'_{ki}]] + \lambda (\sum_k \pi_k - 1) + \lambda_i (\sum_i (\mu_{ki} + \mu'_{ki}) - 1)$$

where  $\lambda$  and  $\lambda_i$  are called Lagrange multipliers.<sup>1</sup>

- Setting to zero the derivatives with respect to  $\pi_k$ ,  $\mu_{ki}$  and  $\mu'_{ki}$  gives

$$\pi_k = - \sum_n z_{nk} / \lambda \text{ and } \mu_k = - \sum_n z_{nk} x_{ni} / \lambda_i \text{ and } \mu'_k = - \sum_n z_{nk} x'_{ni} / \lambda_i$$

- Replacing this into the constraint gives  $\lambda = -N$  and  $\lambda_i = -\sum_n z_{nk}$  and, thus,

$$\pi_k^{ML} = \frac{\sum_n z_{nk}}{N} \text{ and } \mu_{ki}^{ML} = \frac{\sum_n z_{nk} x_{ni}}{\sum_n z_{nk}}$$

<sup>1</sup>Any stationary point of the Lagrangian function is a stationary point of the original function subject to the constraints. Moreover, the log likelihood function is concave.

## Maximum Likelihood

- Given a sample  $\{\mathbf{x}_n\}$  of size  $N$  from a mixture of multivariate Bernoulli distributions, the expected log likelihood function is

$$\begin{aligned}\mathbb{E}_{\mathbf{z}}[\log p(\{\mathbf{x}_n, \mathbf{z}_n\} | \boldsymbol{\mu}, \boldsymbol{\pi})] &= \sum_n \sum_{\mathbf{z}_n} p(\mathbf{z}_n | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi}) \log p(\mathbf{x}_n, \mathbf{z}_n | \boldsymbol{\mu}, \boldsymbol{\pi}) \\ &= \sum_n \sum_{\mathbf{z}_n} p(\mathbf{z}_n | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi}) \sum_k z_{nk} [\log \pi_k + \sum_i [x_{ni} \log \mu_{ki} + (1 - x_{ni}) \log(1 - \mu_{ki})]] \\ &= \sum_n \sum_k p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi}) [\log \pi_k + \sum_i [x_{ni} \log \mu_{ki} + (1 - x_{ni}) \log(1 - \mu_{ki})]]\end{aligned}$$

- Following a reasoning analogous to the complete-data case, we obtain that

$$\begin{aligned}\pi_k^{ML} &= \frac{\sum_n p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}{N} \\ \mu_{ki}^{ML} &= \frac{\sum_n x_{ni} p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}{\sum_n p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}\end{aligned}$$

- This is not a closed form solution, but it suggests the following algorithm.

## Expectation Maximization Algorithm

## EM algorithm

Set  $\pi$  and  $\mu$  to some initial values

Repeat until  $\pi$  and  $\mu$  do not change

Compute  $p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})$  for all  $n$       /\* E step \*/

Set  $\pi_k$  to  $\pi_k^{ML}$ , and  $\mu_{ki}$  to  $\mu_{ki}^{ML}$  for all  $k$  and  $i$  /\* M step \*/

- Note that  $p(z_{nk} | x_n, \mu, \pi)$  has to be computed for all  $n$  in each iteration:

$$p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi}) = \frac{p(z_{nk}, \mathbf{x}_n | \boldsymbol{\mu}, \boldsymbol{\pi})}{\sum_k p(z_{nk}, \mathbf{x}_n | \boldsymbol{\mu}, \boldsymbol{\pi})} = \frac{\pi_k p(\mathbf{x} | \boldsymbol{\mu}_k)}{\sum_k \pi_k p(\mathbf{x} | \boldsymbol{\mu}_k)}$$

- ▶ The difficulty of maximizing the expected log likelihood function is not only that no closed form solution exists, but also that the landscape is highly multimodal, i.e. each completion of the data defines a unimodal function but their sum is typically multimodal.
  - ▶ As a result, the EM algorithm can get trapped in local maxima and it is very sensitive to initialization.
  - ▶ The EM algorithm can also be obtained by maximizing  $\log p(\{x_n\}|\mu, \pi)$ .
  - ▶ The EM algorithm is guaranteed to increase  $\log p(\{x_n\}|\mu, \pi)$  in each iteration until a local maximum is reached. So, the algorithm aims for the ML estimates.

## Expectation Maximization Algorithm

- ▶ We can derive the EM algorithm for mixtures of multivariate Gaussian distributions in much the same way. Simply,

$$\pi_k^{ML} = \frac{\sum_n p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}{N}$$

$$\boldsymbol{\mu}_k^{ML} = \frac{\sum_n \mathbf{x}_n p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}{\sum_n p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}$$

$$\boldsymbol{\Sigma}_k^{ML} = \frac{\sum_n (\mathbf{x}_n - \boldsymbol{\mu}_k^{ML})(\mathbf{x}_n - \boldsymbol{\mu}_k^{ML})^T p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}{\sum_n p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}$$

## Number of Mixture Components

- ▶ Too few/many components will result in underfitting/overfitting.
- ▶ We can perform a search over the number of components by scoring each number with, for instance, the Bayesian information criterion:

$$\log p(\{\mathbf{x}_n\} | \boldsymbol{\mu}^{ML}, \boldsymbol{\pi}^{ML}) - \frac{M}{2} \log N$$

where  $M$  is the number of free parameters in the mixture model. Note that the EM algorithm has to be run for each candidate number.

- ▶ Under some conditions, the score above can be seen as an approximation of the Bayesian score:

$$\log p(\{\mathbf{x}_n\}) = \int \int \log p(\{\mathbf{x}_n\} | \boldsymbol{\mu}, \boldsymbol{\pi}) p(\boldsymbol{\mu}, \boldsymbol{\pi}) d\boldsymbol{\mu} d\boldsymbol{\pi}$$

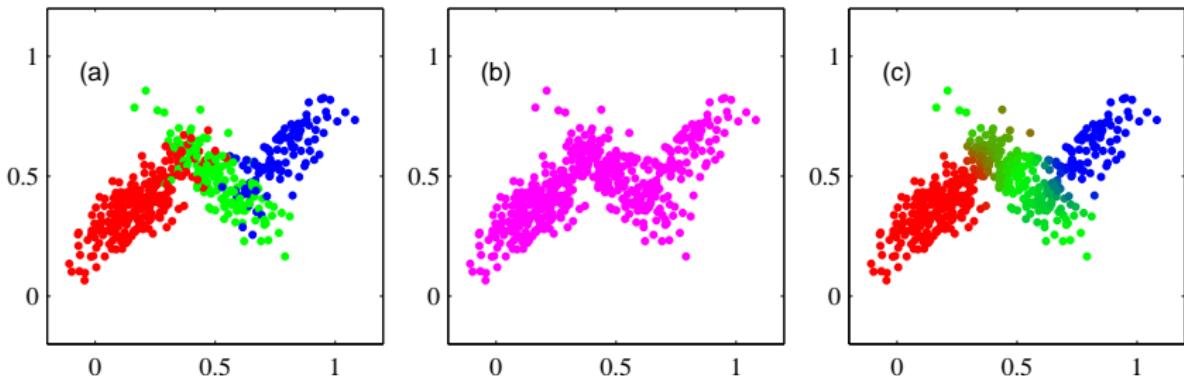
- ▶ There also exist algorithms that iteratively split and merge components according to the scores above until no further improvement occurs.
- ▶ Nested cross-validations is also an option.

## Model-Based Clustering

- ▶ A mixture model represents a mixture of populations, a.k.a. clusters:
  1. Choose a population according to  $\text{Mult}(k|\pi_1, \dots, \pi_K)$ .
  2. Sample an individual from the chosen population according to  $p(\mathbf{x}|\boldsymbol{\mu}_k)$ .
- ▶ Model-based clustering aims to soft-assign points to the different populations by computing

$$p(k|\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\pi}) = \frac{\pi_k p(\mathbf{x}|\boldsymbol{\mu}_k)}{\sum_k \pi_k p(\mathbf{x}|\boldsymbol{\mu}_k)}$$

- ▶ To do so, the number of populations and their parameters must be estimated: The EM algorithm and BIC score.



(a) Sample with cluster labels, (b) initial clustering, and (c) final clustering.

## K-Means Algorithm

- Recall that

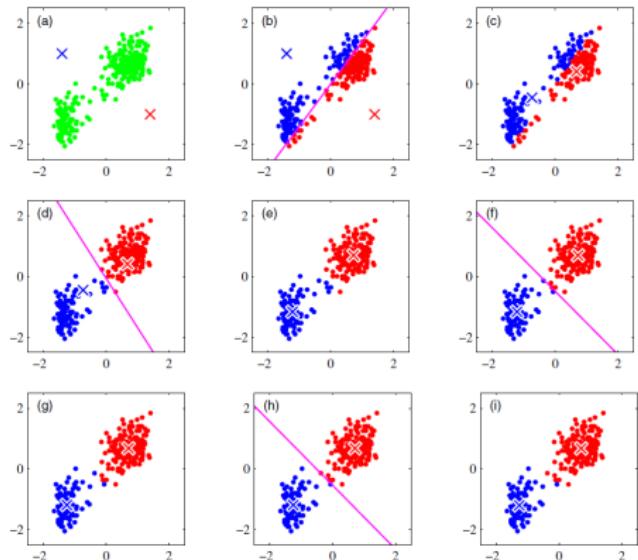
$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{2\pi^{D/2}} \frac{1}{|\boldsymbol{\Sigma}_k|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)\right)$$

- Assume that  $\boldsymbol{\Sigma}_k = \epsilon \mathbf{I}$  where  $\epsilon$  is a variance parameter and  $\mathbf{I}$  is the identity matrix. Then,

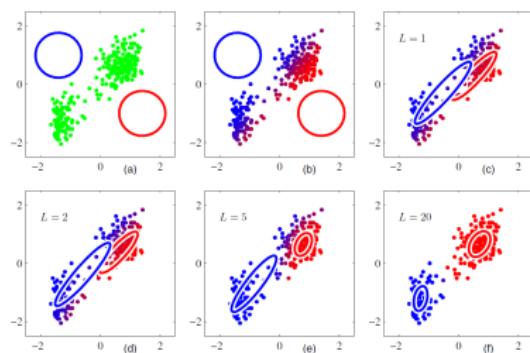
$$\begin{aligned}\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) &= \frac{1}{2\pi^{D/2}} \frac{1}{|\boldsymbol{\Sigma}_k|^{1/2}} \exp\left(-\frac{1}{2\epsilon} \|\mathbf{x} - \boldsymbol{\mu}_k\|^2\right) \\ p(k|\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\pi}) &= \frac{\pi_k \exp\left(-\frac{1}{2\epsilon} \|\mathbf{x} - \boldsymbol{\mu}_k\|^2\right)}{\sum_k \pi_k \exp\left(-\frac{1}{2\epsilon} \|\mathbf{x} - \boldsymbol{\mu}_k\|^2\right)}\end{aligned}$$

- As  $\epsilon \rightarrow 0$ , the smaller  $\|\mathbf{x} - \boldsymbol{\mu}_k\|^2$  the slower  $\exp\left(-\frac{1}{2\epsilon} \|\mathbf{x} - \boldsymbol{\mu}_k\|^2\right)$  goes to 0.
- As  $\epsilon \rightarrow 0$ , individuals are hard-assigned (i.e. with probability 1) to the population with closest mean. This clustering technique is known as *K-means* algorithm.
- Note that  $\boldsymbol{\pi}$  and  $\boldsymbol{\Sigma}$  play no role in the *K-means* algorithm whereas, in each iteration,  $\boldsymbol{\mu}_k$  is updated to the average of the individuals assigned to population  $k$ .

# K-Means Algorithm



K-means algorithm



EM algorithm

## Summary

- ▶ Mixture models: To model complex distributions by linearly combining simple distributions.
- ▶ EM algorithm: To estimate the ML parameters of mixture models. It converges to a local maximum of the log likelihood of the observed data.
- ▶ We can see mixture models as model-based clustering, and the  $K$ -means algorithm as a limit case thereof.

# Linear classification methods

## Lecture 3

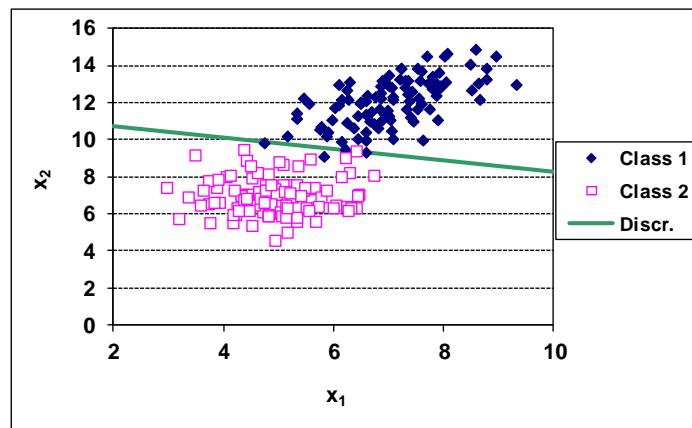
- Discriminant Analysis models
- Logistic regression
- Generalized Linear Model

# Classification

- Given data  $D = \{(X_i, Y_i), i = 1 \dots N\}$ 
  - $Y_i = Y(X_i) = C_j \in \mathcal{C}$
  - Class set  $\mathcal{C} = (C_1, \dots, C_K)$

## Classification problem:

- Decide  $\hat{Y}(x)$  that maps **any**  $x$  into some class  $C_K$ 
  - Decision boundary



# Classifiers

- **Deterministic:** decide a rule that directly maps  $X$  into  $\hat{Y}$
- **Probabilistic:** define a model for  $P(Y = C_i | X), i = 1 \dots K$

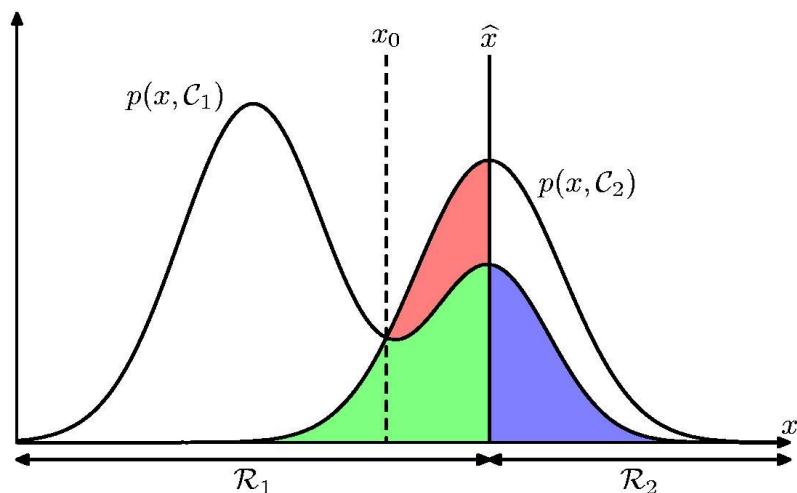
## Disadvantages of deterministic classifiers:

- Sometimes simple mapping is not enough (risk of cancer)
- Difficult to embed loss-> rerun of optimizer is often needed
- Combining several classifiers into one is more problematic
  - Algorithm A classifies as spam, Algorithm B classifies as not spam → ???
  - $P(\text{Spam} | A) = 0.99, P(\text{Spam} | B) = 0.45 \rightarrow$  better decision can be made

# Probabilities into decision

- Loss minimization

$$\min_{\hat{f}} EL(y, \hat{f}) = \min_{\hat{f}} \int L(y, \hat{f}) p(y, x|w) dx dy$$



When loss is  
 $\begin{cases} 1, \text{wrongly classified} \\ 0, \text{correctly classified} \end{cases}$

Classify  $Y$  as

$$\hat{Y} = \arg \max_c p(Y = c | X)$$

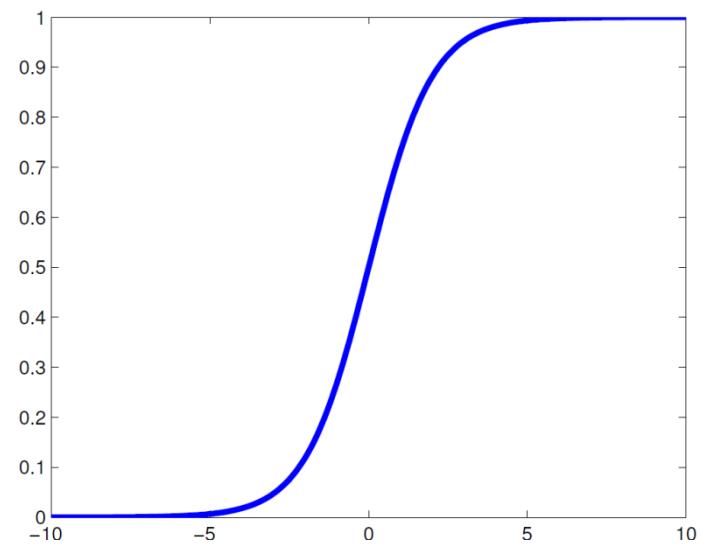
# Logistic regression

- Discriminative model
- Model for binary output
  - $C = \{C_1 = 1, C_2 = 0\}$   
 $p(Y = C_1|X) = \text{sigm}(\mathbf{w}^T \mathbf{x})$

What is  $P(Y = C_2|X)$ ?

$$\text{sigm}(a) = \frac{1}{1 + e^{-a}}$$

- Alternatively
- $$Y \sim \text{Bernoulli}(\text{sigm}(a)), a = \mathbf{w}^T \mathbf{x}$$
- $$\text{sigm}(a) = \frac{1}{1 + e^{-a}}$$



# Logistic regression

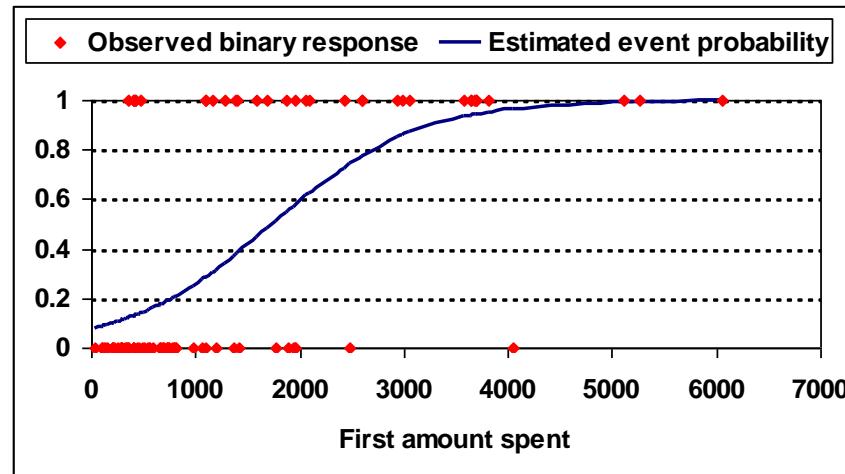
- Logistic model- yet another form

$$\ln \frac{p(Y = 1|X = x)}{P(Y = 0|X = x)} = \ln \frac{p(Y = 1|X = x)}{1 - P(Y = 1|X = x)} = \text{logit}(p(Y = 1|X = x)) = \mathbf{w}^T \mathbf{x}$$

**The log of the odds  
is linear in  $x$**

- Here  $\text{logit}(t) = \ln \left( \frac{t}{1-t} \right)$
- Note  $p(Y|X)$  is connected to  $\mathbf{w}^T \mathbf{x}$  via logit link

**Example:** Probability to buy more than once as function of First Amount Spend



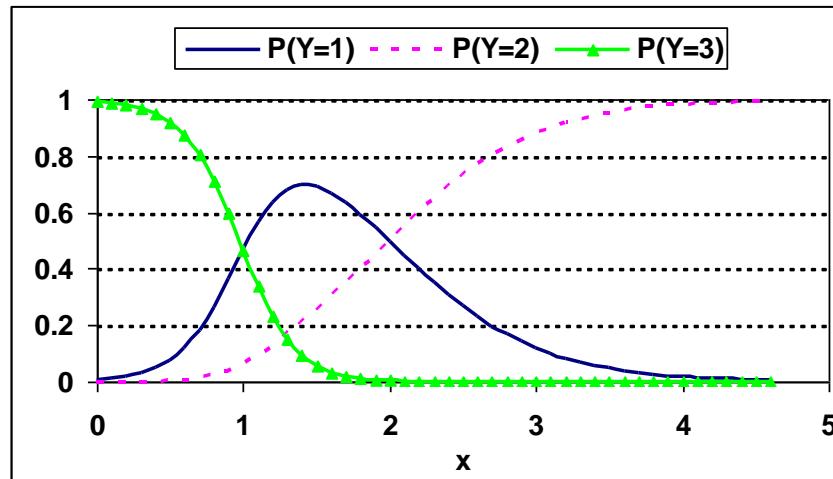
# Logistic regression

- When  $Y$  is categorical,

$$p(Y = C_i|x) = \frac{e^{\mathbf{w}_i^T x}}{\sum_{j=1}^K e^{\mathbf{w}_j^T x}} = \text{softmax}(\mathbf{w}_i^T x)$$

- Alternatively

$$Y \sim \text{Multinoulli} \left( \text{softmax}(\mathbf{w}_1^T x), \dots, \text{softmax}(\mathbf{w}_K^T x) \right)$$



# Logistic regression

## Fitting logistic regression

- In binary case,

$$\log P(D|w) = \sum_{i=1}^N y_i \log(\text{sigm}(w^T x_i)) + (1 - y_i) \log(1 - \text{sigm}(w^T x_i))$$

- Can not be maximized analytically, but unique maximizer exists
- To maximize loglikelihood, optimization used
  - Newton's method traditionally used (Iterative Reweighted Least Squares)
  - Steepest descent, Quasi-newton methods...

## Estimation:

For new  $x$ , estimate  $p(y) = [p_1, \dots p_C]$  and classify as  $\arg \max_i p_i$

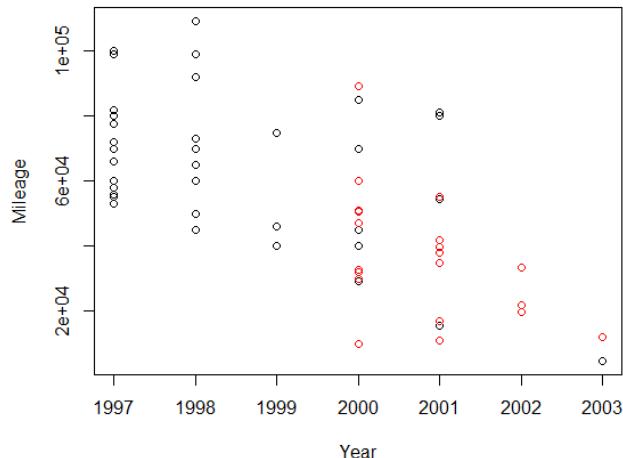
Decision boundaries of logistic regression are linear

# Logistic regression

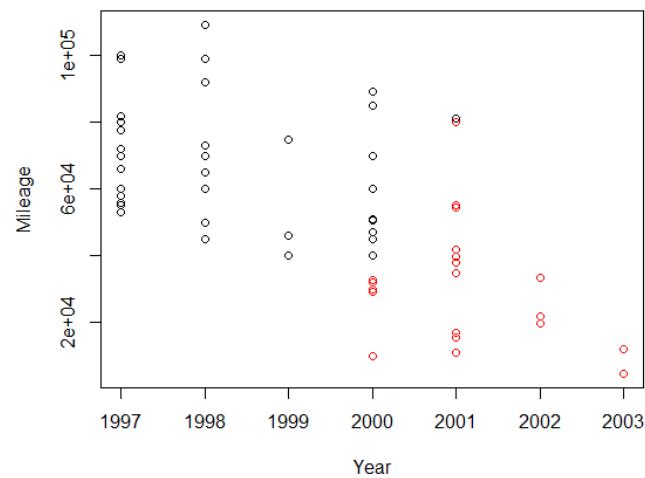
- In R, use `glm()` with `family="binomial"`
  - Predicted probabilities: `predict(fit,newdata, type="response")`

**Example** Equipment=f(Year, mileage)

**Original data**



**Classified data**



# Moving beyond typical distributions

- We know how to model
  - Normally distributed responses -> linear regression
  - Bernoulli and Multinomial responses → logistic regression
  - What if response distribution is more complex?

## Example 1: Daily Stock prices NASDAQ

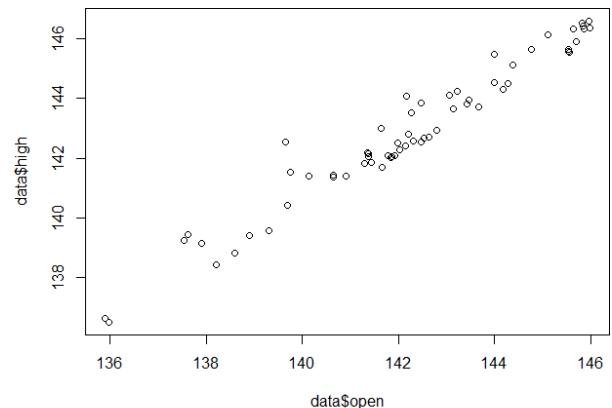
- Open
- High (within day)

Does it seem that the error is normal here?

## Example 2: Number of calls to bank

- Y=Number of calls
- X= time

Endless amount of classes → multinomial does not work... (Poisson)



# Exponential family

- More advanced error distributions are sometimes needed!
- Many distributions belong to **exponential** family:
  - Normal, Exponential, Gamma, Beta, Chi-squared..
  - Bernoulli, Multinoulli, Poisson...

$$p(\mathbf{x}|\boldsymbol{\eta}) = h(\mathbf{x})g(\boldsymbol{\eta})e^{(\boldsymbol{\eta}^T \mathbf{u}(\mathbf{x}))}$$

- Easy to find MLE and MAP
- Non-exponential family distributions: uniform, Student t

**Example:** Bernoulli

# Generalized linear models

- Assume  $Y$  from the exponential family
- **Model** is  $Y \sim EF(\mu, \dots)$ ,  $f(\mu) = \mathbf{w}^T \mathbf{x}$ 
  - Alt  $\mu = f^{-1}(\mathbf{w}^T \mathbf{x})$
  - $f^{-1}$  is activation function
  - $f$  is link function (in principle, arbitrary)
- Arbitrary  $f$  will lead to ( $s$  – dispersion parameter)

$$p(y|w, s) = h(y, s)g(\mathbf{w}, \mathbf{x})e^{\frac{b(\mathbf{w}, \mathbf{x})y}{s}}$$

- If  $f$  is a canonical link, then

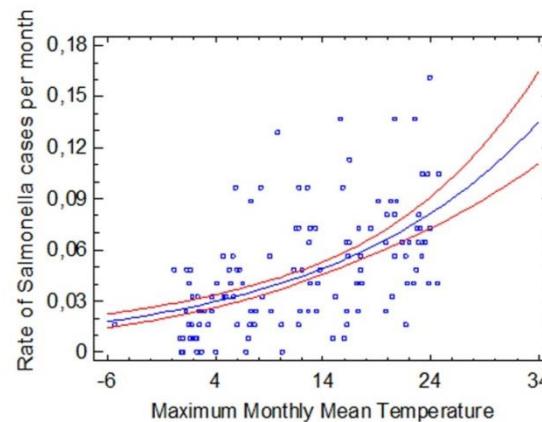
$$p(y|w, s) = h(y, s)g(\mathbf{w}, \mathbf{x})e^{\frac{(\mathbf{w}^T \mathbf{x})y}{s}}$$

# Generalized linear models

- Canonical links are normally used
  - MLE computations simplify
  - MLE  $\hat{w} = F(X^T Y)$  → computations do not depend on all data but rather a summary (sufficient statistics) → computations speed up

**Example:** Poisson regression

$$f^{-1}(\mu) = e^\mu, Y \sim \text{Poisson}(e^{w^T x})$$



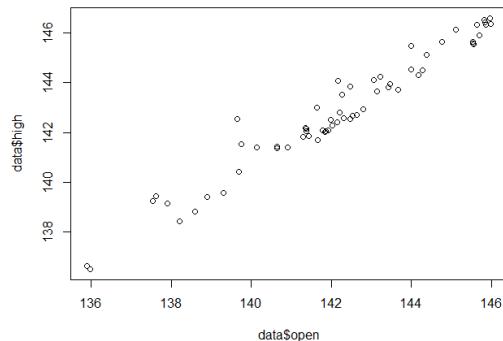
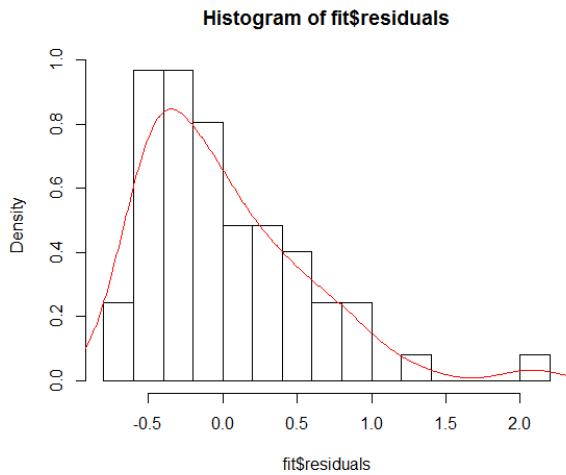
# Generalized linear model: software

- Use `glm(formula, family, data)` in R

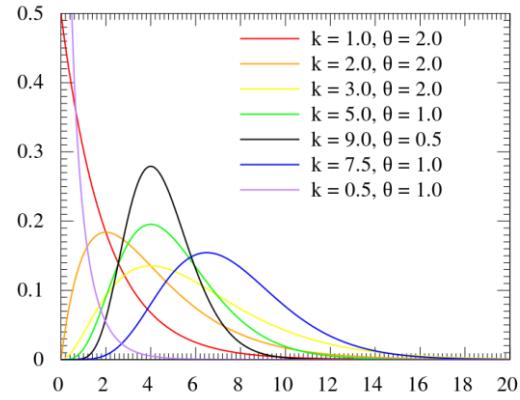
**Example:** Daily Stock prices NASDAQ

- Open
- High (within day)

1. Try to fit usual linear regression, study histogram of residuals



Gamma distribution: Wikipedia



# Generalized linear model

Assume

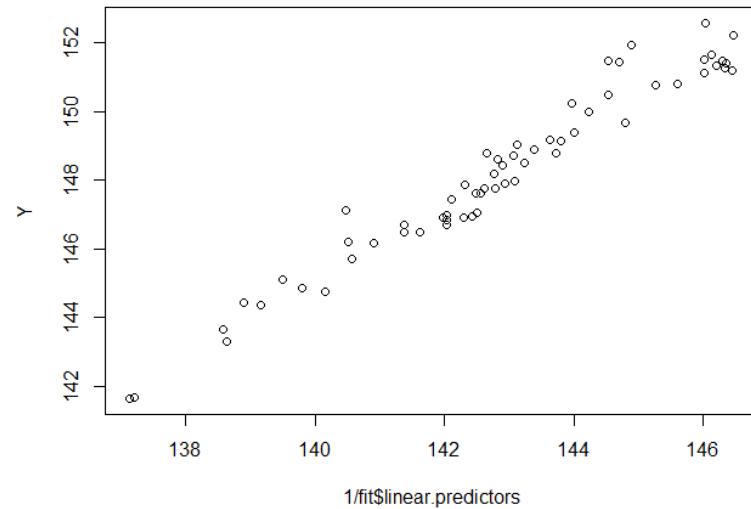
$$High \sim Gamma(1, \frac{1}{w_0 + w_1 Open})$$

What is link function here?

```
call:  
glm(formula = high ~ open, family = Gamma(link = "inverse"),  
    data = data)  
  
Deviance Residuals:  
    Min      1Q      Median      3Q      Max  
-0.0052879 -0.0028896 -0.0006678  0.0016598  0.0148083  
  
Coefficients:  
            Estimate Std. Error t value Pr(>|t|)  
(Intercept) 1.355e-02 1.962e-04 69.06 <2e-16 ***  
open        -4.604e-05 1.379e-06 -33.39 <2e-16 ***  
---  
signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

New generated data

- Has similar pattern as original data!

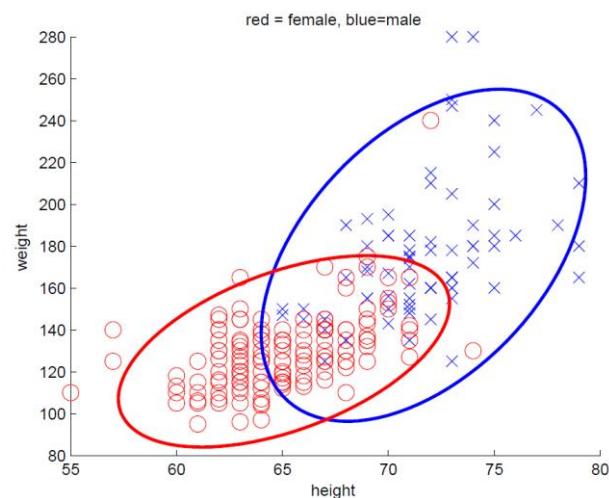
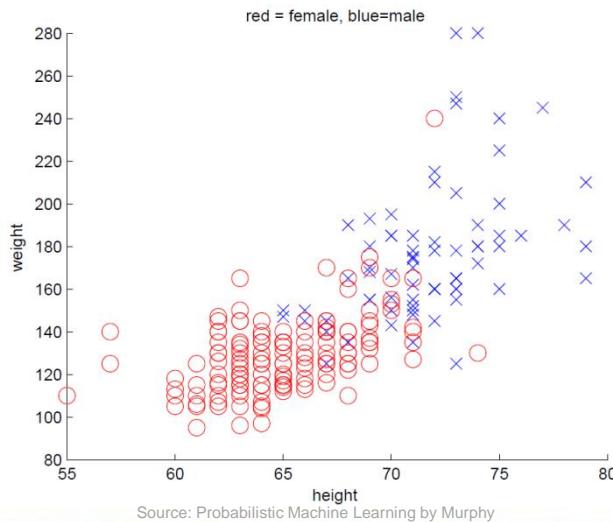


# Quadratic discriminant analysis

- Generative classifier
- Main assumptions:
  - $x$  is now **random** as well as  $y$

$$p(x|y = C_i, \theta) = N(x|\mu_i, \Sigma_i)$$

Unknown parameters  $\theta = \{\mu_i, \Sigma_i\}$



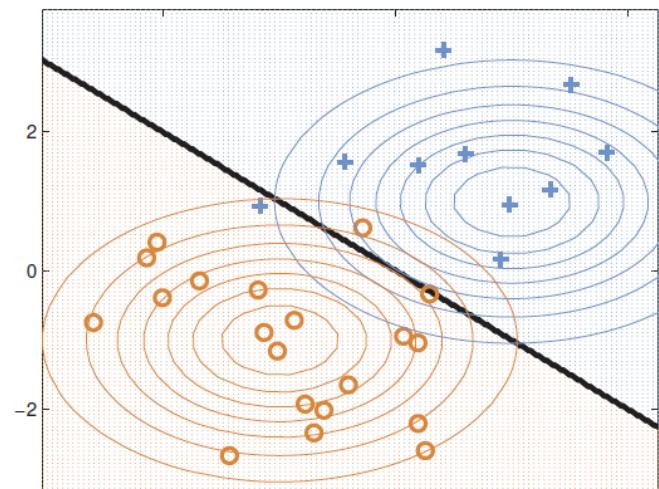
# Linear discriminant analysis (LDA)

- Assumption  $\Sigma_i = \Sigma, i = 1, \dots, K$
- Then  $p(y = c_i|x) = \text{softmax}(w_i^T x + w_{0i})$   
→ exactly the same form as the logistic regression

- $w_{0i} = -\frac{1}{2}\boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i + \log \pi_i$
- $w_i = \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i$

- Decision boundaries are linear
  - Discriminant function:

$$\delta_k(x) = x^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_k - \frac{1}{2} \boldsymbol{\mu}_k^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_k + \log \pi_k$$



Source: Probabilistic Machine Learning by Murphy

# Linear discriminant analysis (LDA)

- Difference LDA vs logistic regression??
  - Coefficients will be estimated differently! (models are different)
- How to estimate coefficients
  - find MLE.

$$\hat{\mu}_c = \frac{1}{N_c} \sum_{i:y_i=c} \mathbf{x}_i, \quad \hat{\Sigma}_c = \frac{1}{N_c} \sum_{i:y_i=c} (\mathbf{x}_i - \hat{\mu}_c)(\mathbf{x}_i - \hat{\mu}_c)^T$$

$$\hat{\Sigma} = \frac{1}{N} \sum_{c=1}^k N_c \hat{\Sigma}_c$$

- Sample mean and sample covariance are MLE!
- If class priors are parameters (**proportional priors**),

$$\hat{\pi}_c = \frac{N_c}{N}$$

# LDA and QDA: code

- Syntax in R, library **MASS**

`lda(formula, data, ..., subset, na.action)`

- Prior – class probabilities
- Subset – indices, if training data should be used

`qda(formula, data, ..., subset, na.action)`

`predict(..)`

# LDA: output

```
resLDA=lda(Equipment~Mileage+Year, data=mydata)
print(resLDA)
```

```
> print(resLDA)
call:
lda(Equipment ~ Mileage + Year, data = mydata)

Prior probabilities of groups:
 0      1 
0.6440678 0.3559322

Group means:
  Mileage   Year
0 63539.21 1998.447
1 36857.62 2000.762

coefficients of linear discriminants:
            LD1
Mileage -1.500069e-05
Year     5.745893e-01
```

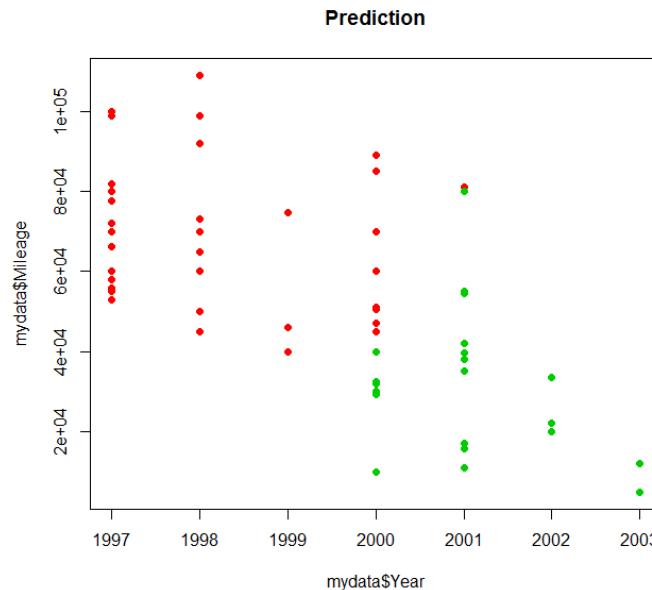
# LDA: output

- Misclassified items

```
plot(mydata$Year, mydata$Mileage,  
col=as.double(Pred$class)+1, pch=21,  
bg=as.double(Pred$class)+1,  
main="Prediction")
```

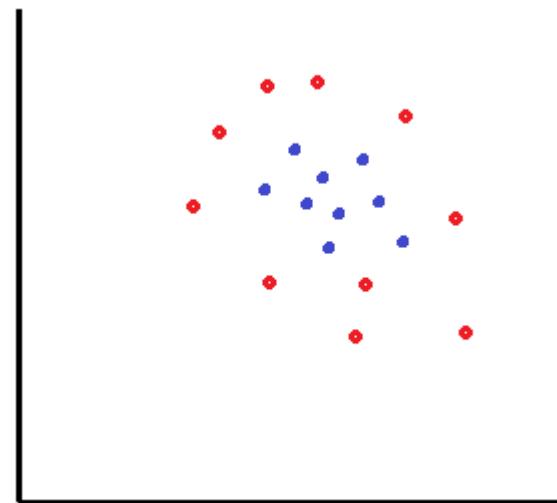
```
> table(Pred$class, mydata$Equipment)
```

	0	1
0	31	6
1	7	15



# LDA versus Logistic regression

- Generative classifiers are easier to fit, discriminative involve numeric optimization
- LDA and Logistic have same model form but are fit differently
- LDA has stronger assumptions than Logistic, some other generative classifiers lead also to Logistic expression
- New class in the data?
  - Logistic: fit model again
  - LDA: estimate new parameters from the new data
- Logistic and LDA: complex data fits badly unless interactions are included



# LDA versus Logistic regression

- LDA (and other generative classifiers) handle missing data easier
- Standardization and generated inputs:
  - Not a problem for Logistic
  - May affect the performance of the LDA in a complex way
- Outliers affect  $\Sigma \rightarrow$  LDA is not robust to gross outliers
- LDA is often a good classification method even if the assumption of normality and common covariance matrix are not satisfied.

# 732A95 Introduction to Machine Learning

## Lecture 3a: Online Learning

Jose M. Peña  
IDA, Linköping University, Sweden

- ▶ Online Learning
- ▶ Online Kernel Methods
- ▶ Online EM Algorithm
- ▶ Cross-Validation
- ▶ Summary

## Online Learning

- ▶ Offline learning: Batch of data used to optimize a function such as

$$\frac{1}{N} \sum_n L(t_n, y(\mathbf{x}_n, \boldsymbol{\theta}))$$

where  $L(\cdot)$  is log loss, squared error, 0/1 loss, etc.

- ▶ Online learning: Streaming data so we cannot wait until the end to start processing the data. Instead, we update our estimate with each new point's arrival. It may even be an interesting option if the batch of data does not fit in main memory.

---

### Online learning

---

for  $n = 1, 2, \dots$

Receive question  $\mathbf{x}_n$

Predict  $y(\mathbf{x}_n, \boldsymbol{\theta}_n)$

Receive true answer  $t_n$

Suffer loss  $L(t_n, y(\mathbf{x}_n, \boldsymbol{\theta}_n))$

---

- ▶ Examples: Weather prediction, spam filtering.
- ▶ Goal: Minimize the cumulative loss suffered along the run.

## Online Learning

- ▶ Assume that  $|\Theta| < \infty$  and  $t_n = y(\mathbf{x}_n, \boldsymbol{\theta}^*)$  for all  $n$  with  $\boldsymbol{\theta}^* \in \Theta$ .

---

Consistent

---

$$\Theta_1 = \Theta$$

for  $n = 1, 2, \dots$

Receive question  $\mathbf{x}_n$

Choose any  $\boldsymbol{\theta}_n \in \Theta_n$

Predict  $y(\mathbf{x}_n, \boldsymbol{\theta}_n)$

Receive true answer  $t_n$

Update  $\Theta_{n+1} = \{\boldsymbol{\theta} \in \Theta_n : y(\mathbf{x}_n, \boldsymbol{\theta}) = t_n\}$

---

- ▶ The algorithm above makes at most  $|\Theta| - 1$  errors.

---

Halving

---

$$\Theta_1 = \Theta$$

for  $n = 1, 2, \dots$

Receive question  $\mathbf{x}_n$

Predict  $\arg \max_{r \in \{0,1\}} |\{\boldsymbol{\theta} \in \Theta_n : y(\mathbf{x}_n, \boldsymbol{\theta}) = r\}|$

Receive true answer  $t_n$

Update  $\Theta_{n+1} = \{\boldsymbol{\theta} \in \Theta_n : y(\mathbf{x}_n, \boldsymbol{\theta}) = t_n\}$

---

- ▶ The algorithm above makes at most  $\log_2 |\Theta|$  errors.

- If we drop the assumption that  $t_n = y(\mathbf{x}_n, \boldsymbol{\theta}^*)$  for all  $n$  with  $\boldsymbol{\theta}^* \in \Theta$ , then our goal may be restated as minimizing the regret

$$\frac{1}{N} \sum_n L(t_n, y(\mathbf{x}_n, \boldsymbol{\theta}_n)) - \min_{\boldsymbol{\theta}^* \in \Theta} \frac{1}{N} \sum_n L(t_n, y(\mathbf{x}_n, \boldsymbol{\theta}^*))$$

- Note that the second term above is the loss of the batch solution.
- We may also be satisfied if we find a predictor whose regret grows sub-linearly with respect to  $N$ .
- These goals seem hopeless anyway, as the adversary may wait for our prediction and then answer the opposite label ( $\text{regret} = N - N/2$  at least, e.g. if the batch solution is majority voting).
- Solution: Output a posterior class distribution or equivalent. Although the adversary knows that we will select a label according to this distribution, his/her power will be limited.

---

Weighted majority (input:  $\eta \in (0, 1)$ )

---

$\mathbf{w}_1 = (1/M, \dots, 1/M)$  where  $M = |\Theta|$

for  $n = 1, 2, \dots$

Choose  $m \sim \mathbf{w}_n$  and predict  $y(\mathbf{x}_n, \boldsymbol{\theta}_m)$

Receive costs of all models  $\mathbf{c}_n \in [0, 1]^M$

Update  $\mathbf{w}_{n+1} = \frac{\mathbf{w}_n \exp(-\eta \mathbf{c}_n)}{\sum_j \mathbf{w}_{nj} \exp(-\eta \mathbf{c}_{nj})}$  for all  $i$

---

- The name of the algorithm is due to the fact that it predicts the label 1 with probability

$$p_n = \sum_m w_{nm} y(\mathbf{x}_n, \boldsymbol{\theta}_m)$$

- The loss can be written as

$$|p_n - t_n| = \left| \sum_m w_{nm} y(\mathbf{x}_n, \boldsymbol{\theta}_m) - t_n \right| = \sum_m w_{nm} |y(\mathbf{x}_n, \boldsymbol{\theta}_m) - t_n|$$

- Moreover, if  $\eta = \sqrt{N \log M}$  then

$$\sum_n |p_n - t_n| \leq \min_{1 \leq m \leq M} \sum_n c_{nm} + 2\sqrt{N \log M}$$

which implies a sub-linear regret wrt  $N$ , i.e.  $\frac{\text{regret}}{N} \rightarrow 0$  as  $N \rightarrow \infty$ .

- Moreover, if  $\eta = 1/2$  then

$$\sum_n |p_n - t_n| \leq 2 \min_{1 \leq m \leq M} \sum_n c_{nm} + 4 \log M$$

which reduces to  $4 \log M$  for the realizable case (similar to Halving).

## Online Learning

- ▶ If we drop the assumption that  $|\Theta| < \infty$ , minimizing the regret may be achieved (!?) by online/stochastic/sequential gradient descent, which updates the parameters as

$$\boldsymbol{\theta}_{n+1} = \text{proj}_{\Theta}(\boldsymbol{\theta}_n - \eta_n \nabla L(t_n, y(\mathbf{x}_n, \boldsymbol{\theta}_n)))$$

where  $\text{proj}_{\Theta}(\mathbf{v}) = \arg \min_{\boldsymbol{\theta} \in \Theta} \|\mathbf{v} - \boldsymbol{\theta}\|$ , and it is needed only if  $\Theta$  is a subset of  $\mathbb{R}^D$ .

- ▶ To ensure that stochastic gradient descent converges, we need to check that

$$\sum_{n=1}^{\infty} \eta_n = \infty \text{ and } \sum_{n=1}^{\infty} \eta_n^2 < \infty$$

e.g.  $\eta_n = 1/n$ .

- ▶ Convergence vs adaptation or concept drift.
- ▶ Minimizing the regret implies looking at the past. Instead, we may want to minimize the expected loss in the future. Then, we want to minimize

$$\mathbb{E}_{\{\mathbf{x}, T\}}[L(t, y(\mathbf{x}, \boldsymbol{\theta}))]$$

- ▶ One solution is to use a running average:

$$\bar{\boldsymbol{\theta}}_n = \frac{1}{n} \sum_{i=1}^n \boldsymbol{\theta}_i$$

## Online Kernel Methods

- ▶ Kernel classifier:

$$y(\mathbf{x}) = \begin{cases} 0 & \text{if } \sum_n \mathbf{1}_{\{t_n=1\}} k\left(\frac{\mathbf{x}-\mathbf{x}_n}{h}\right) \leq \sum_n \mathbf{1}_{\{t_n=0\}} k\left(\frac{\mathbf{x}-\mathbf{x}_n}{h}\right) \\ 1 & \text{otherwise} \end{cases}$$

- ▶ Support vector machines:

$$y(\mathbf{x}) = \sum_{m \in S} a_m t_m k(\mathbf{x}, \mathbf{x}_m) + b$$

- ▶ The kernel classifier implies no learning and, thus, it can be applied to online learning directly. Deploying it may however become computationally demanding at some point, since it sums over all the training points.
- ▶ Training a support vector machine need to be adapted for online learning, i.e. retraining is not an option as the training data may grow very fast. Deploying it is feasible due to its sparseness.

---

## Online SVM

---

- 1  $\mathcal{S} = \emptyset$
  - 2  $b = 0$
  - 3 Receive a random example  $(\mathbf{x}_i, t_i)$
  - 4 Compute  $y(\mathbf{x}_i) = \sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_i, \mathbf{x}_m) + b$
  - 5 If  $t_i y(\mathbf{x}_i) \leq 0$  then
  - 6      $\mathcal{S} = \mathcal{S} \cup \{i\}$
  - 7      $a_i = 1$
  - 8 Go to step 3
- 

- ▶ It converges to a separating hyperplane in the separable case.
- ▶ It does not attempt to maximize the margin.
- ▶ It only adds support vectors.
- ▶ Therefore, it may overfit the training data (and increase computational cost for deploying it).

## Online Kernel Methods

---

### Budget online SVM (input: $\beta$ and $M$ )

---

- 1  $\mathcal{S} = \emptyset$
  - 2  $b = 0$
  - 3 Receive a random example  $(\mathbf{x}_i, t_i)$
  - 4 Compute  $y(\mathbf{x}_i) = \sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_i, \mathbf{x}_m) + b$
  - 5 If  $t_i y(\mathbf{x}_i) \leq \beta$  then
  - 6      $\mathcal{S} = \mathcal{S} \cup \{i\}$
  - 7      $a_i = 1$
  - 8     If  $|\mathcal{S}| > M$  then  $\mathcal{S} = \mathcal{S} \setminus \{\arg \max_{m \in \mathcal{S}} t_m (y(\mathbf{x}_m) - a_m t_m k(\mathbf{x}_m, \mathbf{x}_m))\}$
  - 9 Go to step 3
- 

- ▶ **Quiz:** Why does the removal criterion in step 8 make sense ?
- ▶ It tolerates mild noisy data but does not maximize the margin.
- ▶ It may still overfit the training data (and increase computational cost for deploying it) due to its myopic nature.
- ▶ More sophisticated addition and removal criteria are needed, e.g. LASVM which handles noisy data (slack variables) and converges to the batch solution.

# Online Kernel Methods

- ▶ Experimental results for binary classification.
- ▶ Gaussian kernel, which implies dot product of 784 gray level pixel values.
- ▶ LIBSVM: Batch SVM.
- ▶ LASVM (x1/x2): Online SVM with one or two passes over the training data.

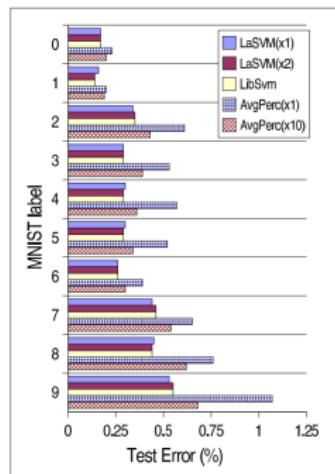


Figure 1: Compared test error rates for the ten MNIST binary classifiers.

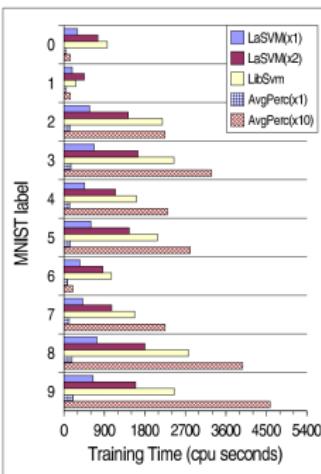


Figure 2: Compared training times for the ten MNIST binary classifiers.

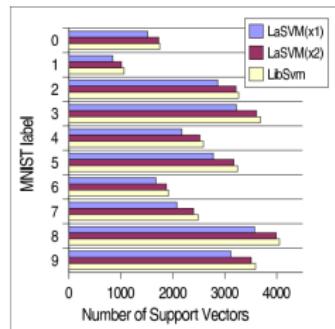


Figure 5: Compared numbers of support vectors for the ten MNIST binary classifiers.

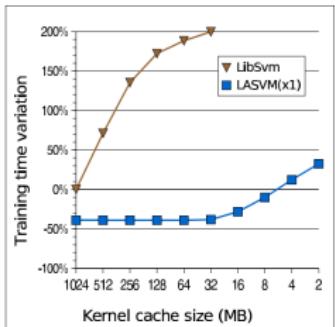


Figure 6: Training time variation as a function of the cache size. Relative changes with respect to the 1GB LIBSVM times are averaged over all ten MNIST classifiers.

## Online EM Algorithm

## EM algorithm

Set  $\pi$  and  $\mu$  to some initial values

Repeat until  $\pi$  and  $\mu$  do not change

Compute  $p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})$  for all  $n$  /\* E step \*/

Set  $\pi_k$  to  $\pi_k^{ML}$ , and  $\mu_{ki}$  to  $\mu_{ki}^{ML}$  for all  $k$  and  $i$  /\* M step \*/

## Mixture of Bernoullis

$$\pi_k^{ML} = \frac{\sum_n p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}{N}$$

$$\mu_{ki}^{ML} = \frac{\sum_n x_{ni} p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}{\sum_n p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}$$

## Mixture of Gaussians

$$\pi_k^{ML} = \frac{\sum_n p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}{N}$$

$$\boldsymbol{\mu}_k^{ML} = \frac{\sum_n \mathbf{x}_n p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}{\sum_n p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}$$

$$\sum_k^{ML} = \frac{\sum_n (\mathbf{x}_n - \boldsymbol{\mu}_k^{ML})(\mathbf{x}_n - \boldsymbol{\mu}_k^{ML})^T p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}{\sum_n p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}$$

- ▶ Online EM algorithm: Replace the E step with a stochastic E step such as

$$S_{1:n} = (1 - \eta_n) S_{1:n-1} + \eta_n \mathbb{E}_{\mathcal{Z}}[S_n | \mu_n, \pi_n]$$

where  $\{\eta_n\}$  is the learning rate, e.g.  $\eta_n = 1/n$ . Similarly for the mixture of Gaussians.

- Convergent under mild conditions. Adaptive too, e.g.  $\eta_n = 1/2$ .

## Summary

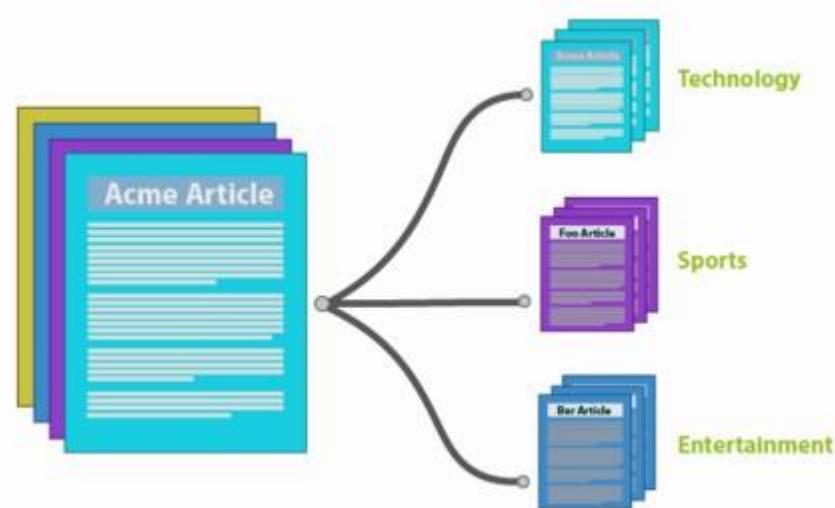
- ▶ New concepts: Adversarial environment, regret.
- ▶ New algorithms:
  - ▶ Realizable and finite class of models: Consistent and halving.
  - ▶ Finite class of models: Weighted majority.
  - ▶ Online SVM, budget online SVM, (LASVM), and online EM.

# Naïve Bayes classifiers Decision trees

Lecture 3b

# Naive Bayes classifiers: motivation

- Consider  $n$  labeled text documents
  - $Y = \{0,1\}$ ,  $0 = "Science fiction"$ ,  $1 = "Comedy"$
  - $X = \{X_1, \dots X_{100}\}$  does the document contain the keyword ( $0=\text{No}, 1=\text{Yes}$ )
    - $X_1$  corr. "space",  $X_2$  corr. "fun", ...
- Want to classify a new document



Källa: kdnuggets.com

# Naive Bayes classifiers: motivation

Idea: use Bayes classifier

$$p(Y = y|X) = \frac{P(X|Y = y)P(Y = y)}{\sum_j P(X|Y = y_j)P(Y = y_j)}$$

Chance of observing a given combination of words in science fiction

Proportion of science fiction documents

# Naive Bayes classifiers: motivation

- Attempt 1:
  - Model  $P(X = (x_1, \dots, x_p) | Y = y_i)$  and  $P(Y = y_i)$  as unknown parameters
  - Use data to derive those with Maximum Likelihood
  - Classify by use of the posterior distribution
- How many parameters?
  - How many different combinations of  $X$ ?  $2^p$
  - Amount of  $P(X = (x_1, \dots, x_p) | Y = y_i)$  is  $2 * 2^p - 2$ 
    - Probabilities for each  $Y$  sum up to one
- If  $p = 100$ ,  $10^{30}$  parameters need to be estimated → ouch!

# Naive Bayes classifiers

- Naive Bayes assumption: conditional independence

$$P(X = (x_1, \dots x_p) | Y = y) = \prod_{i=1}^p P(X_i = x_i | Y = y)$$

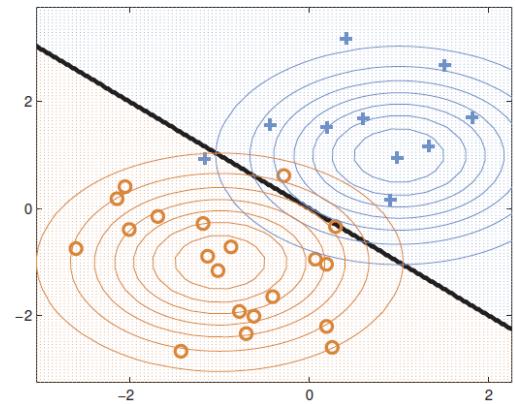
- How many parameters now?
  - $P(X_i = x_i | Y = y), i = 1, \dots p, x_i = \{0,1\}, y = \{0,1\}$   $2 * p$
- Is Naive Bayes assumption always valid?
  - $P(\text{Space, ship} | \text{SciFi}) = P(\text{Space} | \text{SciFi}) * P(\text{Ship} | \text{SciFi})$  ?

# Naive Bayes classifiers - discrete inputs

- Given  $D = \{(X_{m1}, \dots, X_{mp}, Y_m), m = 1, \dots, n\}$
- Assume  $X_i \in \{x_1, \dots, x_J\}, i = 1, \dots, p, Y \in \{y_1, \dots, y_K\}$
- Denote  $\theta_{ijk} = p(X_i = x_j | Y = y_k)$ 
  - How many parameters?  $(J - 1)Kp$
- Denote  $\pi_k = p(Y = y_k)$
- Maximum likelihood: assume  $\theta_{ijk}$  and  $\pi_k$  are constants
  - $\hat{\theta}_{ijk} = \frac{\#\{X_i=x_j \& Y=Y_k\}}{\#\{Y=y_k\}}$
  - $\hat{\pi}_k = \frac{\#\{Y=Y_k\}}{n}$
  - Classification using 0-1 loss:  $\hat{Y} = \arg \max_y p(Y = y | X)$

# Naive Bayes – continuous inputs

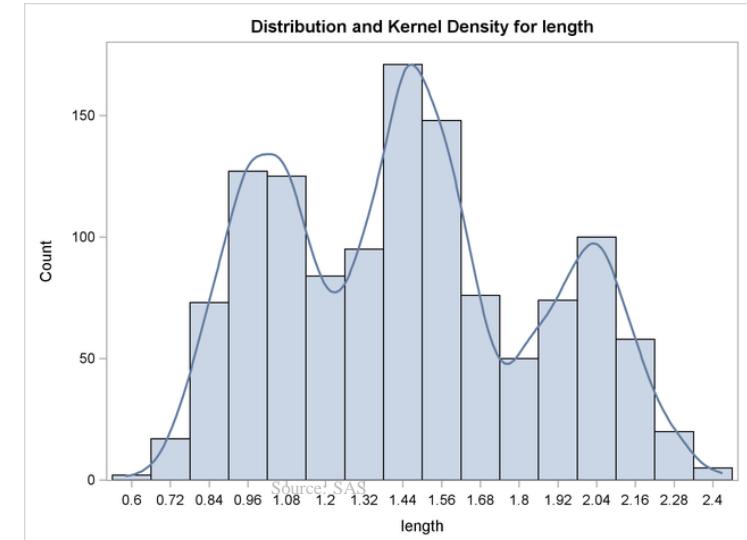
- $X_i$  are continuous
- Assumption A:  $x_j | y = C$  are univariate Gaussian
  - $p(x_j | y = C_i, \theta) = N(x_j | \mu_{ij}, \sigma_{ij}^2)$
- Therefore  $p(\mathbf{x} | y = C_i, \theta) = N(\mathbf{x} | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ 
  - $\boldsymbol{\Sigma}_i = \text{diag}(\sigma_{i1}^2, \dots, \sigma_{ip}^2)$



- Naive bayes is a special case of LDA (given A)
  - → MLE are means and variances (per class)

# Naive Bayes – continuous inputs

- Assumption B:  $p(x_j|y = C)$  are unknown functions of  $x_j$  that can be estimated from data
    - Nonparametric density estimation (kernel for ex.)
1. Estimate  $p(X_i = x_j|Y = y_k)$  using nonparametric methods
  2. Estimate  $p(Y = y_k)$  as class proportions
  3. Use Bayes rule and 0-1 loss to classify



# Naive Bayes in R

- `naiveBayes` in package **e1071**

**Example:** Satisfaction of householders with their present housing circumstances

```
library(MASS)
library(e1071)
n=dim(housing)[1]
ind=rep(1:n, housing[,5])
housing1=housing[ind,-5]

fit=naiveBayes(Sat~., data=housing1)
fit

Yfit=predict(fit, newdata=housing1)
table(Yfit,housing1$Sat)
```

> `table(Yfit,housing1$Sat)`

Yfit	Low	Medium	High
Low	294	162	144
Medium	20	23	20
High	253	261	504

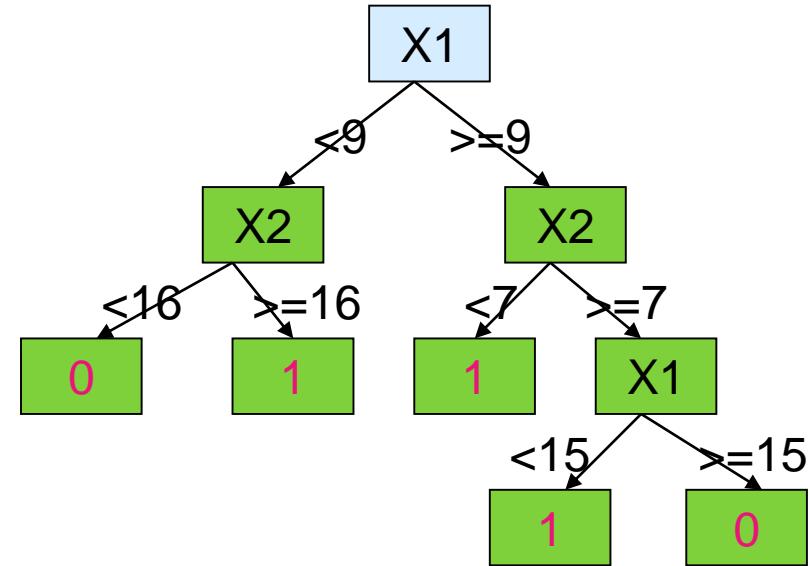
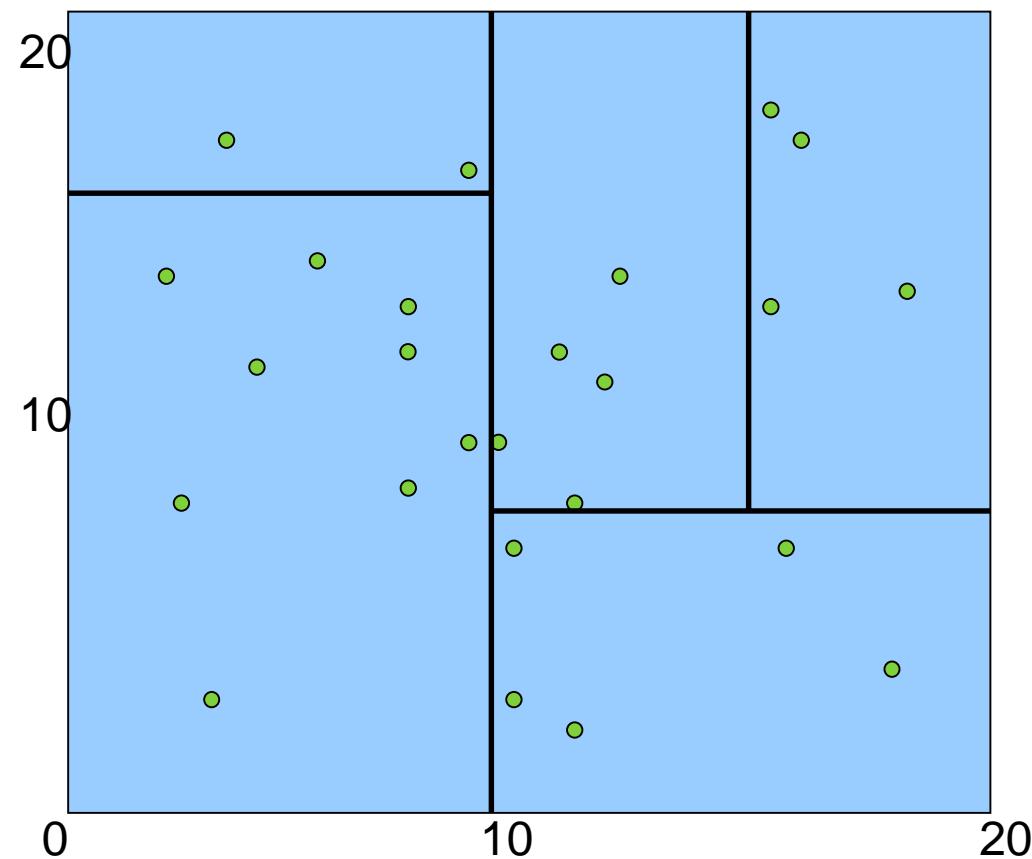
# Decision trees

## Idea

Split the domain of predictor set into the set of hypercubes (rectangles, cubes) and define the target value to be constant within each hypercube

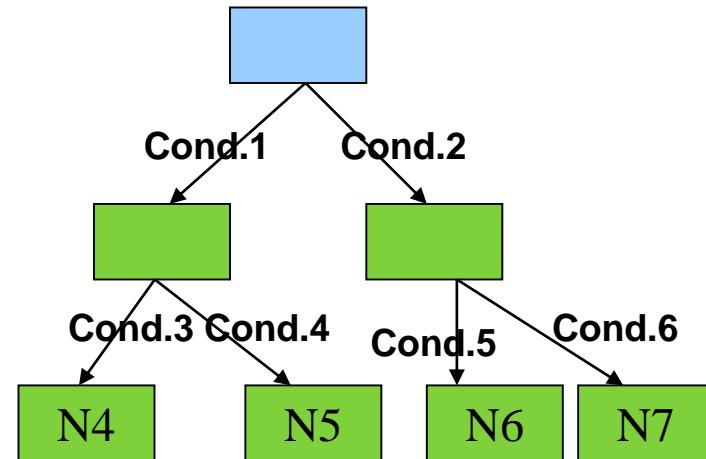
- Regression trees:
  - Target is a continuous variable
- Classification trees
  - Target is a class (qualitative) variable

# Classification tree toy example



# Definitions

- Root node
- Nodes
- Leaves (terminal nodes)
- Parent node, child node
- Decision rules
- A value is assigned to the leaves



# A classification problem

Create a classification tree that would describe the following patterns

ID	x1	x2	x3	x4	x5	x6	x7	y
Name	Body temperature	Skin cover	Gives birth	Aquatic creature	Aerial creature	Has legs	Hibernates	Class label
human	warm-blooded	hair	yes	no	no	yes	no	mammal
python	cold-blooded	scales	no	no	no	no	yes	non-mammal
salmon	cold-blooded	scales	no	yes	no	no	no	non-mammal
whale	warm-blooded	hair	yes	yes	no	no	no	mammal
frog	cold-blooded	none	no	semi	no	yes	yes	non-mammal
komodo	cold-blooded	scales	no	no	no	yes	no	non-mammal
bat	warm-blooded	hair	yes	no	yes	yes	yes	mammal
pigeon	warm-blooded	feathers	no	no	yes	yes	no	non-mammal
cat	warm-blooded	fur	yes	no	no	yes	no	mammal
shark	cold-blooded	scales	yes	yes	no	no	no	non-mammal
turtle	cold-blooded	scales	no	semi	no	yes	no	non-mammal
penguin	warm-blooded	feathers	no	semi	no	yes	no	non-mammal
porcupine	warm-blooded	quills	yes	no	no	yes	yes	mammal
eel	cold-blooded	scales	no	yes	no	no	no	non-mammal
salamander	cold-blooded	none	no	semi	no	yes	yes	non-mammal

# Several solutions

Tree 1

Creature =  
Non-  
mammal

Large misclassification  
rate!

Tree 3

Body  
temperature

Cold

Warm

Creature  
= Non-mammal

Tree 2

Body  
temperature

Cold

Warm

Creature  
= Non-mammal

Creature  
= Mammal

A lower misclassification rate

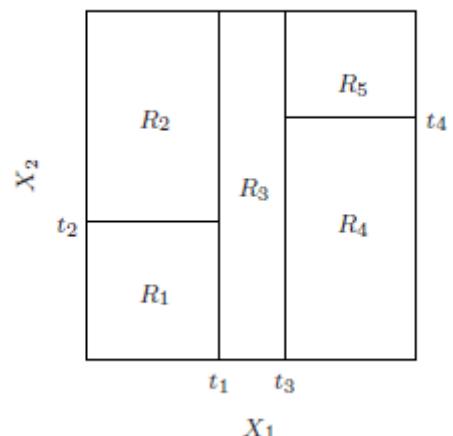
Green boxes represent pure nodes =nodes where observed values are  
the same

# Decision trees

- A tree  $T = \langle r_i, s_{r_i}, R_j, i = 1 \dots S, j = 1 \dots L \rangle$ 
  - $x_{r_i} \leq s_{r_i}$  splitting rules (conditions),  $S$ - their amount
  - $R_j$ -terminal nodes,  $L$ - their amount
  - means  $\mu_j$  in each terminal node

## Model:

- $Y|T$  for  $R_j$  comes from exponential family with mean  $\mu_j$
- Fitting by MLE:
  - Step 1: Finding optimal tree
  - Step 2: Finding optimal labels in terminal nodes



# Decision trees

Example:

- **Normal model** leads to regression trees
  - Objective: MSE
- **Multinoulli model** leads to classification trees
  - Objective: cross-entropy (**deviance**)

# Classification trees

- Target is categorical
- Classification probability  $p_{mk} = p(Y = k | X \in R_m)$  is estimated for every class in a node
- How to estimate  $p_{mk}$  for class  $k$  and node  $R_m$ ?

Class proportions

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$$

- For any node (leave), a label can be assigned

$$k(m) = \arg \max_k \hat{p}_{mk}$$

# Classification trees

- Impurity measure  $Q(R_m)$ 
  - $R_m$  is a tree node (region)
  - Node can be split unless it is pure

Misclassification error:  $\frac{1}{N_m} \sum_{i \in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)}$

Gini index:  $\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^K \hat{p}_{mk} (1 - \hat{p}_{mk})$

Cross-entropy or deviance:  $- \sum_{k=1}^K \hat{p}_{mk} \log \hat{p}_{mk}.$

- Note: In many sources, **deviance** is  $Q(R_m) N(R_m)$

**Example:** Cross –entropy is MLE of  $Y_j | T \sim \text{Multinomial}(p_{j1}, \dots p_{jc})$

# Fitting regression trees: CART

**Step 1: Finding optimal tree:** grow the tree in order to minimize global objective

1. Let  $C_0$  be a hypercube containing all observations
2. Let queue  $C = \{C_0\}$
3. Pick up some  $C_j$  from  $C$  and find a variable  $X_j$  and value  $s$  that split  $C_j$  into two hypercubes

and solve 
$$R_1(j, s) = \{X | X_j \leq s\} \text{ and } R_2(j, s) = \{X | X_j > s\}$$

$$\min_{j,s} [N_1 Q(R_1) + N_2 Q(R_2)]$$

4. Remove  $C_j$  from  $C$  and add  $R_1$  and  $R_2$
5. Repeat 3-4 as many times as needed (or until each cube has only 1 observation)

# CART: comments

- Greedy algorithm (optimal tree is not found)
- The largest tree will interpolate the data → large trees = **overfitting** the data
- Too small trees= **underfitting** (important structure may not be captured)
- Optimal tree length?

# Optimal trees

- **Postpruning**

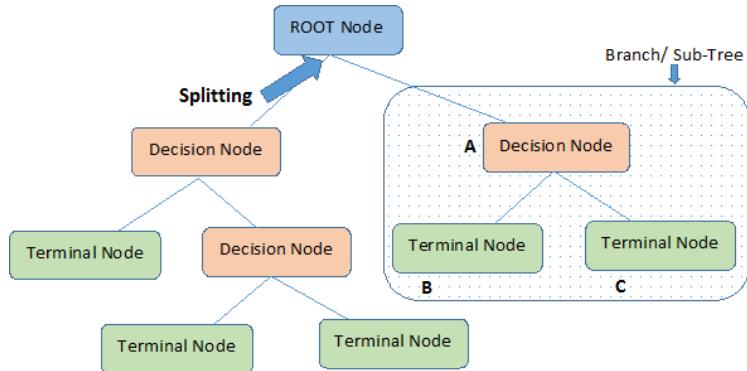
## Weakest link pruning:

1. Merge two leaves that have smallest  $N(\text{parent}) * Q(\text{parent}) - N(\text{leave1})Q(\text{leave1}) - N(\text{leave2})Q(\text{leave2})$
2. For the current tree T, compute

$$I(T) = \sum_{R_i \in \text{leaves}} N(R_i)Q(R_i) + \alpha|T|$$

$$|T| = \#\text{leaves}$$

3. Repeat 1-2 until the tree with one leave is obtained
4. Select the tree with smallest I(T)



Source: <http://www.analyticsvidhya.com>

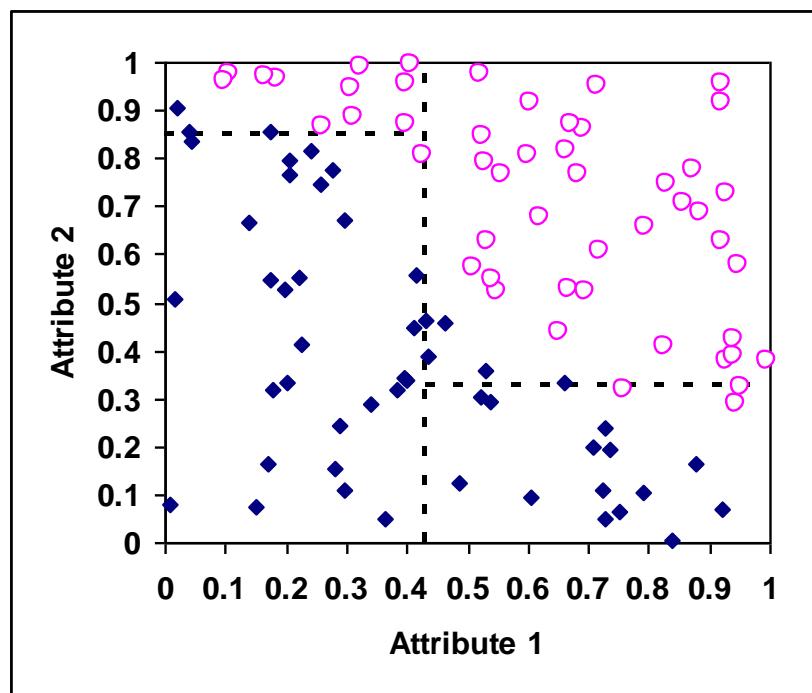
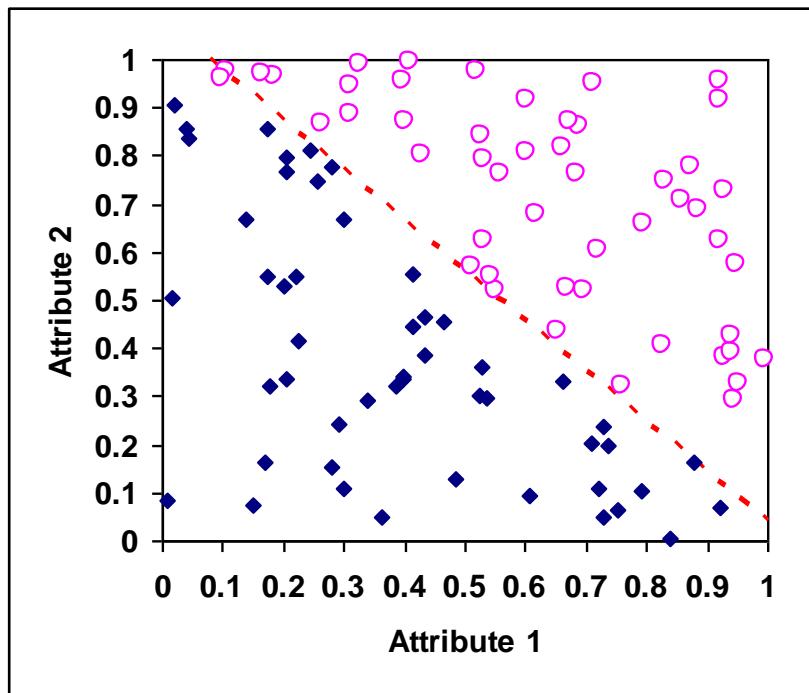
How to find the optimal  $\alpha$ ? Cross validation!

# Decision trees: comments

- Similar algorithms work for regression trees
  - Easy to interpret
  - Easy to handle all types of predictors in one model
  - **Automatic variable selection**
  - Relatively robust to outliers
  - Handle large datasets
- 
- Trees have high variance: a small change in response → totally different tree
  - Greedy algorithms → fit may be not so good
  - Lack of smoothness

# Decision trees: issues

- Large trees may be needed to model an easy system:



# Decision trees in R

- **tree** package
  - Alternative: **rpart**

```
tree(formula, data, weights, control, split = c("deviance", "gini"), ...)  
print(), summary(), plot(), text()
```

**Example:** breast cancer as a function av biological measurements

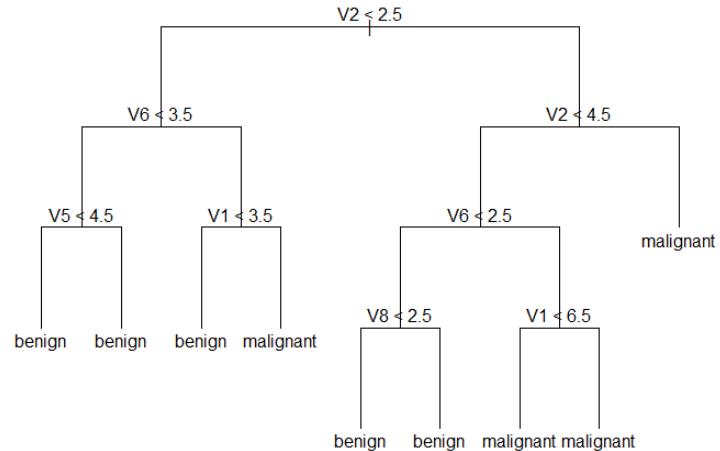
```
library(tree)  
n=dim(biopsy)[1]  
fit=tree(class~., data=biopsy)  
plot(fit)  
text(fit, pretty=0)  
fit  
summary(fit)
```

# Decision trees in R

- Adjust the splitting in the tree with *control* parameter (leaf size for ex)

```
> fit
node), split, n, deviance, yval, (yprob)
  * denotes terminal node

1) root 683 884.400 benign ( 0.650073 0.349927 )
  2) V2 < 2.5 418 108.900 benign ( 0.971292 0.028708 )
    4) V6 < 3.5 395 25.130 benign ( 0.994937 0.005063 )
      8) V5 < 4.5 389 0.000 benign ( 1.000000 0.000000 ) *
      9) V5 > 4.5 6 7.638 benign ( 0.666667 0.333333 ) *
     5) V6 > 3.5 23 31.490 benign ( 0.565217 0.434783 )
    10) V1 < 3.5 11 0.000 benign ( 1.000000 0.000000 ) *
    11) V1 > 3.5 12 10.810 malignant ( 0.166667 0.833333 ) *
  3) V2 > 2.5 265 217.900 malignant ( 0.143396 0.856604 )
  6) V2 < 4.5 90 120.300 malignant ( 0.388889 0.611111 )
  12) V6 < 2.5 30 27.030 benign ( 0.833333 0.166667 )
    24) V8 < 2.5 19 0.000 benign ( 1.000000 0.000000 ) *
    25) V8 > 2.5 11 15.160 benign ( 0.545455 0.454545 ) *
  13) V6 > 2.5 60 54.070 malignant ( 0.166667 0.833333 )
    26) V1 < 6.5 28 35.160 malignant ( 0.321429 0.678571 ) *
    27) V1 > 6.5 32 8.900 malignant ( 0.031250 0.968750 ) *
  7) V2 > 4.5 175 30.350 malignant ( 0.017143 0.982857 ) *
```



```
> summary(fit)
```

```
Classification tree:
tree(formula = class ~ ., data = biopsy)
Variables actually used in tree construction:
[1] "V2" "V6" "V5" "V1" "V8"
Number of terminal nodes: 9
Residual mean deviance: 0.1603 = 108 / 674
Misclassification error rate: 0.03221 = 22 / 683
```

# Decision trees in R

- Misclassification results

```
Yfit=predict(fit, newdata=biopsy, type="class")
table(biopsy$class,Yfit)
```

```
> table(biopsy$class,Yfit)
   Yfit
      benign malignant
benign        440       18
malignant       7      234
```

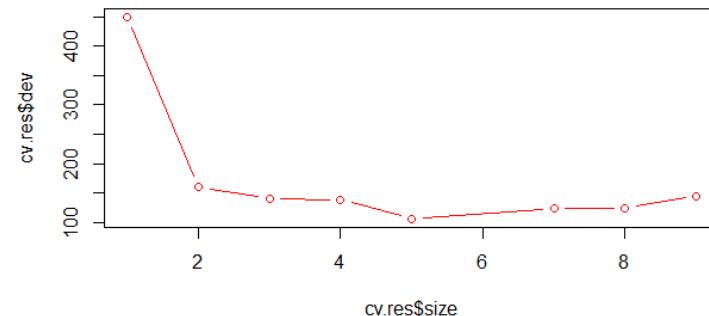
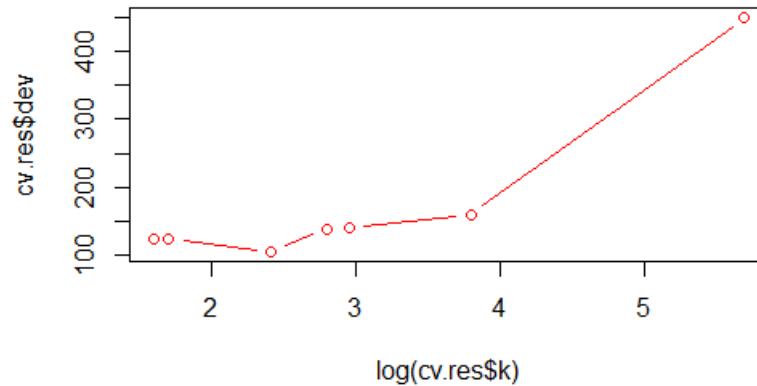
# Decision trees in R

- Selecting optimal tree by penalizing
  - Cv.tree()

```
set.seed(12345)
ind=sample(1:n, floor(0.5*n))
train=biopsy[ind,]
valid=biopsy[-ind,]

fit=tree(class~, data=train)
set.seed(12345)
cv.res=cv.tree(fit)
plot(cv.res$size, cv.res$dev, type="b",
col="red")
plot(log(cv.res$k), cv.res$dev,
type="b", col="red")
```

What is optimal number of leaves?



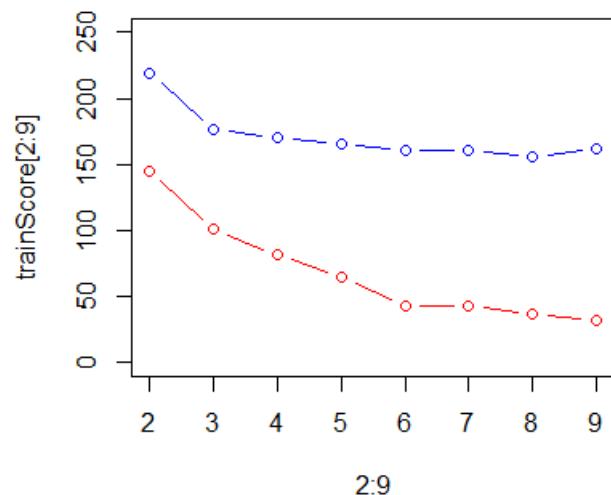
# Decision trees in R

- Selecting optimal tree by train/validation

```
fit=tree(class~, data=train)

trainScore=rep(0,9)
testScore=rep(0,9)

for(i in 2:9) {
  prunedTree=prune.tree(fit,best=i)
  pred=predict(prunedTree, newdata=valid,
type="tree")
  trainScore[i]=deviance(prunedTree)
  testScore[i]=deviance(pred)
}
plot(2:9, trainScore[2:9], type="b", col="red",
ylim=c(0,250))
points(2:9, testScore[2:9], type="b", col="blue")
```



What is optimal number of leaves?

# Decision trees in R

- Final tree: 5 leaves

```
finalTree=prune.tree(fit, best=5)
Yfit=predict(finalTree, newdata=valid,
type="class")
table(valid$class,Yfit)
```

```
> table(valid$class,Yfit)
   Yfit
      benign malignant
benign        222       8
malignant       6      114
```

# Text – document classification after 3 pages

## Lecture 3b block 2

### High-dimensional problems

Document	has('ball')	has('EU')	has('political_arena')	wordlen	Lex. Div.	Topic
Article1	Yes	No	No	4.1	5.4	Sports
Article2	No	No	No	6.5	13.4	Sports
:	:	:		:	:	:
ArticleN	No	No	Yes	7.4	11.1	News

# Wide data

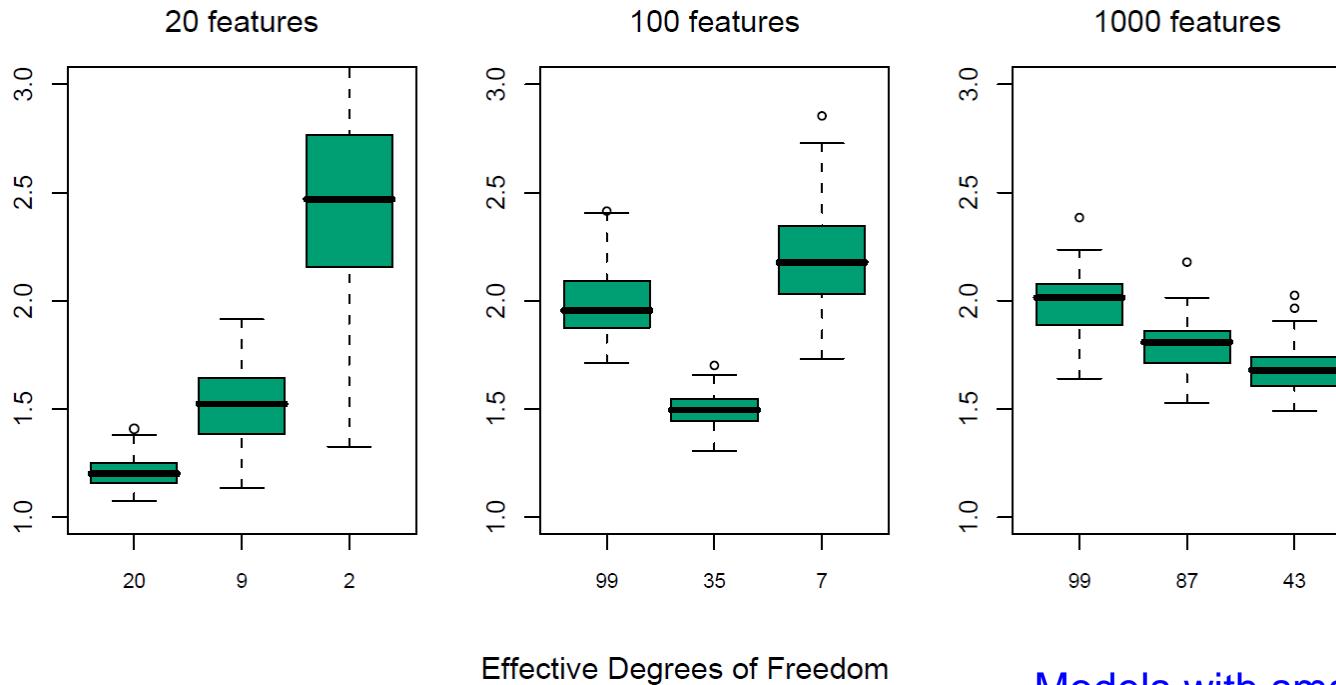
- **Wide data**  $p \gg n$ . Many variables, few data points.
  - Genomics
  - Text
- **Tall data:**  $p \ll n$ . Few variables, many data points.  
Most of applications
  - Economics, for ex. Currency exchange rates vs time
  - Industry, Car performance characteristics vs probability of malfunctioning
  - Surveys, customer satisfaction vs survey answers
- Tall and Wide. Supermarket scanners. Many purchases, many products.

# A problem with wide data

- Linear regression  $\mu = w^T x, Y \sim N(\mu, \sigma^2)^2$
- ML solution  $\hat{w} = (X^T X)^{-1} X^T Y$ 
  - $X$  is  $n \times p$ , has rank  $n$
  - $X^T X$  is  $p \times p$ , has rank  $n$
  - $\rightarrow X^T X$  is not invertible!
- Solutions:
  - **Dimensionality reduction**: PCA, PCR
  - **Shrinkage**: Lasso, Ridge, Elastic network
  - **Forward variable selection**
- Algorithms need sometimes be modified for wide data.

# Effective amount of features for wide data

- Linear response generated with different  $p$ ,  $n=100$
- Ridge is applied with different  $\lambda$



Source: Hastie et al (2009)

Models with smaller effective number of features have better prediction

# Classification: LDA

- Standard LDA

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

$$\hat{\boldsymbol{\mu}}_c = \frac{1}{N_c} \sum_{i:y_i=c} \mathbf{x}_i, \quad \hat{\boldsymbol{\Sigma}}_c = \frac{1}{N_c} \sum_{i:y_i=c} (\mathbf{x}_i - \hat{\boldsymbol{\mu}}_c)(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_c)^T$$

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N} \sum_{c=1}^k N_c \hat{\boldsymbol{\Sigma}}_c$$

- $\rightarrow \Sigma^{-1}$  does not exist...

# Classification: diagonal-covariance LDA

- Data is not enough to estimate dependences in covariance
- For wide data, we do **diagonal-covariance LDA** (naive Bayes):

$$\Sigma = \text{diag}(\sigma_1^2, \dots, \sigma_p^2)$$

- Discriminant function

$$\delta(x^{new}) = - \sum_{j=1}^p \frac{(x_j^{new} - \bar{x}_{kj})^2}{s_j^2} + 2 \log \pi_k$$

- $s_j^2 = \frac{1}{n} \sum_i n_i \text{var}(x_j | Y = C_i)$
- $\bar{x}_{kj} = \text{mean}(x_j | Y = C_k), \bar{x}_j = \text{mean}(x_j)$
- Classify to the highest discriminant function value
- **Drawback:** all features are in the model → difficult to use in interpretations.

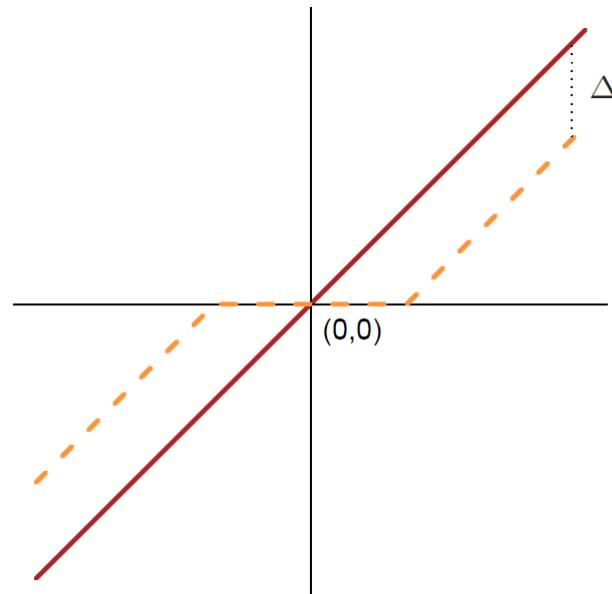
# Classification: NSC

## Nearest Shrunken Centroids

- Idea: Shrink classwise means towards overall mean

- Compute  $d_{kj} = \frac{\bar{x}_{kj} - \bar{x}_j}{m_k(s_j + s_0)}$
- Shrink  $d'_{kj} = sign(d_{kj})(|d_{kj}| - \Delta)_+$
- Set  $x'_{kj} = \bar{x}_j + m_k(s_j + s_0)d'_{kj}$

Only features with nonzero  $d'_{kj}$  contribute to classification! → insignificant features are shrunk!



# NSC: example

- Package **pamr**
  - **pamr.train()**
  - **pamr.cv**

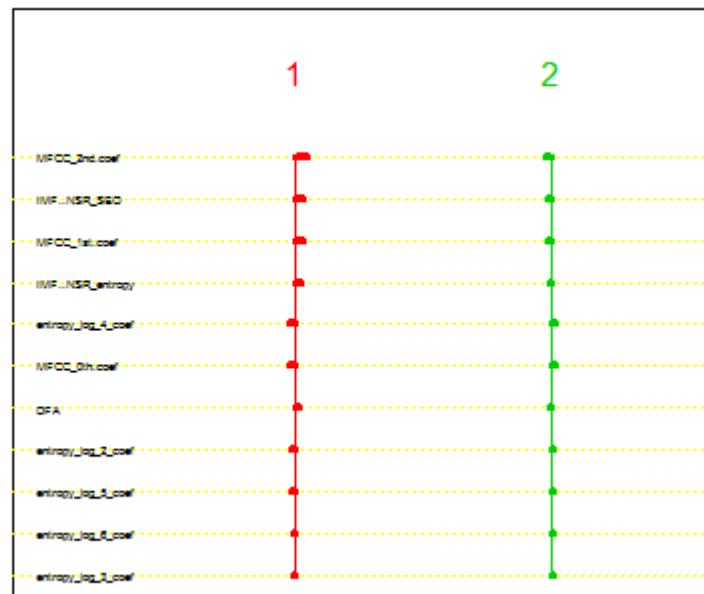
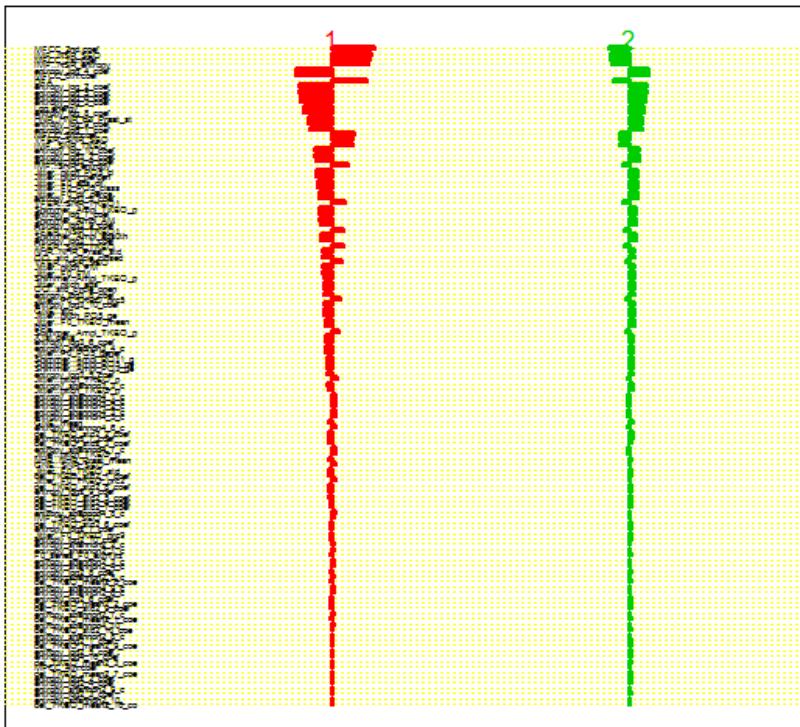
```
data0=read.csv2("voice.csv")
data=data0
data=as.data.frame(scale(data))
data$Quality=as.factor(data0$Quality)
library(pamr)
rownames(data)=1:nrow(data)
x=t(data[,-311])
y=data[[311]]
mydata=list(x=x,y=as.factor(y),geneid=as.character(1:nrow(x)), genenames=rownames(x))
model=pamr.train(mydata,threshold=seq(0,4, 0.1))
pamr.plotcen(model, mydata, threshold=1)
pamr.plotcen(model, mydata, threshold=2.5)

a=pamr.listgenes(model,mydata,threshold=2.5)
cat( paste( colnames(data)[as.numeric(a[,1])], collapse='\n' ) )

cvmodel=pamr.cv(model,mydata)
print(cvmodel)
pamr.plotcv(cvmodel)
```

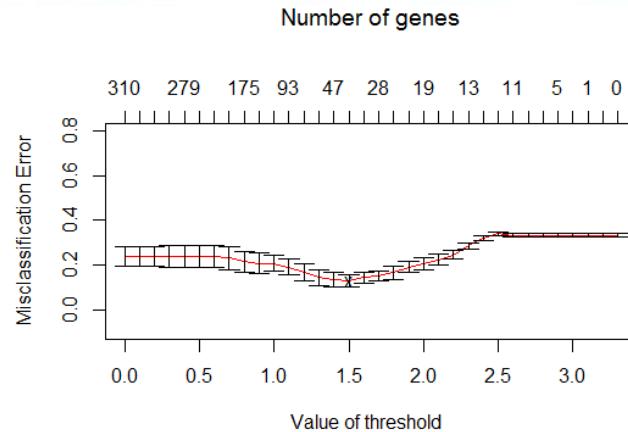
# NSC: example

- Centroid plot,  $\Delta= 1$  and  $\Delta= 2.5$



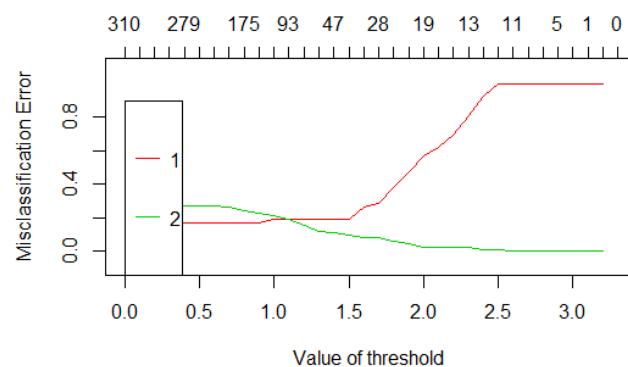
# NSC: example

```
> pamr.listgenes(model,mydata,threshold=2.5)
   id 1-score 2-score
[1,] 86  0.0897 -0.0449
[2,] 80  0.0702 -0.0351
[3,] 85  0.0652 -0.0326
[4,] 82  0.0517 -0.0259
[5,] 153 -0.0507  0.0253
[6,] 84  -0.05   0.025
[7,] 60  0.0359 -0.0179
[8,] 151 -0.0316  0.0158
[9,] 154 -0.0299  0.0149
[10,] 155 -0.0193  0.0096
[11,] 152 -0.018   0.009
```



- Confusion matrix optimal  $\Delta$

	Pred 1	Pred 2
True 1	33	9
True 2	5	79



# RDA

## Regularized discriminant analysis

- Another way of solving singularity of  $\Sigma$ 
  - $\gamma$  is some constant
$$\hat{\Sigma}(\gamma) = \gamma\hat{\Sigma} + (1 - \gamma)diag(\hat{\Sigma})$$
- $\gamma = 0 \rightarrow$  diagonal-covariance LDA
- $\gamma$  is chosen by CV
- R: rda() in **klaR**

# Regularized logistic regression

- Usual logistic regression

$$p(Y = C_i | x) = \frac{e^{w_{i0} + \mathbf{w}_i^T \mathbf{x}}}{\sum_{j=1}^K e^{w_{j0} + \mathbf{w}_j^T \mathbf{x}}} = \text{softmax}(w_{i0} + \mathbf{w}_i^T \mathbf{x})$$

- L<sub>p</sub>-Regularization:

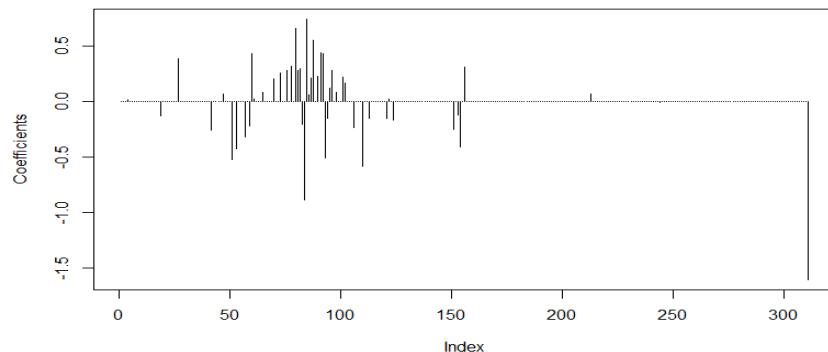
$$\max_{\mathbf{w}} \sum_{i=1}^n \log p(Y_i | x_i) - \frac{\lambda}{2} \sum_{k=1}^K \|\mathbf{w}_i\|^p$$

- Parameter redundancy is solved
- **L1 regularization:** some w are shrunk to 0
- Numerical optimization is used to solve
- R: LiblineaR() in package **LiblineaR**

# L1 logistic regression

- Voice rehabilitation

```
W=model2$W  
plot(t(W), type="h", ylab="Coefficients")
```



	Pred 1	Pred 2
True 1	41	1
True 2	0	84

Overfitted?

- Support Vector Machine do not suffer from  $p \gg n$  problem
  - Largest margin can be found even if the data is perfectly separable

# Computational shortcuts p>>n

- SVD decomposition  $X = UDV^T = RV^T$
- If model is linear in parameters and has quadratic penalties:
  - Transform data observations from X into R
  - Minimize loss (minus log likelihood) with R instead of X and get  $\theta$
  - Original parameters  $w = V\theta$
- Can be applied to many methods
- Example: ridge regression

# Elastic net

- L1 regularization

$$\min_w -\log p(D|w) + \lambda \|w\|_1$$

$$\|w\|_1 = \sum_i |w_i|$$

- For  $p > n$ , LASSO can extract at most  $n$  nonzero components
  - Severe regularization if  $p \gg n$
- L1 regularization → selects some feature among the correlated ones
- L2 regularization → w's of the correlated variables are shrunk towards each other are nonzero

# Elastic net

Combine L1 and L2 to diminish effect of L1 regularization.

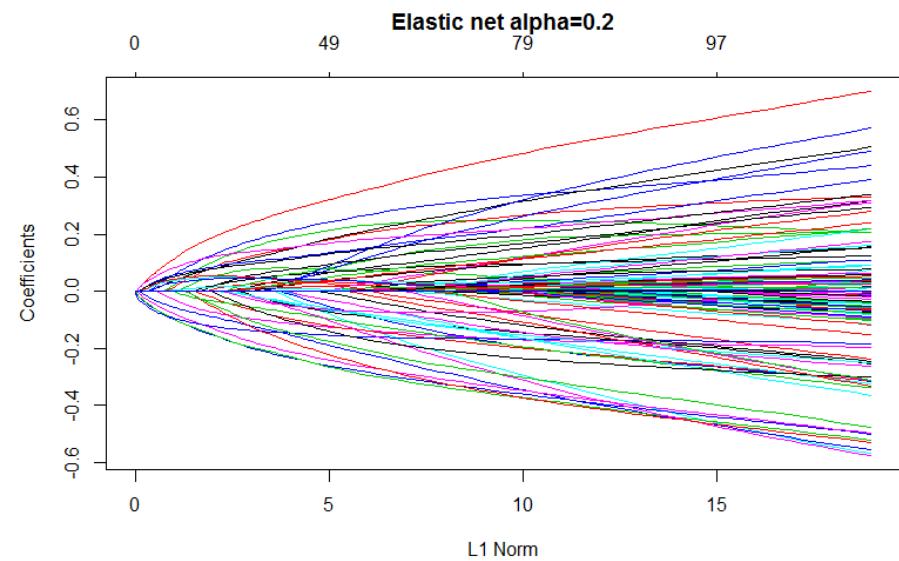
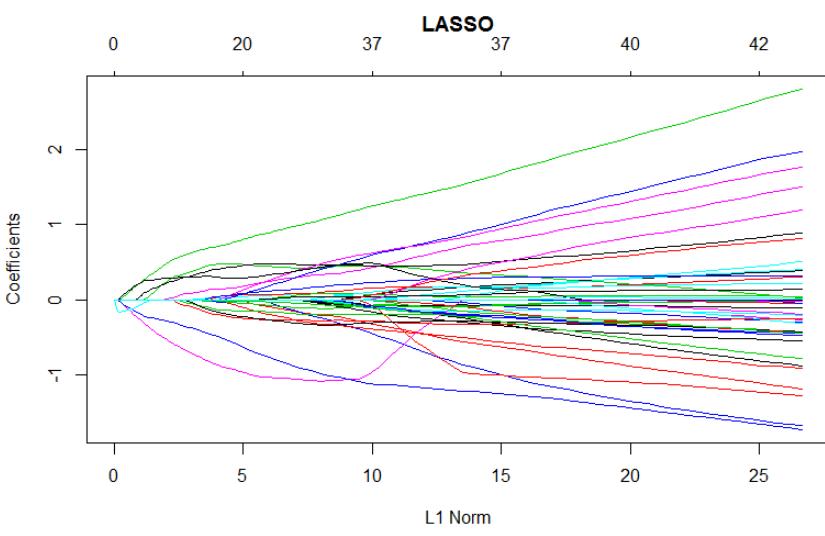
- Elastic net regularization:

$$\min_w -\log p(D|w) + \lambda(\alpha\|w\|_1 + (1-\alpha)\|w\|_2)$$

- $\alpha$  is set ad hoc or chosen by CV
- Elastic net may select more than  $n$  features
- R: `glmnet()` in **glmnet** package
  - Specify "family" for classification or regression

# Elastic net

- Voice rehabilitation



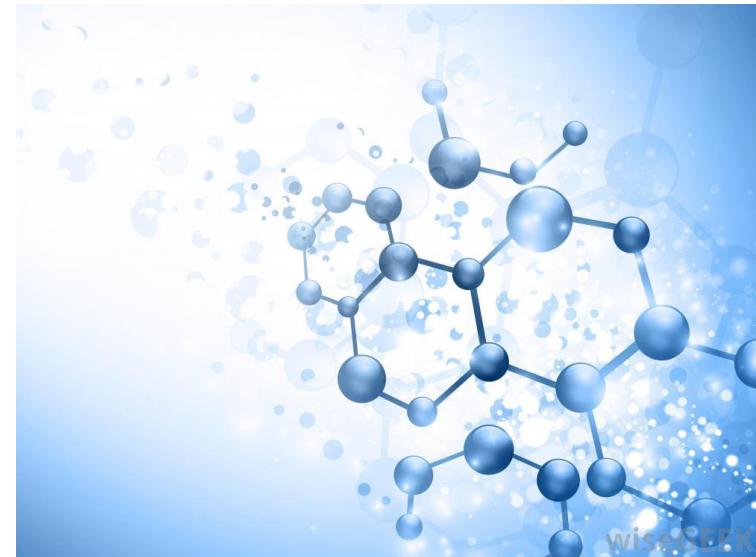
# Comparative analysis

- Gene expression data

Methods	CV errors (SE)	Test errors	Number of Genes Used
	Out of 144	Out of 54	
1. Nearest shrunken centroids	35 (5.0)	17	6,520
2. $L_2$ -penalized discriminant analysis	25 (4.1)	12	16,063
3. Support vector classifier	26 (4.2)	14	16,063
4. Lasso regression (one vs all)	30.7 (1.8)	12.5	1,429
5. $k$ -nearest neighbors	41 (4.6)	26	16,063
6. $L_2$ -penalized multinomial	26 (4.2)	15	16,063
7. $L_1$ -penalized multinomial	17 (2.8)	13	269
8. Elastic-net penalized multinomial	22 (3.7)	11.8	384

# When features are not available

- Sometimes it is difficult to define or use the feature set
  - Molecule
  - Text document
    - possible, but can be very high dimensional
- ..but a proximity measure  $K(x, x')$  is easier to define
  - Ex: How much one document is different from another one



Source: <http://images.wisegeek.com/illustration-of-a-molecule.jpg>

→ We can compute similarity matrix  
 $K = XX^T$

# When features are not available

- Many methods can use  $K$  instead of  $X$ 
  - Note: p is not involved in calculations!!
- SVM: kernel trick →  $K$  can be used directly
- K-Nearest neighbors
  - Transform similarity into distance  $d_{ij}^2 = K(x_i, x_i) + K(x_i, x_j) - 2K(x_i, x_j)$
  - Use distances to find neighbors
- Can also be done for
  - Logistic and multinomial regression with L2 penalty
  - LDA
  - PCA: kernel PCA

# Kernel PCA

- Usual PCA
  - Center  $X$
  - Find  $S\mathbf{u}_i = \lambda_i \mathbf{u}_i, S = \frac{1}{n} X^T X, S = [p \times p]$ 
    - $\mathbf{u}_i$  has dimension  $p$
  - Project data on PCs:  $Z = X U$
- **Problems:**  $X$  is unknown, and it can be  $p$  can be very large

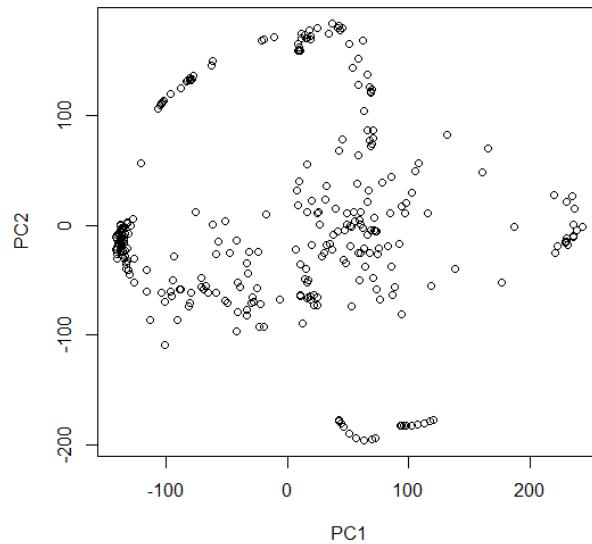
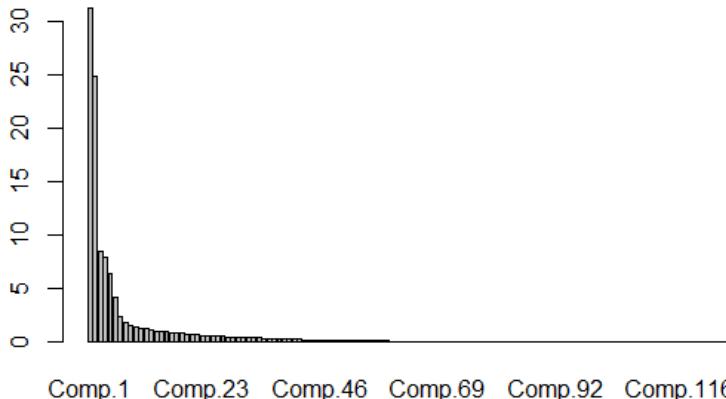
# Kernel PCA

- Kernel PCA: Equivalent formulation
1. Solve  $\mathbf{K}'\mathbf{a}_i = \lambda'_i \mathbf{a}_i, i = 1, \dots M$ 
    - $\mathbf{K} = \|K(\mathbf{x}_i, \mathbf{x}_j), i, j = 1, \dots n\|$
    - Centering  $\mathbf{K}' = \mathbf{K} - \mathbf{1}_n \mathbf{K} - \mathbf{K} \mathbf{1}_n + \mathbf{1}_n \mathbf{K} \mathbf{1}_n$
    - $\lambda_i = \lambda'_i / n$
  2. Scores for  $PC_i$ :  $z_i(\mathbf{x}) = \sum_{i=1}^n a_{in} K(\mathbf{x}, \mathbf{x}_n)$
- There are at most  $n$  eigenvectors even if  $p >> n$

# Kernel PCA in R

- Use `kPCA()` in **kernlab**

```
library(kernlab)
K <- as.kernelMatrix(crossprod(t(x)))
res=kPCA(K)
barplot(res@eig)
plot(res@rotated[,1], res@rotated[,2], xlab="PC1",
ylab="PC2")
```



# Feature assessment

- Which features are important?
  - Ex: Which protein values differ between normal and cancer samples
- P-values in our predictive models can not be computed (too few observations)
- → Traditional hypothesis testing is used

# Feature assessment

- Individual gene: t-test

$H_{0j}$ : treatment has no effect on gene  $j$

$H_{1j}$ : treatment has an effect on gene  $j$

$$t = \frac{\bar{x}_{2j} - \bar{x}_{1j}}{se_j}$$

- Alternatively, nonparametric tests (permutation tests) can be used to compare two populations
- Testing hypothesis for all genes? → multiple hypothesis testing
- Control family-wise error rate
  - Bonferroni correction:  $\alpha' = \alpha/M$
  - Ex:  $\alpha=0.05, M=12000 \rightarrow \alpha' \approx 10^{-6}$

	$X_j$	$Y$
...	...	0
...	...	0
...	...	...
...	...	1
...	...	1

mean:  $\bar{x}_{1j}$

mean:  $\bar{x}_{2j}$

In practice, no genes with such small p-values

# Feature assessment

- Hypothesis testing Voice Rehabilitation
  - Feature "MFCC\_2nd.coef"

```
res=t.test(MFCC_2nd.coef~Quality,data=data,  
alternative="two.sided")  
res$p.value
```

```
> res$p.value  
[1] 1.21246e-11  
~
```

```
res=oneway_test(MFCC_2nd.coef~as.factor(Qu  
ality), data=data,paired=FALSE)  
pvalue(res)
```

```
> pvalue(res)  
[1] 3.166942e-09  
~
```

# Feature assessment

- Alternative: **false discovery rate** (FDR)
  - Can not be exactly computed in practice

	Called nonsignif	Called signif	Total
H0 true	U	V	M0
H0 false	T	S	M1
Total	M-R	R	M

$$FDR = E \left( \frac{V}{R} \right)$$

# Feature assessment

- Benjamini-Hochberg method (BH method)
  - Shown that  $FDR(BH) < \alpha$  for independent hypotheses
  - → we can control FDR!

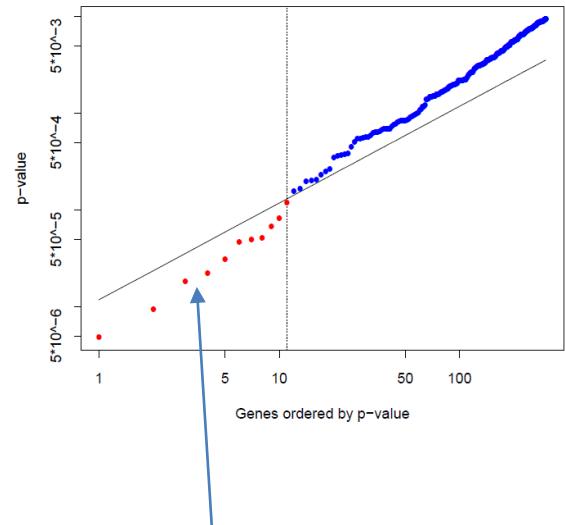
---

**Algorithm 18.2** Benjamini–Hochberg (BH) Method.

1. Fix the false discovery rate  $\alpha$  and let  $p_{(1)} \leq p_{(2)} \leq \dots \leq p_{(M)}$  denote the ordered  $p$ -values
2. Define

$$L = \max \left\{ j : p_{(j)} < \alpha \cdot \frac{j}{M} \right\}.$$

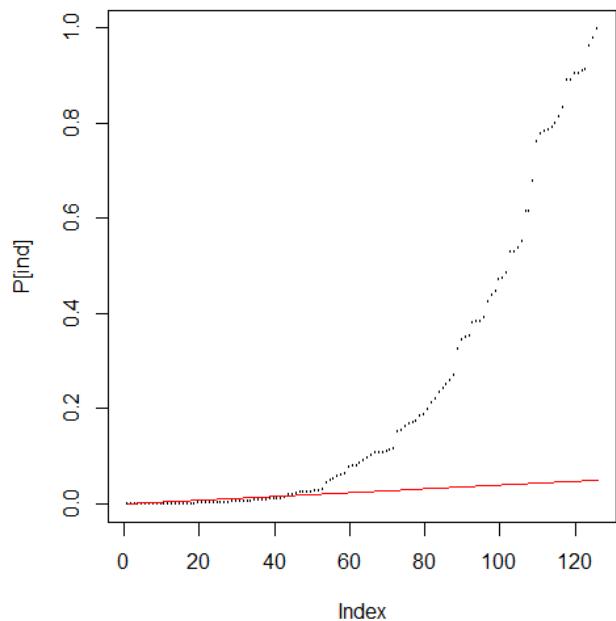
3. Reject all hypotheses  $H_{0j}$  for which  $p_j \leq p_{(L)}$ , the BH rejection threshold.
- 



Rejected hypotheses 32

# Feature assessment

- Voice rehabilitation



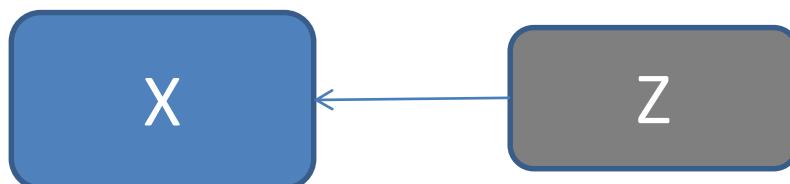
```
> cat( paste( Feats, collapse='\n' ) )
MFCC_2nd.coef
IMF..NSR_SEO
MFCC_1st.coef
IMF..NSR_entropy
MFCC_0th.coef
DFA
Log.energy
HNR..HNR_db_Praat_std
MFCC_3rd.coef
VFER..SNR_SEO
IMF..SNR_TKEO
IMF..SNR_entropy
jitter..pitch_PQ5_classical_schoentgen
jitter..pitch_percent
jitter..F0_abs_dif
jitter..F0_PQ5_classical_schoentgen
jitter..F0_dif_percent
VFER..SNR_TKEO1
Shimmer..Ampl_TKEO_prc25
Shimmer..Ampl_AM
VFER..NSR_TKEO1
shimmer..Ampl_abs0th_perturb
VFER..SNR_TKEO
NHR..NHR_Praat_std
OQ..std_cycle_closed
VFER..NSR_SEO
Jitter..F0_FM
Jitter..pitch_FM
Shimmer..Ampl_TKEO_prc75
Jitter..pitch_abs
OQ..std_cycle_open
Jitter..F0_TKEO_prc5
GNE..std
Jitter..pitch_PQ5_generalised_schoentgen
Jitter..F0_TKEO_mean
PPE
Shimmer..Ampl_TKEO_prc95
X1st.delta
Jitter..F0_PQ5_generalised_Schoentgen
shimmer..Ampl_PQ3_generalised_Schoentgen
shimmer..Ampl_PQ5_generalised_Schoentgen
shimmer..Ampl_PQ11_generalised_Schoentgen
Jitter..pitch_TKEO_prc25
```

# Lecture 4a

Latent variable models

# Latent variables

- Sometimes data depends on the latent variable we can not measure (hard to measure)
  - Answers on the test depend on Intelligence
  - Brain activity in the brain is measured by sensors
  - Stock prices depend on market confidence



Source: Leadliaison.com

# Latent variables

- Latent factor discovered → data storage may decrease a lot

3 | 3 | 3 | 3 | 3

- Latent factors
  - Center
  - Scaling
- Original vs compressed
  - $100 \times 100 \times 5 = 50000$
  - $100 \times 100 + 2 \times 5 + 2 \times 5 = 10020$

# Principal Component Analysis (PCA)

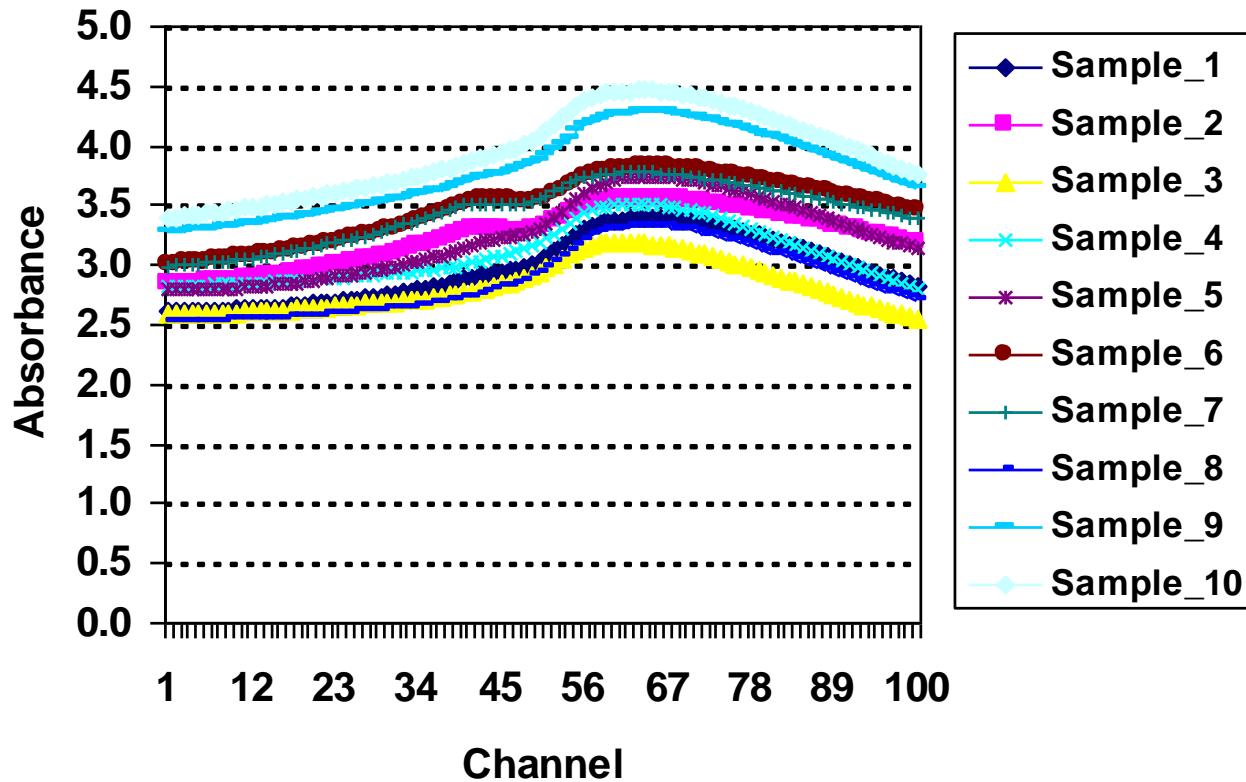
- PCA is a technique for reducing the complexity of high dimensional data
- It can be used to approximate high dimensional data with a few dimensions (latent features) → much less data to store
- New variables might have a special interpretation

## Applications

- Image recognition
- Information compression
- Subspace clustering
- ...

## Absorbance records for ten samples of chopped meat

### Parallel coordinate plot for “FAT”



**1 response variable  
(fat)**

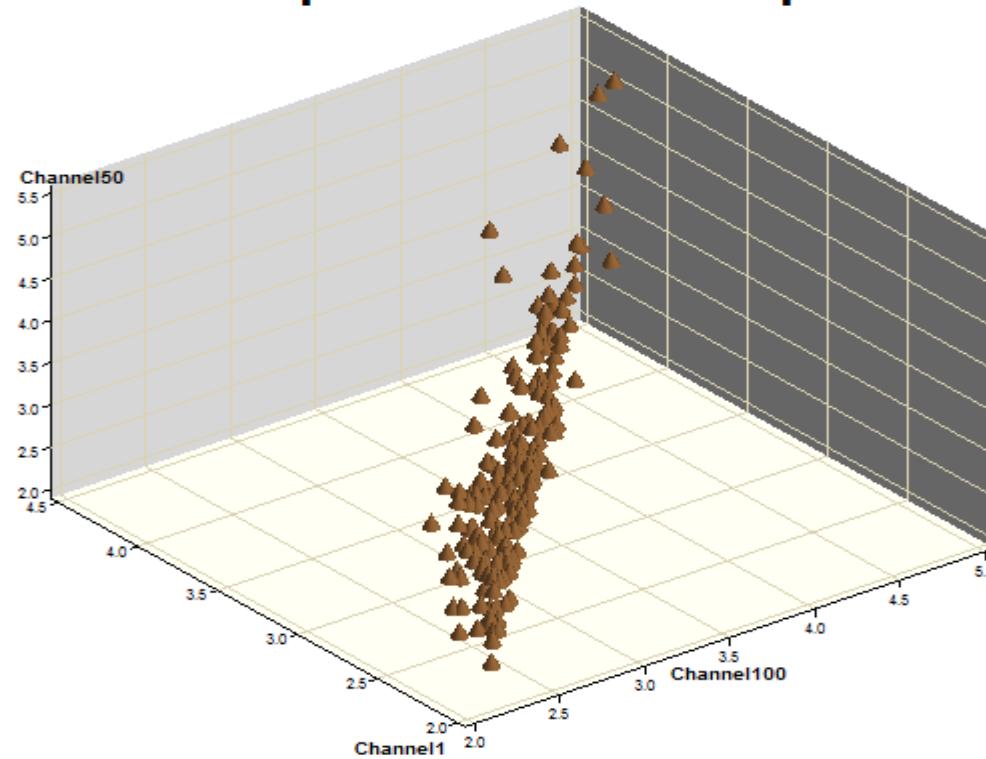
**100 predictors  
(absorbance at 100  
wavelengths or  
channels)**

**The predictors are  
strongly correlated  
to each other**

# 3-D plots of absorbance records for samples of meat

- channels 1, 50 and 100

**Scatterplot for three components**



# Principal components analysis

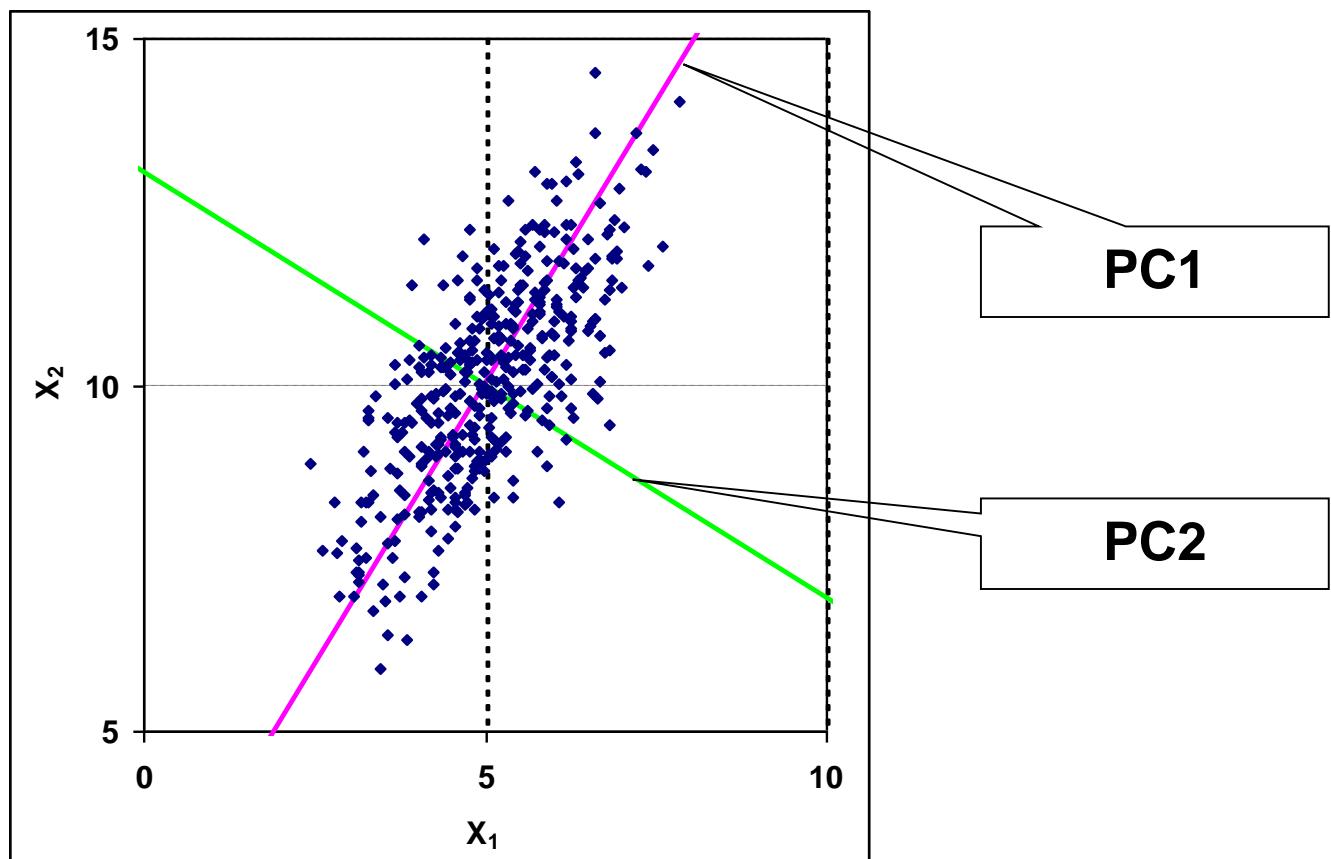
**Idea:** Introduce a new coordinate system (PC1, PC2, ...) where

- The first principal component (PC1) is the direction that maximizes the variance of the projected data
- The second principal component (PC2) is the direction that maximizes the variance of the projected data after the variation along PC1 has been removed
- The third principal component (PC3) is the direction that maximizes the variance of the projected data after the variation along PC1 and PC2 has been removed
- ....

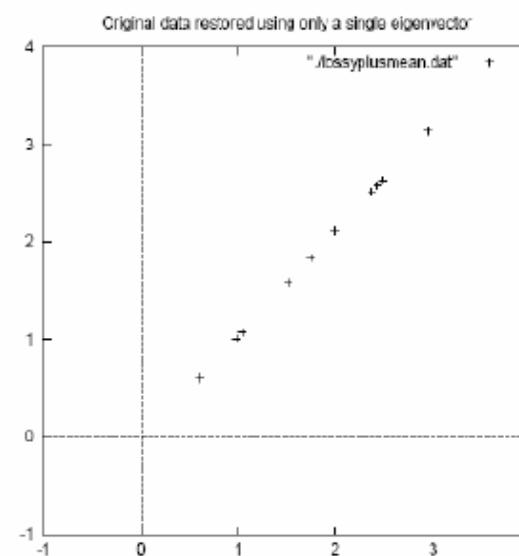
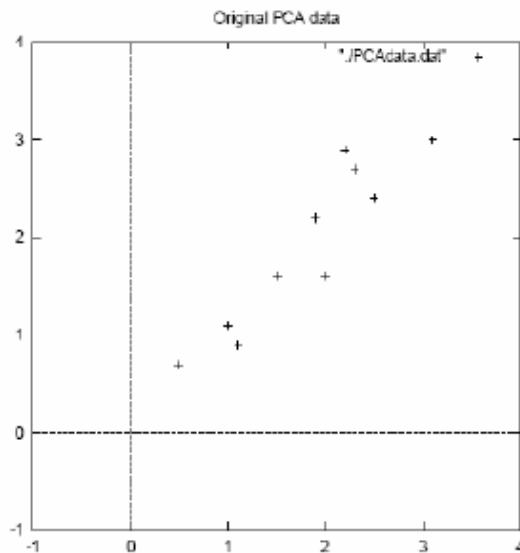
In the new coordinate system, coefficients corresponding to the last principal components are very small → can take away these columns

# Principal Component Analysis

- two inputs



# PCA- after reducing dimensionality

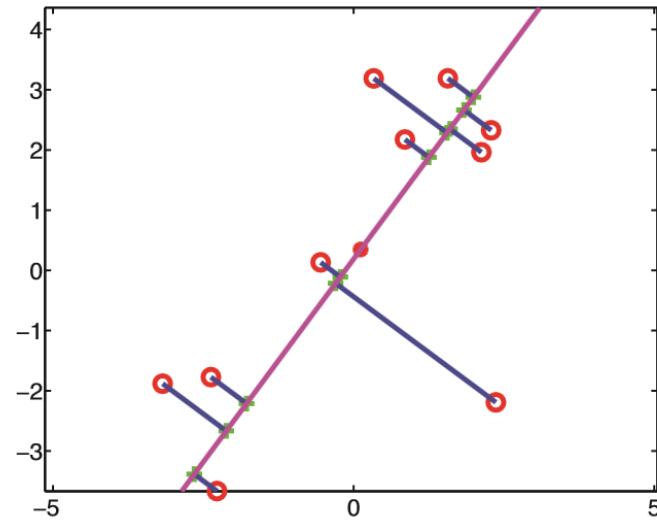


- Data became approximate (but less data to store)
- $PC_1, \dots, PC_M$  are actually eigenveccions of sample covariance (first largest eigenvalue,...,Mth largest egenvalue)

# PCA: another view

- Aim: minimize the distance between the original and projected data

$$\min_V \sum_{i=1}^N \|x_n - \tilde{x}_n\|^2$$



Source: Murphy

# PCA: computations

Data  $D = \|\mathbf{x}_1 \ \mathbf{x}_2 \ \dots \ \mathbf{x}_p\|, \quad \mathbf{x}_i = (x_{i1}, \dots, x_{in})$

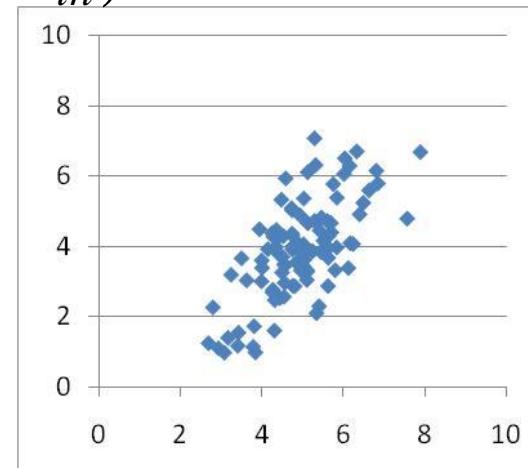
1. Centred data

$$X = \|\mathbf{x}_1 - \bar{\mathbf{x}}_1 \ \mathbf{x}_2 - \bar{\mathbf{x}}_2 \ \dots \ \mathbf{x}_p - \bar{\mathbf{x}}_p\|,$$

2. Covariance matrix

$$\mathbf{S} = \frac{1}{N} X^T X$$

3. Search for eigenvectors and eigenvalues of  $\mathbf{S}$



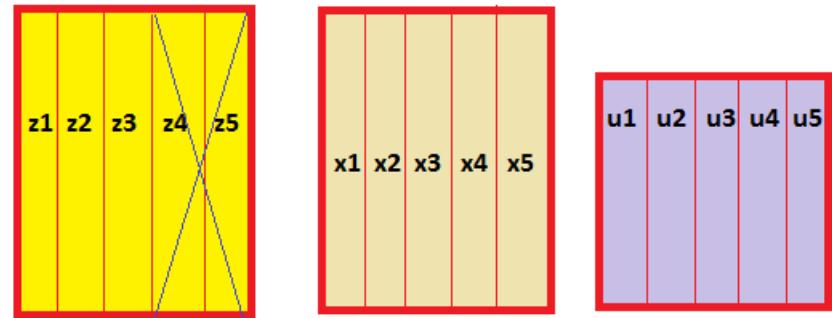
	Column 1	Column 2
Column 1	0.951	0.905
Column 2	0.905	1.883

# PCA: computations

4. Coefficients of any data point

$x = (x_1 \dots x_p)$  in the new coordinate system:

$$z = (z_1, \dots z_n), z_i = x^T u_i$$



Matrix form:  $Z = X U$

5. Discard principle components after some  $M$

6. New data will have dimensions  $N \times M$  instead of  $N \times p$

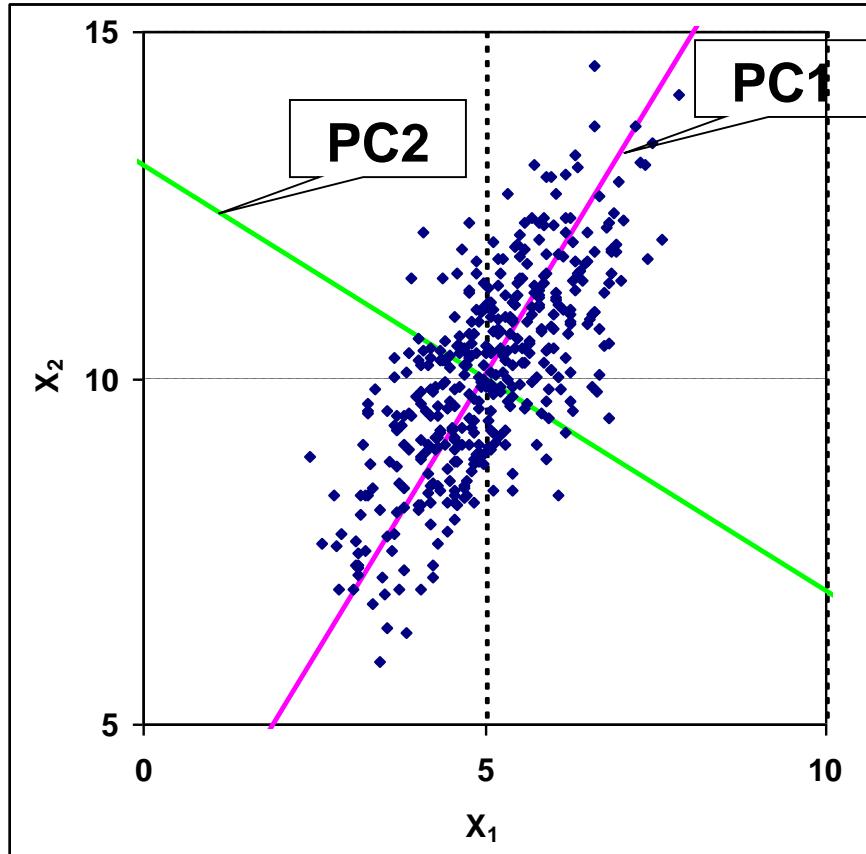
Getting approximate original data:

$$X' = ZU_M^T$$

Store:  $N \times M + p \times M$   
instead  $N \times p$

100\*50 vs  
100\*4+50\*4

# Principal Component Analysis



Eigenanalysis of the Covariance Matrix

Eigenvalue	2.8162	0.3835
------------	--------	--------

Proportion	0.880	0.120
------------	-------	-------

Cumulative	0.880	1.000
------------	-------	-------

Variable	PC1	PC2
----------	-----	-----

X1	0.523	0.852
----	-------	-------

X2	0.852	-0.523
----	-------	--------

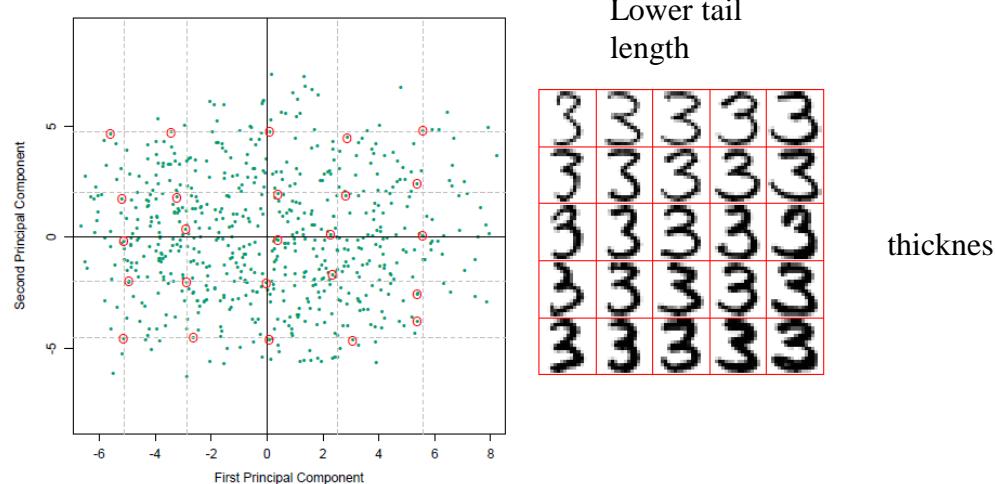
**Loadings (U)**

# Principal Component Analysis

- Digits: two eigenvectors extracted

$$\mathbf{x} = \text{[digit]} + z_1 \cdot \text{[eigenvector 1]} + z_2 \cdot \text{[eigenvector 2]}$$

- Interpretation of eigenvectors



# PCA in R

- Prcomp(), biplot(), screeplot()

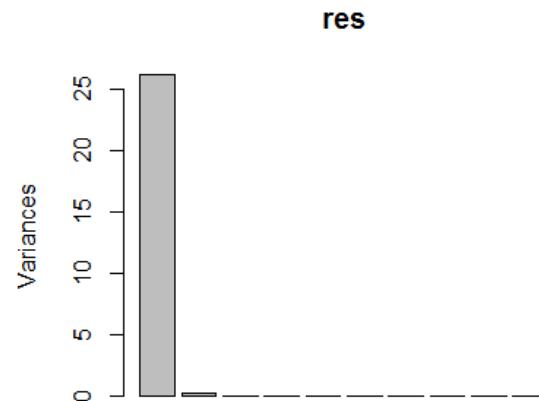
```
mydata=read.csv2("tecator.csv")
data1=mydata
data1$Fat=c()
res=prcomp(data1)
lambda=res$sdev^2
#eigenvalues
lambda
#proportion of variation
sprintf("%2.3f",lambda/sum(lambda)*100)
screeplot(res)
```

> `lambda`

```
[1] 2.612713e+01 2.385369e-01 7.844883e-02 3.018501e-01
[7] 2.052212e-04 1.084213e-04 2.077326e-05 1.150359e-05
```

> `sprintf("%2.3f",lambda/sum(lambda)*100)`

```
[1] "98.679" "0.901" "0.296" "0.114" "0.006"
[9] "0.000" "0.000" "0.000" "0.000" "0.000"
```



Only 1 component captures the  
99% of variation!

# PCA in R

- Principal component **loadings (U)**

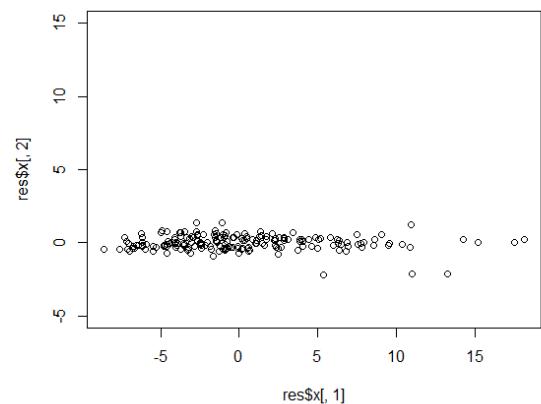
```
U=res$rotation  
head(U)
```

```
> head(U)
```

	PC1	PC2	PC3	
Channel1	0.07938192	0.1156228	0.08073156	-0.0927
Channel2	0.07987445	0.1170972	0.07887873	-0.0981
Channel3	0.08036498	0.1185571	0.07702127	-0.1031
Channel4	0.08085611	0.1200006	0.07515015	-0.1077
Channel5	0.08135022	0.1214075	0.07323819	-0.1119
Channel6	0.08184806	0.1227401	0.07125048	0.1156

- Data in (PC1, PC2) – **scores (Z)**

```
plot(res$x[,1], res$x[,2], ylim=c(-5,15))
```

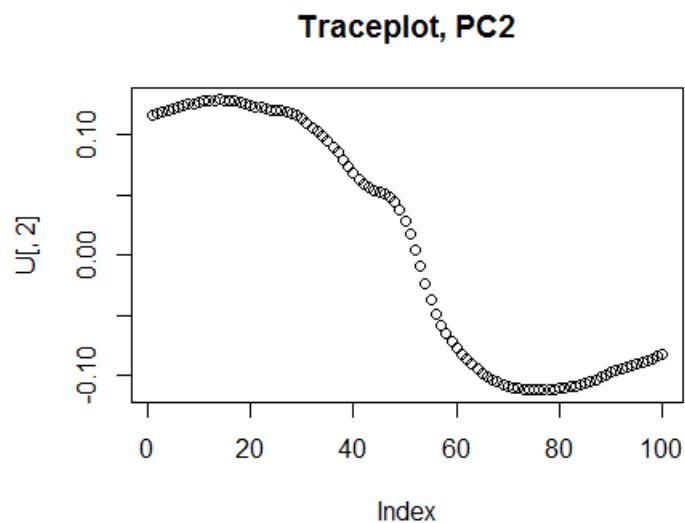
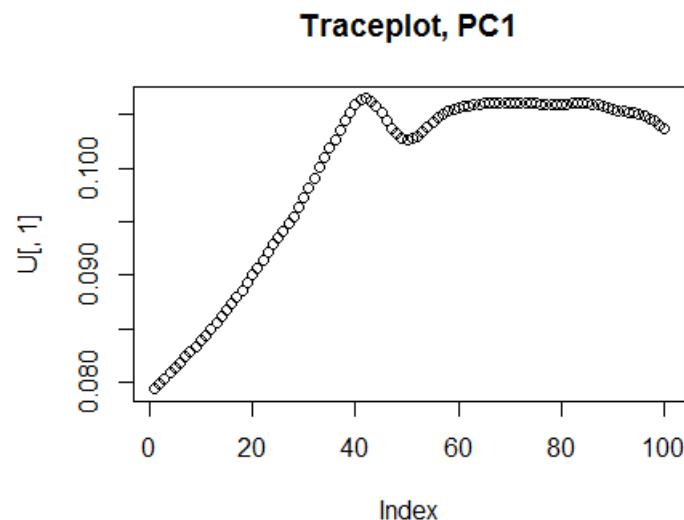


Do we need second dimension?

# PCA in R

- Trace plots

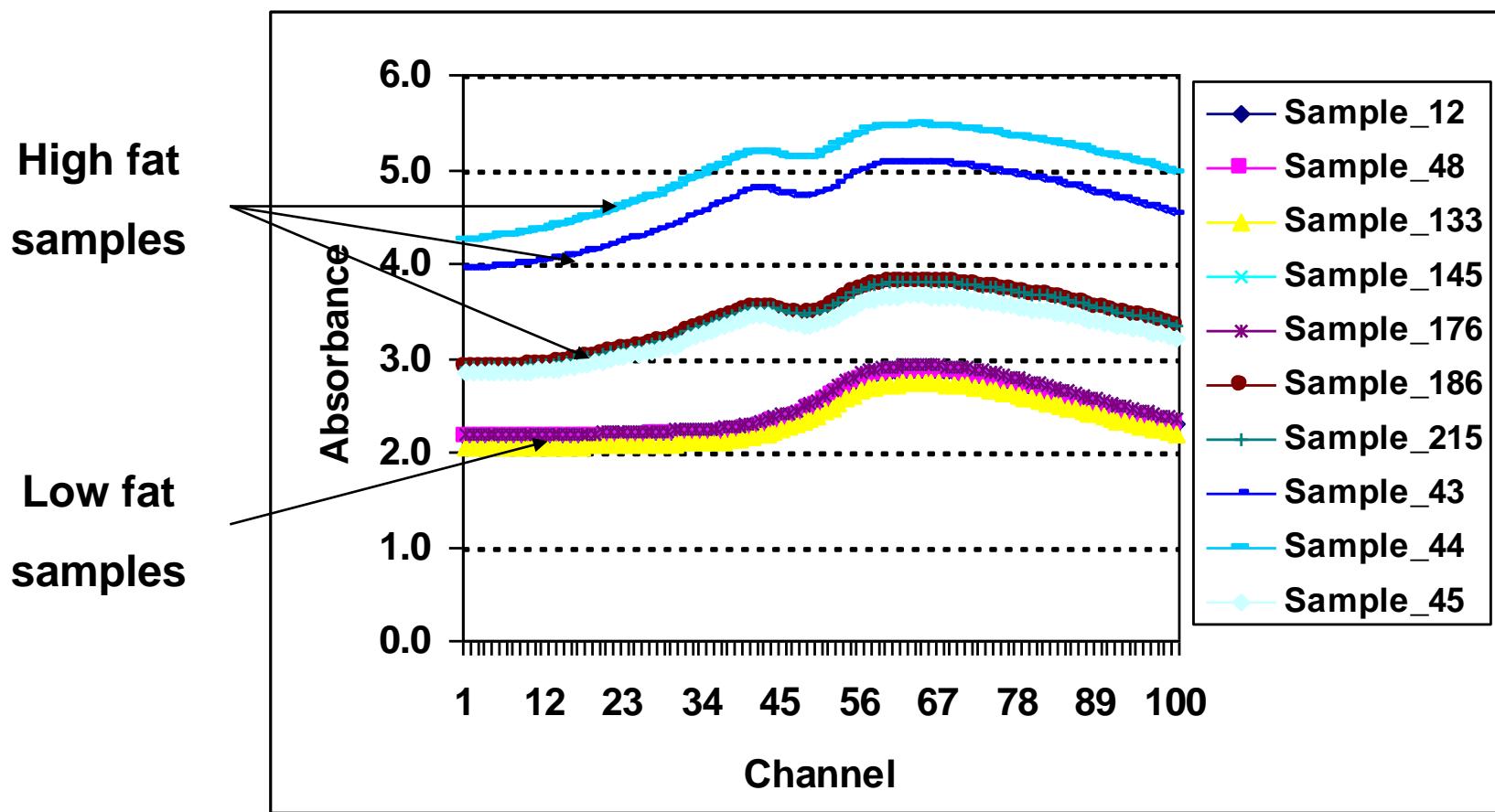
```
U=loadings(res)
plot(U[,1], main="Traceplot, PC1")
plot(U[,2],main="Traceplot, PC2")
```



Which components contribute to PC1-2?

## Absorbance records for ten samples of chopped meat

PCA2 captures the most of remaining variation



# PCA for high-dimensional data

- Standard PCA for  $p \gg N$ 
  - At most  $N$  eigenvalues are nonzero
  - Running time is  $O(p^3)$
- High-dimensional PCA
  1. Use  $S' = \frac{1}{N}XX^T$  (instead of  $S = \frac{1}{N}X^TX$ )
  2. Eigenvalues do not change
  3. Eigenvectors of  $S$  are  $X^T v_i$

# Probabilistic PCA

- $z_i$ -latent variables,  $x_i$ - observed variables

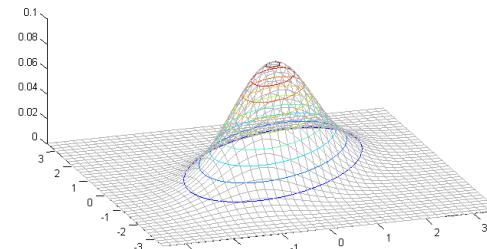
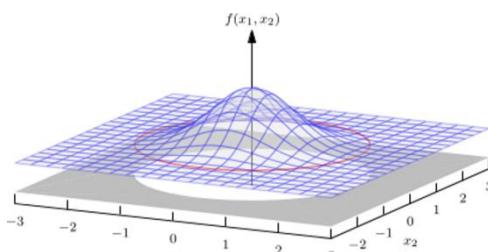
$$\mathbf{z} \sim N(\mathbf{0}, \mathbf{I})$$

$$x|z \sim N(x|Wz + \mu, \sigma^2 I)$$

- Alternatively

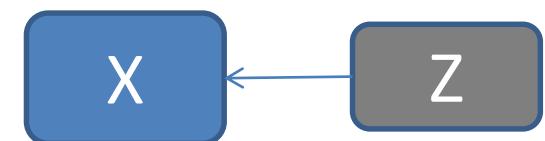
$$\mathbf{z} \sim N(\mathbf{0}, \mathbf{I}), x = \mu + Wz + \epsilon, \epsilon \sim N(\mathbf{0}, \sigma^2 I)$$

- **Interpretation:** Observed data ( $X$ ) is obtained by rotation, scaling and translation of standard normal distribution ( $Z$ ) and adding some noise.



$Z$

$X$



# Probabilistic PCA

- **Aim:** extract  $Z$  from  $X$
- Distribution of  $x$ :

$$\begin{aligned}x &\sim N(\mu, C) \\ C &= WW^T + \sigma^2 I\end{aligned}$$

- Rotation invariance
  - Assume that  $x$  was generated from  $z' = Rz, RR^T = I$ ,  $p(x)$  does not change!

$$x|z' \sim N(x|Wz' + \mu, \sigma^2 I)$$

- **Model will not be able find latent factors uniquely!** ☹
  - It does not distinguish  $z$  from  $z'$

# Probabilistic PCA

- Estimation of parameters: ML

**Theorem.** ML estimates are given by

$$\begin{aligned}\mu_{ML} &= \bar{x} \\ W_{ML} &= U_M(L_M - \sigma^2 I)^{\frac{1}{2}}R \\ \sigma_{ML}^2 &= \frac{1}{p-M} \sum_{i=M+1}^p \lambda_i\end{aligned}$$

- $U_M$  matrix of  $M$  eigenvectors
- $L_M$  diagonal matrix of  $M$  eigenvalues
- $R$  any orthogonal matrix

# Probabilistic PCA

- Estimation of  $Z$

- Use mean of posterior

$$\hat{z} = (W_{ML}^T W_{ML} + \sigma_{ML}^2 I)^{-1} W_{ML}^T (x - \mu)$$

- Connection to standard PCA

- Assume  $R = I, \sigma^2 = 0 \rightarrow$  get standard PCA components scaled by inverse root of eigenvalues

$$Z = XUL^{-\frac{1}{2}}$$

# Advantages of probabilistic PCA

- More settings to specify → more flexible
- Can be faster when  $M \ll p$
- Missing values can be handled
- $M$  can be derived if a Bayesian version is used
- Probabilistic PCA can be applied to classification problems directly
- Probabilistic PCA can generate new data

# Probabilistic PCA in R

- Use **pcaMethods** from Bioconductor
- Install
  - `source("https://bioconductor.org/biocLite.R")`
  - `biocLite("pcaMethods")`

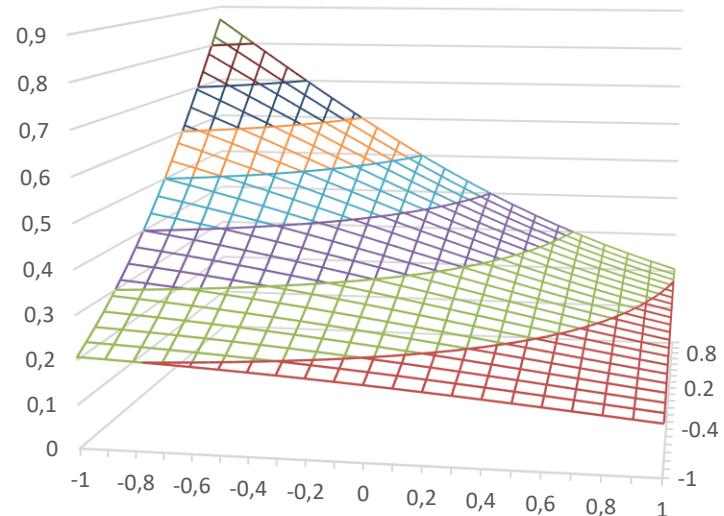
`Ppca(data, nPcs,...)`

**Results:** scores, loadings...

# Independent component analysis (ICA)

- Probabilistic PCA does not capture latent factors
  - Rotation invariance
- Let's choose distribution which is not rotation invariant → will get unique latent factors
- Choose non-Gaussian  $p(z)$

$$p(z) = \prod_{i=1}^M p(z_i)$$



# ICA

- Model

$$x = \mu + Wz + \epsilon, \quad \epsilon \sim N(0, \Sigma)$$

- Maximum likelihood ( $V = W^{-1}$ )

$$\max_V \sum_{i=1}^n \sum_{j=1}^p \log(p_i(v_i^T x_j))$$

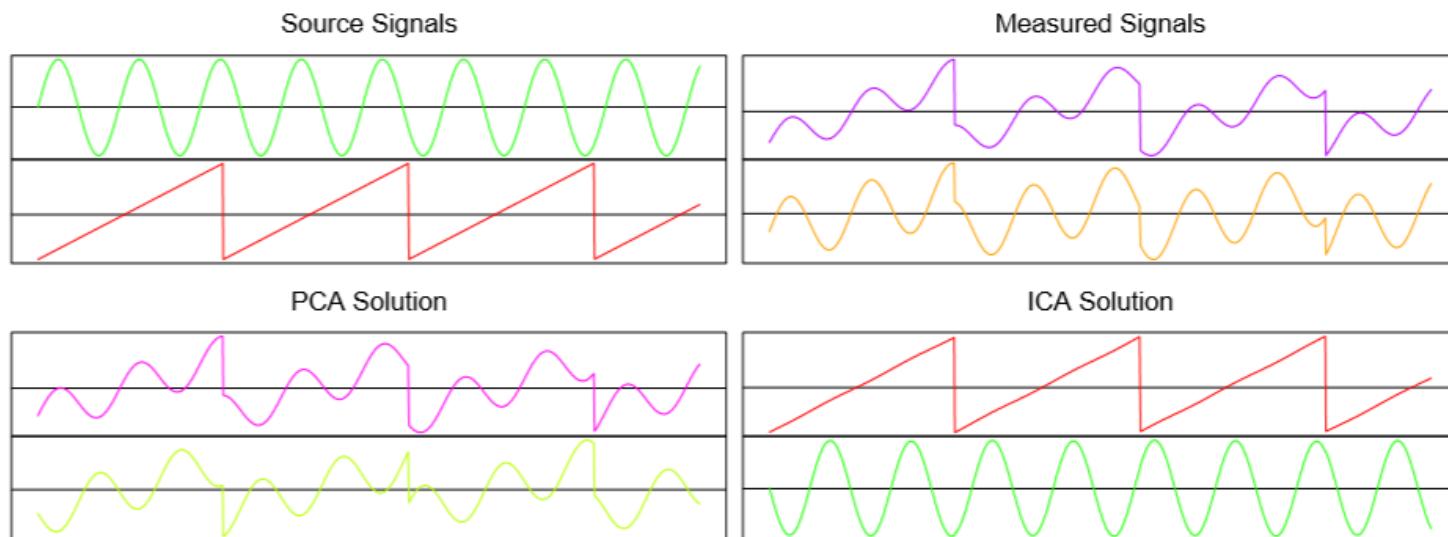
Subject to  $\|v_i\| = 1$

- Another equivalent form  $\rightarrow$  maximize negentropy
  - ICA looks for model which is as much non-Gaussian as possible

$$\text{negentropy}(z) \triangleq \mathbb{H}(\mathcal{N}(\mu, \sigma^2)) - \mathbb{H}(z)$$

# ICA

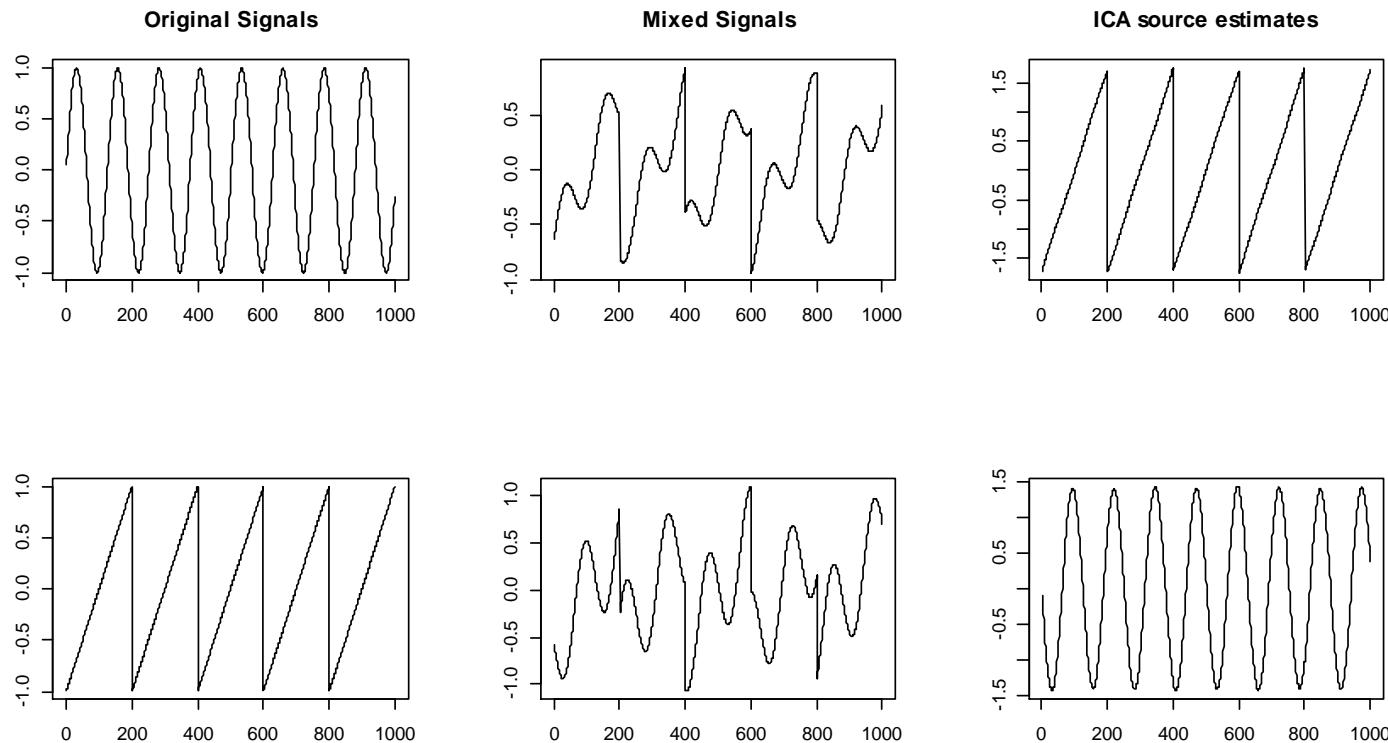
- Example



Source: Elem of stat learn by Hastie

# Independent component analysis: R

```
S <- cbind(sin((1:1000)/20), rep(((1:200)-100)/100), 5)) #Generate data  
A <- matrix(c(0.291, 0.6557, -0.5439, 0.5572), 2, 2)  
X <- S %*% A  
a <- fastICA(X, 2, alg.typ = "parallel", fun = "logcosh", alpha = 1,  
method = "R", row.norm = FALSE, maxit = 200, tol = 0.0001, verbose = TRUE) #ICA
```



# Uncertainty estimation and principal components

Lecture 4b

# Principle component regression

Step 1: Compute principal components  $v_1 \dots v_M$

Step 2: Compute derived data as  $z_i = Xv_i$

Step 3: Compute linear regression using data set  
Z with columns  $z_1 \dots z_M$ , Y

The result

$$\hat{Y}_{pcr} = \sum_{m=1}^M \frac{\langle z_m, Y \rangle}{\langle z_m, z_m \rangle} z_m$$

Coefficients

$$\hat{\beta}_{pcr} = \sum_{m=1}^M \frac{\langle z_m, Y \rangle}{\langle z_m, z_m \rangle} v_m$$

# PCR: comments

- Result depends on the scaling of features → standardize the original data
- If  $M < p$  → reduced regression
- If  $M = p$  → The fitted response will be the same
- Ridge regression shrinks the coefficients along the principal components, principal components regression discards  $p - M$  smallest components

# Partial Least Squares Regression (PLS)

- Idea with PLS is to find  $M < p$  orthogonal directions (as in PCR). The difference:

**PCR**

$$\begin{aligned} & \max_{\alpha} \text{Var}(\mathbf{X}\alpha) \\ \text{subject to } & \|\alpha\| = 1, \alpha^T \mathbf{S} v_\ell = 0, \ell = 1, \dots, m-1, \end{aligned}$$

**PLS**

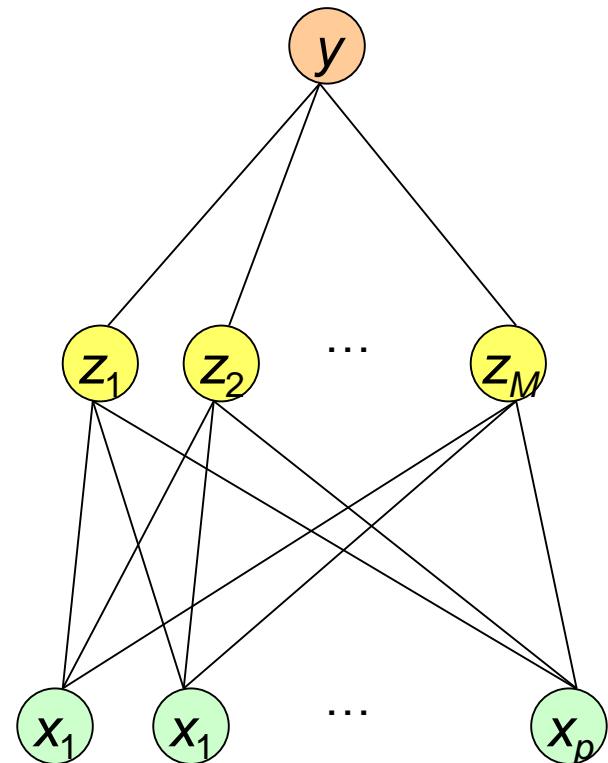
$$\begin{aligned} & \max_{\alpha} \text{Corr}^2(\mathbf{y}, \mathbf{X}\alpha) \text{Var}(\mathbf{X}\alpha) \\ \text{subject to } & \|\alpha\| = 1, \alpha^T \mathbf{S} \hat{\varphi}_\ell = 0, \ell = 1, \dots, m-1. \end{aligned}$$

# Partial Least Squares Regression (PLS)

Extract linear combinations of the features as new features, and then model the target as a linear function of these features

Select the intermediates so that the covariance with the response variable is maximized

Normally, the features are standardized to mean zero and variance one prior to the PLS analysis



# Partial least squares regression (PLS)

Step 1: Standardize features to mean zero and variance one

Step 2: Compute the first derived feature by setting

$$\mathbf{z}_1 = \sum_{j=1}^p \varphi_{1j} \mathbf{x}_j$$

where the  $\varphi_{1j}$  is projection of  $\mathbf{Y}$  on  $\mathbf{x}_j$

Step 3: Orthogonalize  $\mathbf{x}_1 \dots \mathbf{x}_m$  with respect to  $\mathbf{z}_1$

Step 4: repeat from step 2 and find  $\mathbf{z}_2 \dots \mathbf{z}_M$

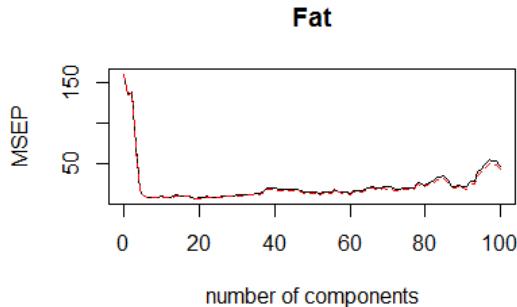
Step 5: Compute regression of  $\mathbf{Y}$  on  $\mathbf{z}_1 \dots \mathbf{z}_M$

# PCR and PLS: R

- Package **pls**
- **PCR**: `pcr(formula, ncomp, data, scale = FALSE, validation = c("none", "CV", "LOO"...))`
- **PLS**: `plsr(...)`

```
predictors=paste("Channel", 1:100, sep="")
f=formula(paste("Fat ~ ", paste(predictors,
collapse=" + )))

set.seed(12345)
pcr.fit=pcr(f, data=train, validation="CV")
summary(pcr.fit)
validationplot(pcr.fit, val.type="MSEP")
```



```
> summary(pcr.fit)
Data: X dimension: 150 100
Y dimension: 150 1
Fit method: svdpc
Number of components considered: 100

VALIDATION: RMSEP
Cross-validated using 10 random segments.
      (Intercept) 1 comps 2 comps 3 comps 4 comps
CV          12.68   11.65   11.75   8.506   4.211
adjCV       12.68   11.65   11.74   8.500   4.198
```

TRAINING: % variance explained						
s	1 comps	2 comps	3 comps	4 comps	5 comps	6 comp
X	98.65	99.59	99.88	99.99	100.00	100.0
Fat	16.79	16.98	56.44	89.97	93.62	95.2
6	--	--	--	--	--	--

# PCR

- Select 3 components

```
pcr.fit1=pcr(f, 3,data=train, validation="none")
summary(pcr.fit1)
coef(pcr.fit1)
scores(pcr.fit1)
l=loadings(pcr.fit1)
print(l,cutoff=0)
Yloadings(pcr.fit1)
plot(pcr.fit1)
```

```
> summary(pcr.fit1)
Data: X dimension: 150 100
Y dimension: 150 1
Fit method: svdpc
Number of components considered: 3
TRAINING: % variance explained
  1 comps  2 comps  3 comps
X    98.65    99.59    99.88
Fat   16.79    16.98    56.44
```

Coefficients in the original variables

```
> coef(pcr.fit1)
, , 3 comps
```

	Fat
Channel1	-2.61109872
Channel2	-2.56607281
Channel3	-2.52081831
Channel4	-2.47510841
Channel5	-2.42831883
Channel6	-2.37920922
Channel7	-2.32674365
Channel8	-2.27010925
Channel9	-2.20867856
Channel10	-2.14358670

# PCR

```
> l=loadings(pcr.fit1)  
> print(l,cutoff=0)
```

Loadings:

	Comp 1	Comp 2	Comp 3
Channel1	-0.079	-0.106	0.089
Channel2	-0.080	-0.108	0.088
Channel3	-0.080	-0.110	0.086
Channel4	-0.081	-0.112	0.084
Channel5	-0.081	-0.113	0.083
Channel6	-0.082	-0.115	0.081
Channel7	-0.082	-0.117	0.079
Channel8	-0.083	-0.118	0.077
Channel9	-0.083	-0.119	0.075
Channel10	-0.084	-0.121	0.073
Channel11	-0.084	-0.122	0.070
Channel12	-0.085	-0.123	0.068

Scores matrix (new coordinates)

```
> scores(pcr.fit1)
```

	Comp 1	Comp 2	Comp 3
155	-9.66855445	0.092805568	-0.1012873027
188	-1.04084544	-0.307978589	-0.3180672681
163	-1.92212872	-0.243714308	0.0124365801
214	1.27768585	-0.232939083	-0.4315433137
97	2.55797242	-0.741279740	0.1582517775
35	-11.17415228	2.124582502	0.2004058835
68	2.96005563	-0.312953959	0.1945895756
106	3.71331162	0.015766621	-0.1130155332

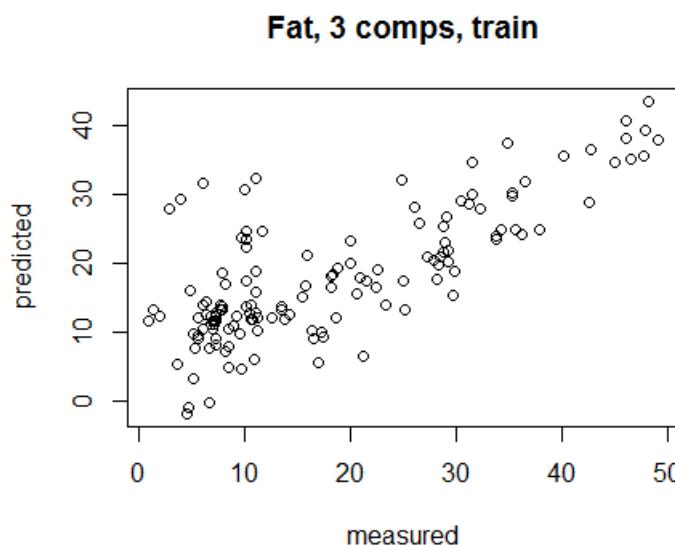
# PCR

```
> Yloadings(pcr.fit1)
```

Loadings:

	Comp 1	Comp 2	Comp 3
Fat	-1.015	1.114	-28.847

What is the regression equation  
in the new coordinates?

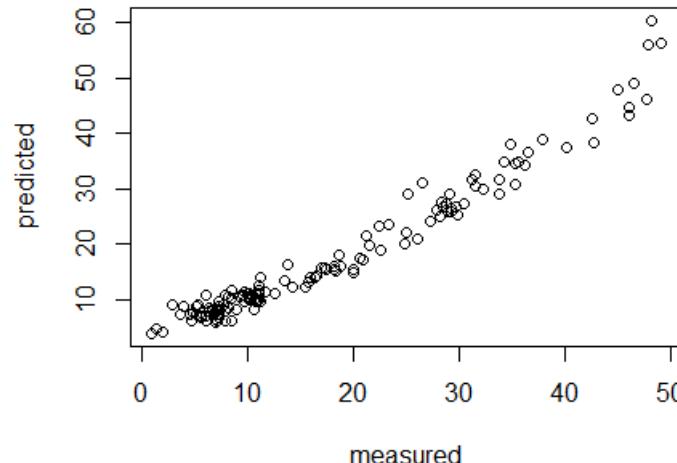


Is fit OK?

# PCR

- Now 6 components

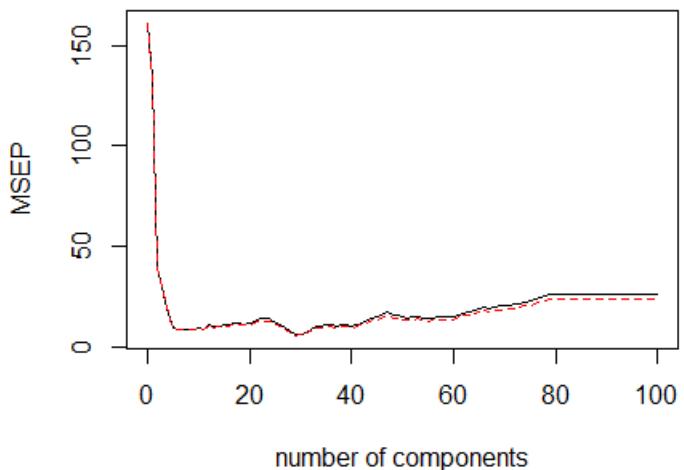
Fat, 6 comps, train



```
> summary(pcr.fit2)
Data: X dimension: 150 100
Y dimension: 150 1
Fit method: svdpc
Number of components considered: 6
TRAINING: % variance explained
      1 comps   2 comps   3 comps   4 comps   5 comps   6 comps
X     98.65    99.59    99.88    99.99    100.00   100.00
Fat   16.79    16.98    56.44    89.97    93.62    95.29
```

# PLS

**Fat**



```
> summary(pls.fit2)
Data: X dimension: 150 100
      Y dimension: 150 1
Fit method: kernelpls
Number of components considered: 6
TRAINING: % variance explained
          1 comps   2 comps   3 comps   4 comps   5 comps   6 comps
X       98.65     98.95    99.75    99.99    100.00    100.00
Fat     17.10     77.67    83.32    90.58    94.93    95.46
```

# Probabilistic models

- Why it is beneficial to assume a **probabilistic** model?
- A common approach to modelling in CS and engineering:

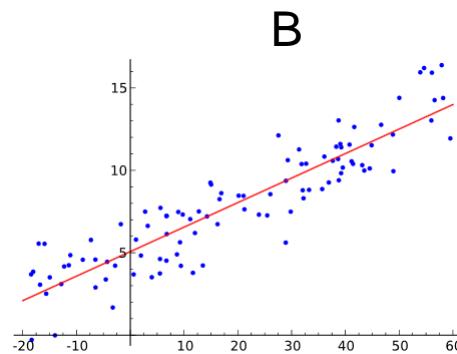
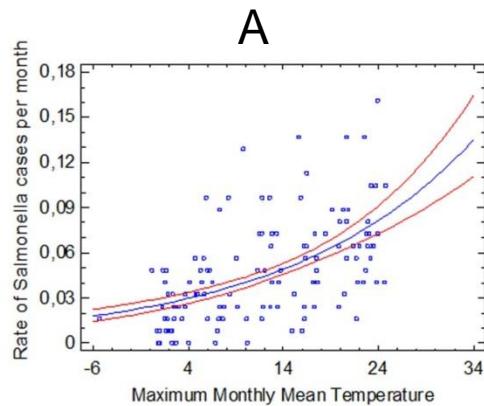
$$y = f(x, w)$$

- $f$  is known,  $w$  is unknown
- Fit model to data with least squares, optimization or ad hoc → find  $w$

# Probabilistic models

## Arguments against deterministic models:

- The model does not really describe actual data (error is not explained)
  - No difference between modelling data A (Poisson) and B (Normal)
  - Estimation strategy for A is not good for B
- The model typically gives a **deterministic answer**, no information about uncertainty
  - "...The exchange rate tomorrow will be 8.22 ..." 😱



# Probabilistic models

## Probabilistic model

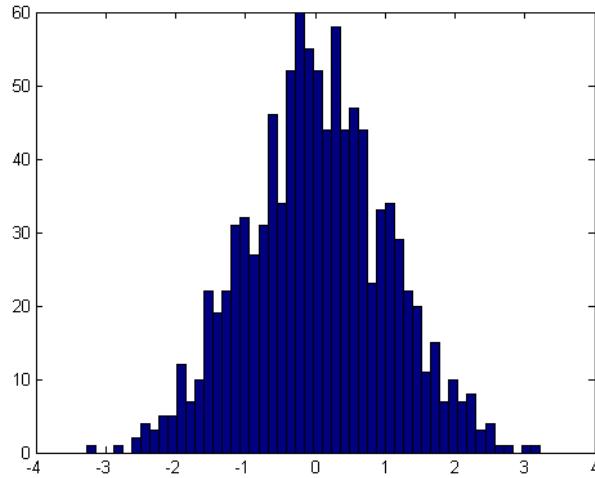
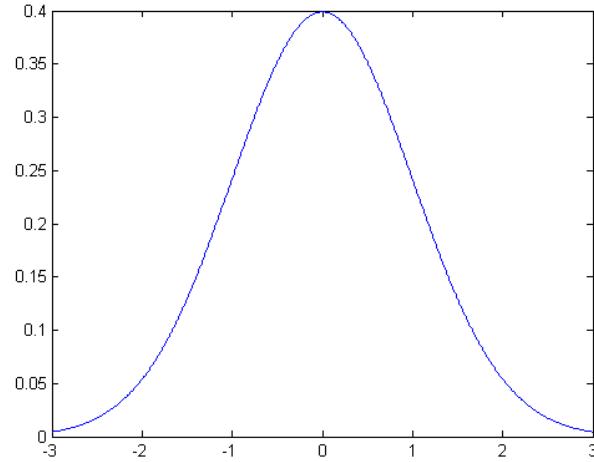
$$Y \sim Distribution(f(x, w), \theta)$$

- Data is fully explained (error as well)
- Automatic principle for finding parameters: MLE , MAP or Bayes theorem
- Automatic principle for finding uncertainty (conf. limits)
  - **Bootstrap**
  - Posterior probability
- Possibility to generate new data of the same type
  - Further testing of the model

# Uncertainty estimation

- Given estimator  $\hat{f} = \hat{f}(x, D)$  (or  $\hat{\alpha} = \delta(D)$ ), how to estimate the uncertainty?
- **Answer 1:** if the distribution for data  $D$  is given, compute analytically the distribution for the estimator → derive confidence limits
  - Often difficult
  - **Example:** In simple linear regression,  $\hat{\alpha}$  follows  $t$  distribution
- **Answer 2: Use bootstrap**

# The bootstrap: general principle

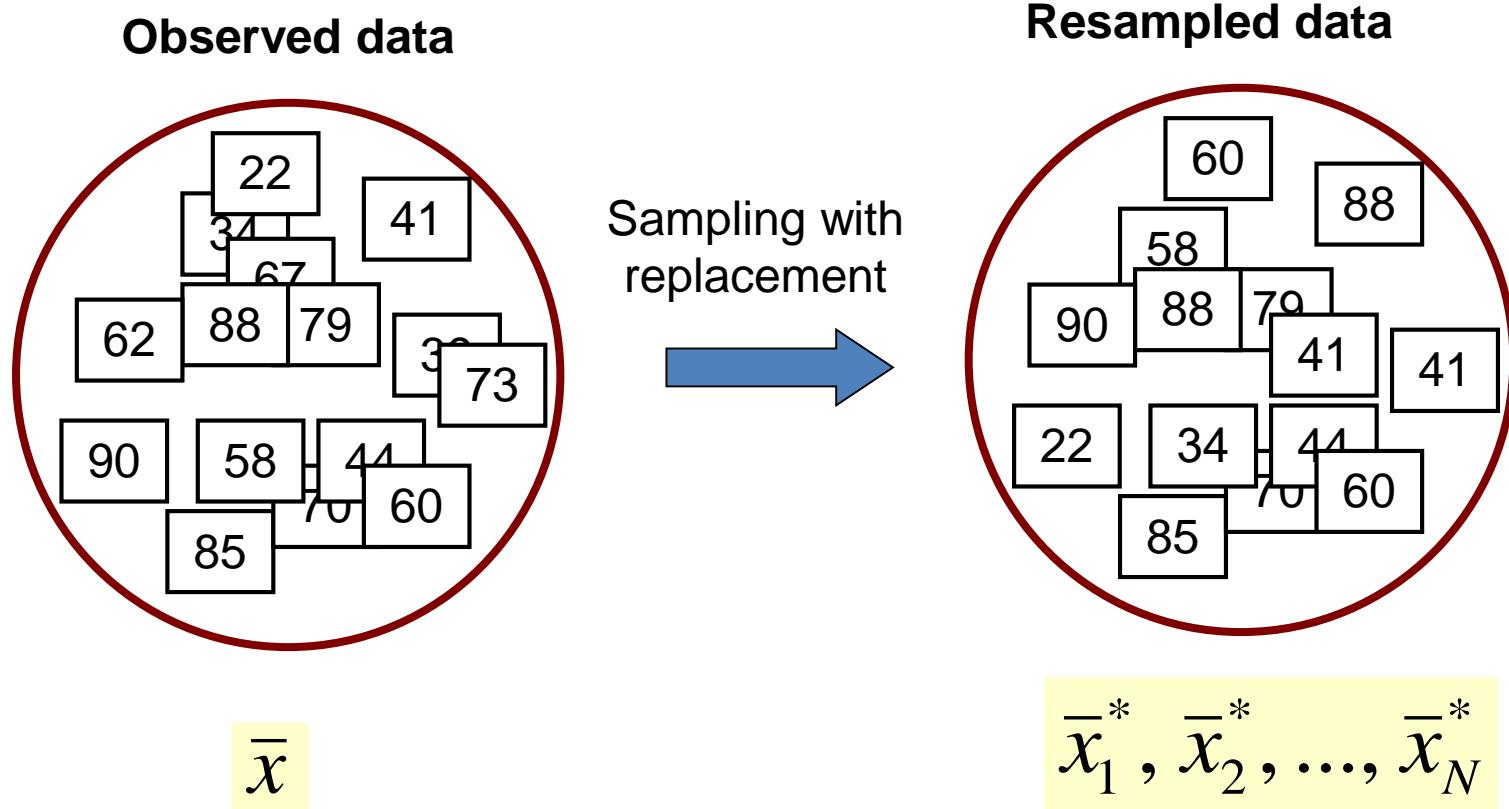


We want to determine uncertainty of  $\hat{f}(D, X)$

1. Generate many different  $D_i$  from their distribution
2. Use histogram of  $\hat{f}(D_i, X)$  to determine confidence limits → unfortunately can not be done (distr of  $D$  is often unknown)

**Instead:** Generate many different  $D_i^*$  from the empirical distribution (histogram)

# Nonparametric bootstrap



# Nonparametric bootstrap

Given estimator  $\hat{w} = \hat{f}(D)$

Assume  $X \sim F(X, w)$ ,  $F$  and  $w$  are unknown

1. Estimate  $\hat{w}$  from data  $D = (X_1, \dots, X_n)$
2. Generate  $D_1 = (X_1^*, \dots, X_n^*)$  by sampling with replacement
3. Repeat step 2  $B$  times
4. The distribution of  $w$  is given by  $\hat{f}(D_1), \dots, \hat{f}(D_B)$

Nonparametric bootstrap can be applied to any deterministic estimator, distribution-free

# Parametric bootstrap

Given estimator  $\hat{w} = \hat{f}(D)$

Assume  $X \sim F(X, w)$ ,  $F$  is known and  $w$  is unknown

1. Estimate  $\hat{w}$  from data  $D = (X_1, \dots, X_n)$
2. Generate  $D_1 = (X_1^*, \dots, X_n^*)$  by generating from  $F(X, \hat{w})$
3. Repeat step 2  $B$  times
4. The distribution of  $w$  is given by  $\hat{f}(D_1), \dots, \hat{f}(D_B)$

Parametric bootstrap is more precise if the distribution form is correct

# Uncertainty estimation

1. Get  $D_1, \dots, D_B$  by bootstrap
  2. Use  $\hat{f}(D_1), \dots, \hat{f}(D_B)$  to estimate the uncertainty
    - Bootstrap percentile
    - Bootstrap Bca
    - ...
- 
- Bootstrap works for all distribution types
  - Can be bad accuracy for small data sets  $n < 40$  (empirical is far from true)
  - Parametric bootstrap works even for small samples

# Bootstrap confidence intervals

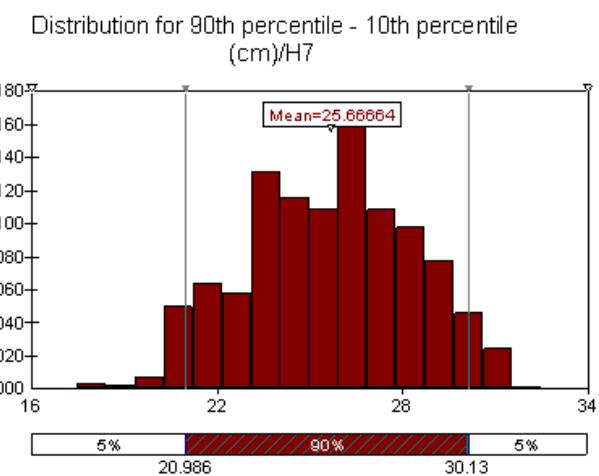
- To estimate  $100(1-\alpha)$  confidence interval for  $w$

## Bootstrap percentile method

1. Using bootstrap, compute  $\hat{f}(D_1), \dots, \hat{f}(D_B)$ , sort in ascending order, get  $w_1 \dots w_B$
2. Define  $A_1 = \text{ceil}(B \alpha/2)$ ,  $A_2 = \text{floor}(B - B \alpha/2)$
3. Confidence interval is given by

$$(w_{A_1}, w_{A_2})$$

Look at the plot...



# Bootstrap: regression context

- Model  $Y \sim F(X, w)$
- Data  $D = \{(Y_i, X_i), i = 1, \dots, n\}$
- Idea: produce several bootstrap sets that are similar to D

Nonparametric bootstrap:

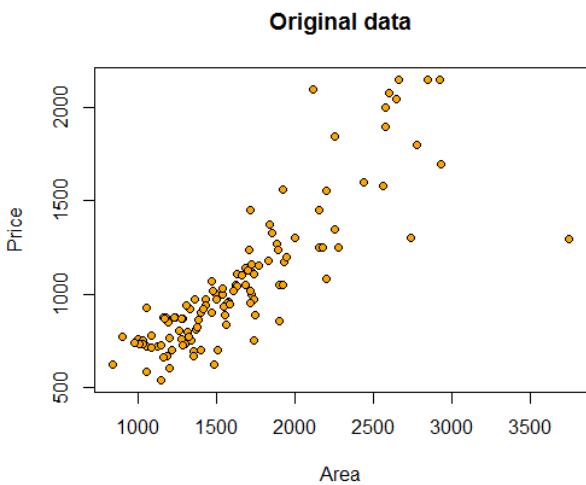
1. Using observation set  $D$ , sample **pairs**  $(X_i, Y_i)$  with replacement and get bootstrap sample  $D_1$
2. Repeat step 1  $B$  times  $\rightarrow$  get  $D_1, \dots, D_B$

# Uncertainty estimation

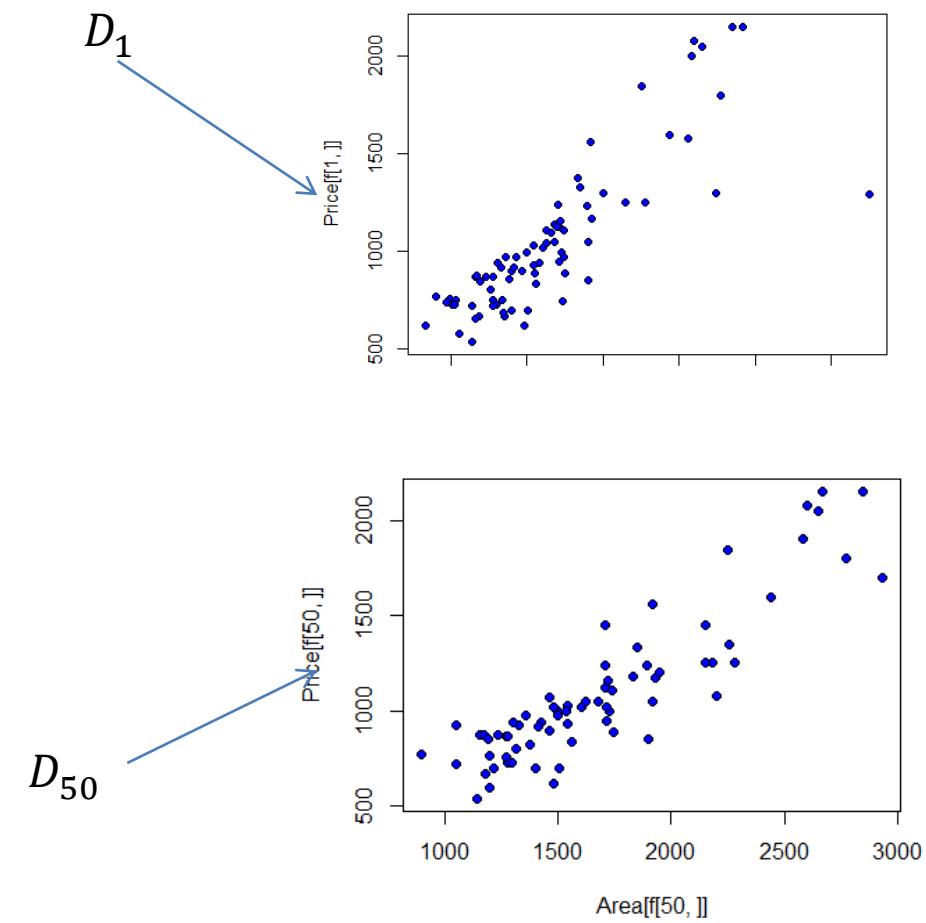
**Example:** Albuquerque dataset:

Y=Price of House

X=Area (sqft)



We sample data index,  
from {1...N}



# Bootstrap: regression context

## Parametric bootstrap

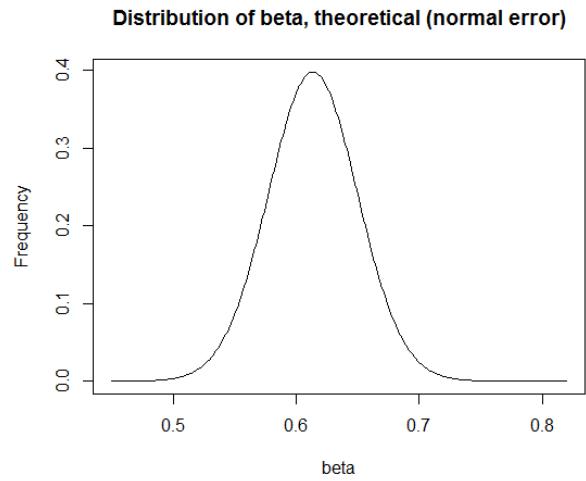
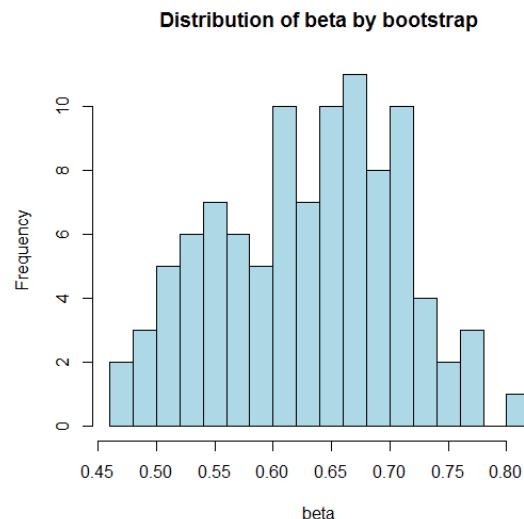
1. Fit a model to  $D \rightarrow$  get  $\hat{w}(D)$ .
2. Set  $X_i^* = X_i$ , generate  $Y_i^* \sim F(X_i, \hat{w})$ .
3.  $D_i = \{(X_i^*, Y_i^*), i = 1, \dots, n\}$
4. Repeat step 2  $B$  times

# Uncertainty estimation

## Albuquerque data

- fit linear regression, estimate uncertainty for  $\beta$ 
  - To think about: do you believe in normality of error here?

$$\hat{\beta} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

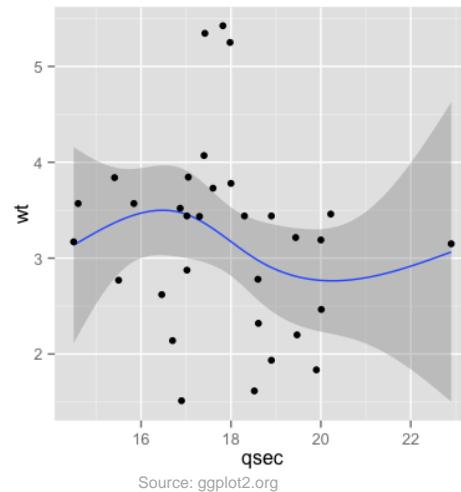


# Confidence intervals in regression

- Given  $Y \sim \text{Distribution}(y|x, w)$ ,  $EY|X = \mu|x = f(x, w)$ 
  - Example:**  $Y \sim N(w^T x, \sigma^2)$ ,  $\mu|x = f(x, w) = w^T x$
- Estimate intervals for  $\mu|x = f(x, w)$  for many X, combine in a **confidence band**
- What is estimator?
  - $\mu|x = f(x, w)$

## Estimation

- Compute  $D_1, \dots, D_B$  using a bootstrap
- Fit model to  $D_1, \dots, D_B \rightarrow$  estimate  $\hat{w}_1, \dots, \hat{w}_B$
- For a given X, compute  $f(X, \hat{w}_1), \dots, f(X, \hat{w}_B)$  and estimate confidence interval by (percentile method)
- Combine confidence intervals in a band



Source: ggplot2.org

# Bootstrap: R

- Package **boot**

- Functions:

- boot()
    - boot.ci() – 1 parameter
    - envelope() – many parameters

- Random random generation for parametric bootstrap:

- Rnorm()
  - Runif()
  - ...

```
boot(data, statistic, R, sim = "ordinary",
      ran.gen = function(d, p) d, mle = NULL,...)
```

### Nonparametric bootstrap:

- Write a function *statistic* that depends on *dataframe* and *index* and returns the estimator

```
library(boot)
data2=data[order(data$Area), ]#reordering data according
to Area

# computing bootstrap samples
f=function(data, ind){
  data1=data[ind, ]# extract bootstrap sample
  res=lm(Price~Area, data=data1) #fit linear model
  #predict values for all Area values from the original
  data
  priceP=predict(res,newdata=data2)
  return(priceP)
}
res=boot(data2, f, R=1000) #make bootstrap
```

# Bootstrap: R

## Parametric bootstrap:

- Compute value  $mle$  that estimates model parameters from the data
- Write function  $ran.gen$  that depends on  $data$  and  $mle$  and which generates new data
- Write function  $statistic$  that depend on  $data$  which will be generated by  $ran.gen$  and should return the estimator

```
mle=lm(Price~Area, data=data2)

rng=function(data, mle) {
  data1=data.frame(Price=data$Price,
  Area=data$Area)
  n=length(data$Price)
  #generate new Price
  data1$Price=rnorm(n,predict(mle,
  newdata=data1),sd(mle$residuals))
  return(data1)
}

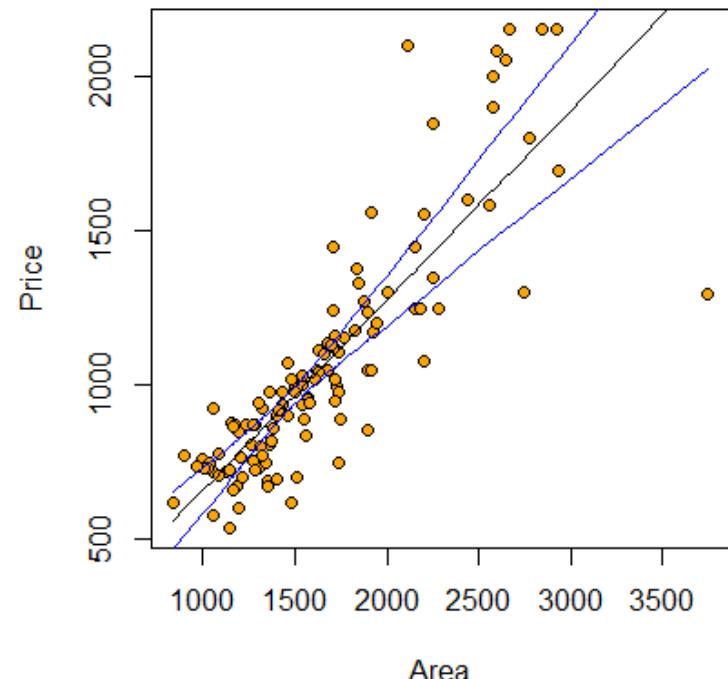
f1=function(data1){
  res=lm(Price~Area, data=data1) #fit linear
  #predict values for all Area values from
  #the original data
  priceP=predict(res,newdata=data2)
  return(priceP)
}

res=boot(data2, statistic=f1, R=1000,
mle=mle,ran.gen=rng, sim="parametric")
```

# Uncertainty estimation: R

- Bootstrap confidence bands for linear model

```
e=envelope(res) #compute confidence bands  
  
fit=lm(Price~Area, data=data2)  
priceP=predict(fit)  
  
plot(Area, Price, pch=21, bg="orange")  
points(data2$Area,priceP,type="l") #plot fitted line  
  
#plot confidence bands  
points(data2$Area,e$point[2,], type="l", col="blue")  
points(data2$Area,e$point[1,], type="l", col="blue")
```



# Prediction bands

- Confidence interval for  $Y|X =$  interval for mean  $EY|X$
- Prediction interval for  $Y|X =$  interval for  $Y|X$

$$Y \sim Distribution(x, w)$$

## Prediction band for parametric bootstrap

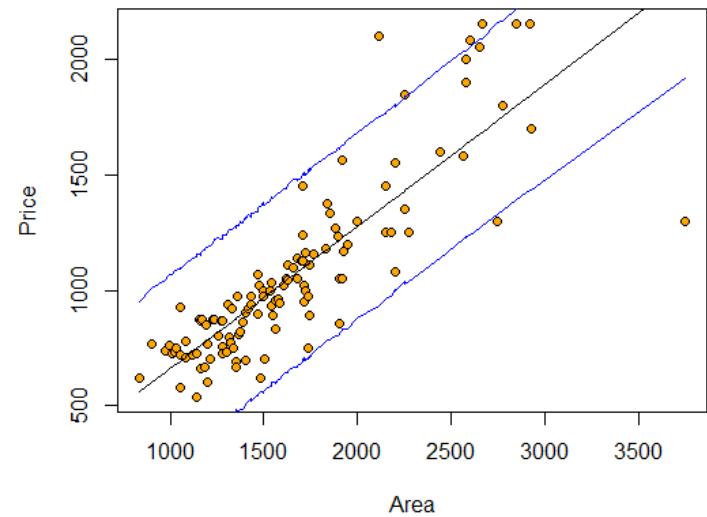
1. Run parametric bootstrap and get  $D_1, \dots, D_B$
2. Fit the model to the data and get  $\hat{w}(D_1), \dots, \hat{w}(D_B)$
3. For each  $X$ , generate from  $Distribution(X, \hat{w}(D_1)), \dots, Distribution(X, \hat{w}(D_B))$  and apply percentile method
4. Connect the intervals → get the band

# Estimation of the model quality

## Example: parametric bootstrap

```
mle=lm(Price~Area, data=data2)

f1=function(data1){
  res=lm(Price~Area, data=data1) #fit linear
  model
  #predict values for all Area values from the
  original data
  priceP=predict(res,newdata=data2)
  n=length(data2$Price)
  predictedP=rnorm(n,priceP, sd(mle$residuals))
  return(predictedP)
}
res=boot(data2, statistic=f1, R=10000,
mle=mle,ran.gen=rng, sim="parametric")
```



Why wider band?

# 732A95 Introduction to Machine Learning

## Lecture 5a: Kernel Methods

Jose M. Peña  
IDA, Linköping University, Sweden

- ▶ Histogram, Moving Window, and Kernel Classification
- ▶ Histogram, Moving Window, and Kernel Regression
- ▶ Histogram, Moving Window, and Kernel Density Estimation
- ▶ Kernel Selection
- ▶ Kernel Trick
- ▶ Summary

## Histogram Classification

- ▶ Consider binary classification with input space  $\mathbb{R}^D$ .
- ▶ The best classifier under the 0-1 loss function is  $y^*(\mathbf{x}) = \arg \max_y p(y|\mathbf{x})$ .
- ▶ Since  $\mathbf{x}$  may not appear in the finite training set  $\{(\mathbf{x}_n, t_n)\}$  available, then
  - ▶ divide the input space into  $D$ -dimensional cubes of side  $h$ , and
  - ▶ classify according to majority vote in the cube  $C(\mathbf{x}, h)$  that contains  $\mathbf{x}$ .

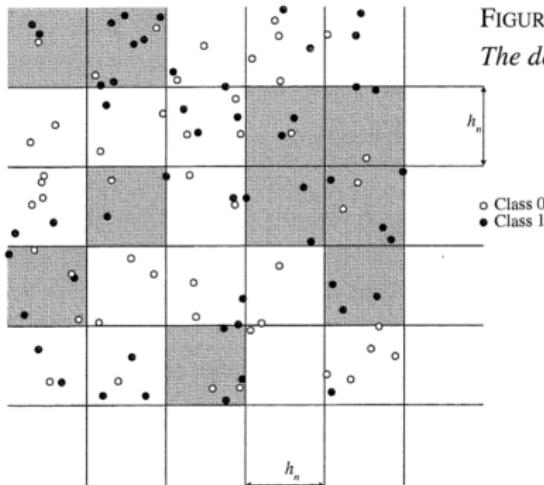


FIGURE 6.1. A cubic histogram rule:  
The decision is 1 in the shaded area.

- ▶ In other words,

$$y_C(\mathbf{x}) = \begin{cases} 0 & \text{if } \sum_n \mathbf{1}_{\{t_n=1, \mathbf{x}_n \in C(\mathbf{x}, h)\}} \leq \sum_n \mathbf{1}_{\{t_n=0, \mathbf{x}_n \in C(\mathbf{x}, h)\}} \\ 1 & \text{otherwise} \end{cases}$$

## Moving Window Classification

- The histogram rule is less accurate at the borders of the cube, because those points are not as well represented by the cube as the ones near the center. Then,
  - consider the points within a certain distance to the point to classify, and
  - classify the point according to majority vote.

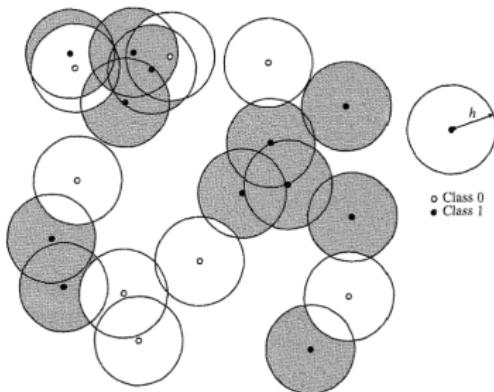


FIGURE 10.1. *The moving window rule in  $\mathbb{R}^2$ . The decision is 1 in the shaded area.*

- In other words,

$$y_S(\mathbf{x}) = \begin{cases} 0 & \text{if } \sum_n \mathbf{1}_{\{t_n=1, \mathbf{x}_n \in S(\mathbf{x}, h)\}} \leq \sum_n \mathbf{1}_{\{t_n=0, \mathbf{x}_n \in S(\mathbf{x}, h)\}} \\ 1 & \text{otherwise} \end{cases}$$

where  $S(\mathbf{x}, h)$  is a  $D$ -dimensional closed ball of radius  $h$  centered at  $\mathbf{x}$ .

## Kernel Classification

- The moving window rule gives equal weight to all the points in the ball, which may be counterintuitive. Then,

$$y_k(\mathbf{x}) = \begin{cases} 0 & \text{if } \sum_n \mathbf{1}_{\{t_n=1\}} k\left(\frac{\mathbf{x}-\mathbf{x}_n}{h}\right) \leq \sum_n \mathbf{1}_{\{t_n=0\}} k\left(\frac{\mathbf{x}-\mathbf{x}_n}{h}\right) \\ 1 & \text{otherwise} \end{cases}$$

where  $k : \mathbb{R}^D \rightarrow \mathbb{R}$  is a kernel function, which is usually non-negative and monotone decreasing along rays starting from the origin. The parameter  $h$  is called smoothing factor or width.

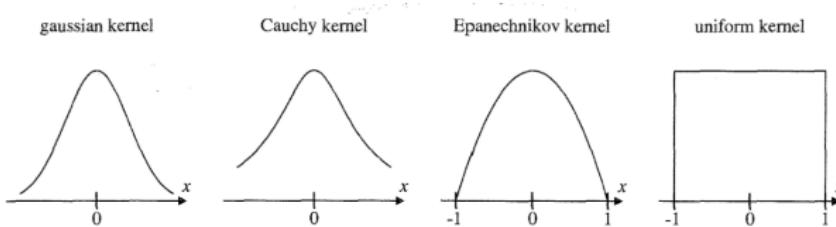


FIGURE 10.3. Various kernels on  $\mathcal{R}$ .

- Gaussian kernel:  $k(u) = \exp(-\|u\|^2)$  where  $\|\cdot\|$  is the Euclidean distance.
- Cauchy kernel:  $k(u) = 1/(1 + \|u\|^{D+1})$
- Epanechnikov kernel:  $k(u) = (1 - \|u\|^2)\mathbf{1}_{\{\|u\| \leq 1\}}$
- Moving window kernel:  $k(u) = \mathbf{1}_{\{u \in S(0,1)\}}$

## Kernel Classification

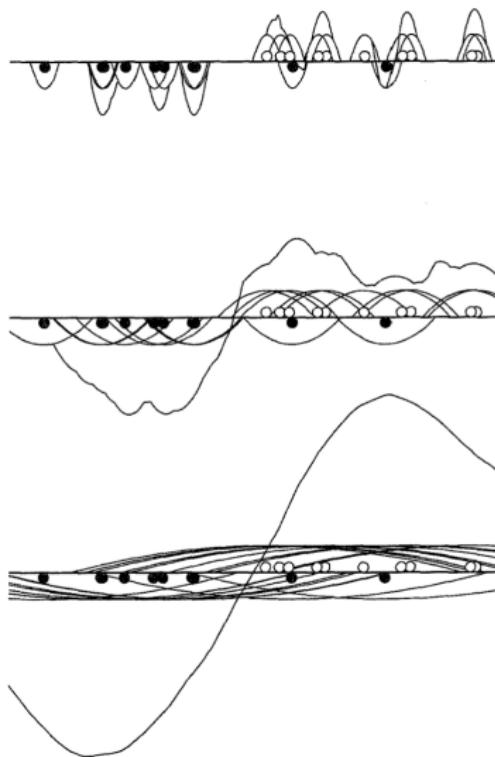


FIGURE 10.2. Kernel rule on the real line. The figure shows  $\sum_{i=1}^n (2Y_i - 1)K((x - X_i)/h)$  for  $n = 20$ ,  $K(u) = (1 - u^2)I_{\{|u| \leq 1\}}$  (the Epanechnikov kernel), and three smoothing factors  $h$ . One definitely undersmooths and one oversmooths. We took  $p = 1/2$ , and the class-conditional densities are  $f_0(x) = 2(1 - x)$  and  $f_1(x) = 2x$  on  $[0, 1]$ .

## Histogram, Moving Window, and Kernel Regression

- ▶ Consider regressing an unidimensional continuous random variable on a  $D$ -dimensional continuous random variable.
- ▶ The best regression function under the squared error loss function is  $y^*(\mathbf{x}) = \mathbb{E}_Y[y|\mathbf{x}]$ .
- ▶ Since  $\mathbf{x}$  may not appear in the finite training set  $\{(\mathbf{x}_n, t_n)\}$  available, then we average over the points in  $C(\mathbf{x}, h)$  or  $S(\mathbf{x}, h)$ , or kernel-weighted average over all the points.
- ▶ In other words,

$$y_C(\mathbf{x}) = \frac{\sum_{\mathbf{x}_n \in C(\mathbf{x}, h)} t_n}{|\{x_n \in C(x, h)\}|}$$

or

$$y_S(\mathbf{x}) = \frac{\sum_{\mathbf{x}_n \in S(\mathbf{x}, h)} t_n}{|\{x_n \in S(x, h)\}|}$$

or

$$y_k(\mathbf{x}) = \frac{\sum_n k\left(\frac{\mathbf{x} - \mathbf{x}_n}{h}\right) t_n}{\sum_n k\left(\frac{\mathbf{x} - \mathbf{x}_n}{h}\right)}$$

## Histogram, Moving Window, and Kernel Density Estimation

- ▶ Consider density estimation for a  $D$ -dimensional random variable.
- ▶ Let  $R \subseteq \mathbb{R}^D$  and  $\mathbf{x} \in R$ . Then,

$$P = \int_R p(\mathbf{x}) d\mathbf{x} \simeq p(\mathbf{x}) \text{Volume}(R)$$

and the number of the  $N$  training points  $\{\mathbf{x}_n\}$  that fall inside  $R$  is

$$|\{\mathbf{x}_n \in R\}| \simeq P N$$

and thus

$$p(\mathbf{x}) \simeq \frac{|\{\mathbf{x}_n \in R\}|}{N \text{Volume}(R)}$$

- ▶ Then,

$$p_C(\mathbf{x}) = \frac{|\{\mathbf{x}_n \in C(\mathbf{x}, h)\}|}{N \text{Volume}(C(\mathbf{x}, h))}$$

or

$$p_S(\mathbf{x}) = \frac{|\{\mathbf{x}_n \in S(\mathbf{x}, h)\}|}{N \text{Volume}(S(\mathbf{x}, h))}$$

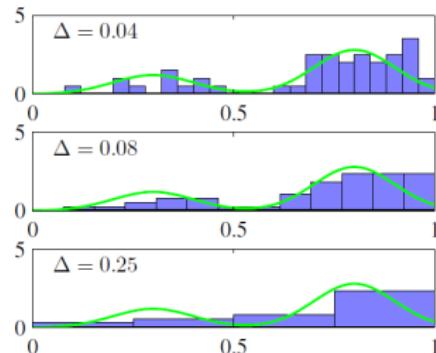
or

$$p_k(\mathbf{x}) = \frac{1}{N} \sum_n k\left(\frac{\mathbf{x} - \mathbf{x}_n}{h}\right)$$

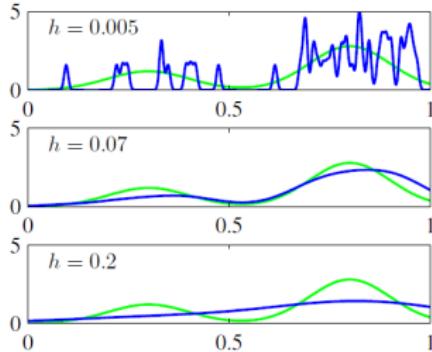
assuming that  $k(u) \geq 0$  for all  $u$  and  $\int k(u) du = 1$ .

# Histogram, Moving Window, and Kernel Density Estimation

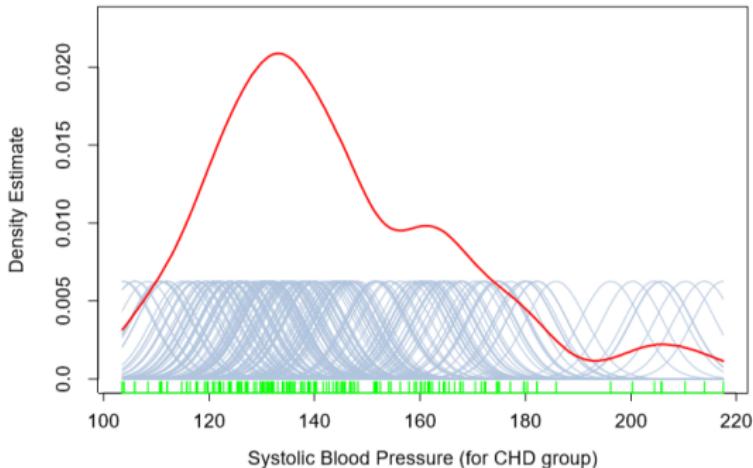
**Figure 2.24** An illustration of the histogram approach to density estimation, in which a data set of 50 data points is generated from the distribution shown by the green curve. Histogram density estimates, based on (2.241), with a common bin width  $\Delta$  are shown for various values of  $\Delta$ .



**Figure 2.25** Illustration of the kernel density model (2.250) applied to the same data set used to demonstrate the histogram approach in Figure 2.24. We see that  $h$  acts as a smoothing parameter and that if it is set too small (top panel), the result is a very noisy density model, whereas if it is set too large (bottom panel), then the bimodal nature of the underlying distribution from which the data is generated (shown by the green curve) is washed out. The best density model is obtained for some intermediate value of  $h$  (middle panel).



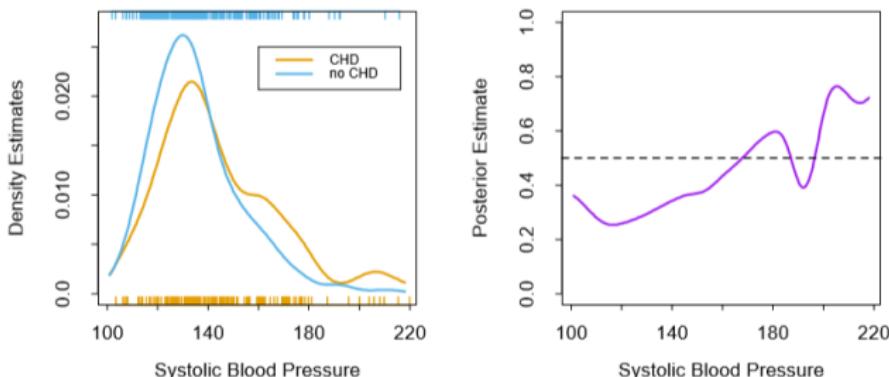
## Histogram, Moving Window, and Kernel Density Estimation



**FIGURE 6.13.** A kernel density estimate for systolic blood pressure (for the CHD group). The density estimate at each point is the average contribution from each of the kernels at that point. We have scaled the kernels down by a factor of 10 to make the graph readable.

- ▶ From kernel density estimation to kernel classification:
  1. Estimate  $p(\mathbf{x}|y = 0)$  and  $p(\mathbf{x}|y = 1)$  using the methods just seen.
  2. Estimate  $p(y)$  as class proportions.
  3. Compute  $p(y|\mathbf{x}) \propto p(\mathbf{x}|y)p(y)$  by Bayes theorem.

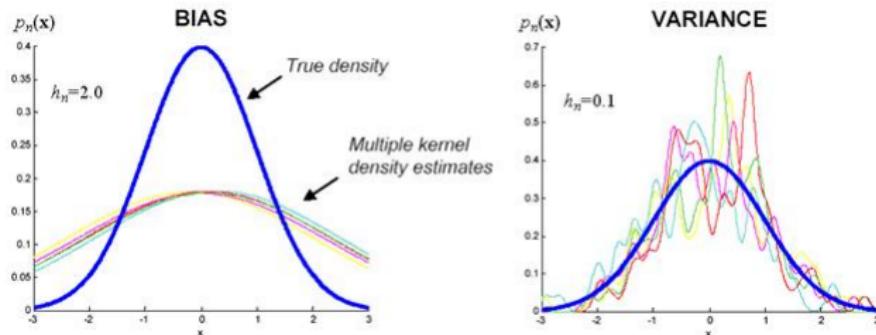
# Histogram, Moving Window, and Kernel Density Estimation



**FIGURE 6.14.** The left panel shows the two separate density estimates for systolic blood pressure in the CHD versus no-CHD groups, using a Gaussian kernel density estimate in each. The right panel shows the estimated posterior probabilities for CHD, using (6.25).

## Kernel Selection

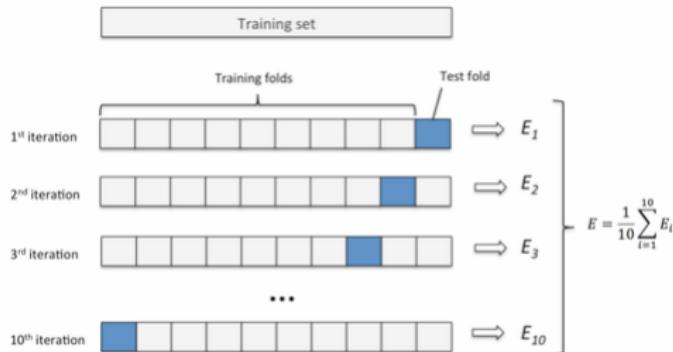
- ▶ How to choose the right kernel and width ? E.g., by cross-validation.
- ▶ What does “right” mean ? E.g., minimize loss function.
- ▶ Note that the width of the kernel corresponds to a bias-variance trade-off.



- ▶ Small width implies considering few points. So, the variance will be large (similar to the variance of a single point). The bias will be small since the points considered are close to  $x$ .
- ▶ Large width implies considering many points. So, the variance will be small and the bias will be large.

## Kernel Selection

- Recall the following from previous lectures.
- Cross-validation is a technique to estimate the prediction error of a model.

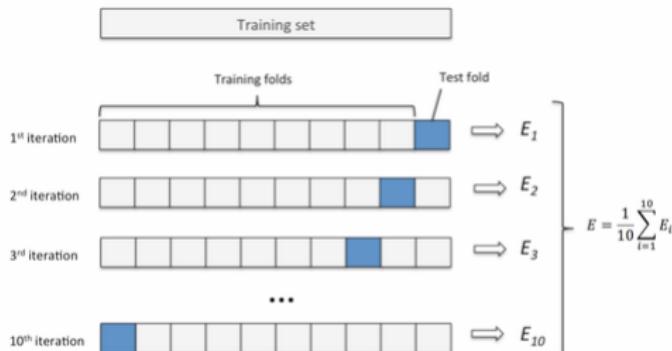


- If the training set contains  $N$  points, note that cross-validation estimates the prediction error when the model is trained on  $N - N/K$  points.
- Note that the model returned is trained on  $N$  points. So, cross-validation overestimates the prediction error of the model returned.
- This seems to suggest that a large  $K$  should be preferred. However, this typically implies a large variance of the error estimate, since there are only  $N/K$  test points.
- Typically,  $K = 5, 10$  works well.

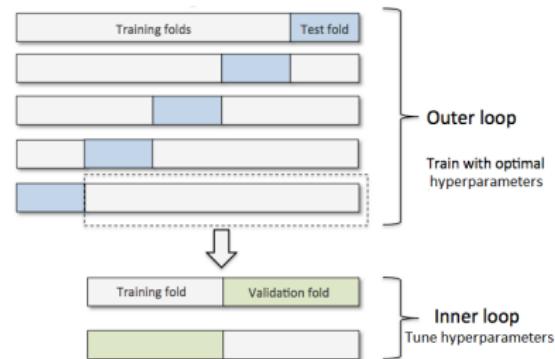
## Kernel Selection

- ▶ Model: For example, ridge regression with a given value for the penalty factor  $\lambda$ . Only the parameters (weights) need to be determined (closed-form solution).
- ▶ Model selection: For example, determine the value for the penalty factor  $\lambda$ . Another example, determine the kernel and width for kernel classification, regression or density estimation. In either case, we do not have a continuous criterion to optimize. Solution: **Nested cross-validation**.

Cross-validation for estimating model prediction error



**Nested** cross-validation for estimating model **selection** prediction error



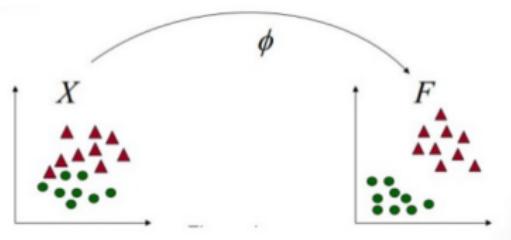
- ▶ Error overestimation may not be a concern for model selection. So,  $K = 2$  may suffice in the inner loop.
- ▶ Which is the fitted model returned by nested cross-validation ?

## Kernel Trick

- ▶ The kernel function  $k\left(\frac{\mathbf{x}-\mathbf{x}'}{h}\right)$  is invariant to translations, and it can be generalized as  $k(\mathbf{x}, \mathbf{x}')$ . For instance,
  - ▶ Polynomial kernel:  $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x} + c)^M$
  - ▶ Gaussian kernel:  $k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2 / 2\sigma^2)$
- ▶ If the matrix

$$\begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \dots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & \dots & k(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix}$$

is symmetric and positive semi-definite for all choices of  $\{\mathbf{x}_n\}$ , then  $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$  where  $\phi(\cdot)$  is a mapping from the input space to the feature space.



- ▶ The feature space may be non-linear and even infinite dimensional. For instance,

$$\phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2c}x_1, \sqrt{2c}x_2, c)$$

for the polynomial kernel with  $M = D = 2$ .

## Kernel Trick

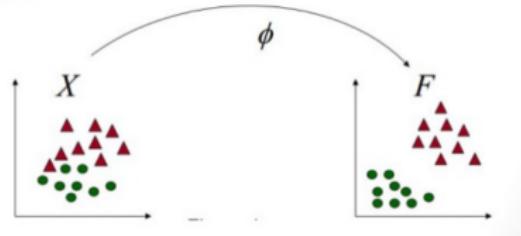
- ▶ Consider again moving window classification, regression, and density estimation.
- ▶ Note that  $\mathbf{x}_n \in S(\mathbf{x}, h)$  if and only if  $\|\mathbf{x} - \mathbf{x}_n\| \leq h$ .
- ▶ Note that

$$\|\mathbf{x} - \mathbf{x}_n\| = (\mathbf{x} - \mathbf{x}_n)^T (\mathbf{x} - \mathbf{x}_n) = \mathbf{x}^T \mathbf{x} + \mathbf{x}_n^T \mathbf{x}_n - 2\mathbf{x}^T \mathbf{x}_n$$

- ▶ Then,

$$\begin{aligned}\|\phi(\mathbf{x}) - \phi(\mathbf{x}_n)\| &= \phi(\mathbf{x}^T) \phi(\mathbf{x}) + \phi(\mathbf{x}_n^T) \phi(\mathbf{x}_n) - 2\phi(\mathbf{x}^T) \phi(\mathbf{x}_n) \\ &= k(\mathbf{x}, \mathbf{x}) + k(\mathbf{x}_n, \mathbf{x}_n) - 2k(\mathbf{x}, \mathbf{x}_n)\end{aligned}$$

- ▶ So, the distance is now computed in a (hopefully) more convenient space.



- ▶ Note that we do not need to compute  $\phi(\mathbf{x})$  and  $\phi(\mathbf{x}_n)$ .

# Kernel Trick

- ▶ Two alternatives for building  $k(\mathbf{x}, \mathbf{x}')$ :
  - ▶ Choose a convenient  $\phi(\mathbf{x})$  and let  $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$ .
  - ▶ Build it from existing kernel functions as follows.

## Techniques for Constructing New Kernels.

Given valid kernels  $k_1(\mathbf{x}, \mathbf{x}')$  and  $k_2(\mathbf{x}, \mathbf{x}')$ , the following new kernels will also be valid:

$$k(\mathbf{x}, \mathbf{x}') = ck_1(\mathbf{x}, \mathbf{x}') \quad (6.13)$$

$$k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}') \quad (6.14)$$

$$k(\mathbf{x}, \mathbf{x}') = q(k_1(\mathbf{x}, \mathbf{x}')) \quad (6.15)$$

$$k(\mathbf{x}, \mathbf{x}') = \exp(k_1(\mathbf{x}, \mathbf{x}')) \quad (6.16)$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \quad (6.17)$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}') \quad (6.18)$$

$$k(\mathbf{x}, \mathbf{x}') = k_3(\phi(\mathbf{x}), \phi(\mathbf{x}')) \quad (6.19)$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{A} \mathbf{x}' \quad (6.20)$$

$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) \quad (6.21)$$

$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a)k_b(\mathbf{x}_b, \mathbf{x}'_b) \quad (6.22)$$

where  $c > 0$  is a constant,  $f(\cdot)$  is any function,  $q(\cdot)$  is a polynomial with nonnegative coefficients,  $\phi(\mathbf{x})$  is a function from  $\mathbf{x}$  to  $\mathbb{R}^M$ ,  $k_3(\cdot, \cdot)$  is a valid kernel in  $\mathbb{R}^M$ ,  $\mathbf{A}$  is a symmetric positive semidefinite matrix,  $\mathbf{x}_a$  and  $\mathbf{x}_b$  are variables (not necessarily disjoint) with  $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$ , and  $k_a$  and  $k_b$  are valid kernel functions over their respective spaces.

# 732A95 Introduction to Machine Learning

## Lecture 5b: Support Vector Machines

Jose M. Peña  
IDA, Linköping University, Sweden

DRAFT

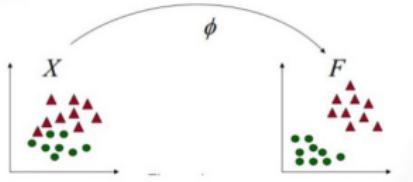
## Support Vector Machines for Classification

- ▶ Consider binary classification with input space  $\mathbb{R}^D$ .
- ▶ Consider a training set  $\{(x_n, t_n)\}$  where  $t_n \in \{-1, +1\}$ .
- ▶ Consider using the linear model

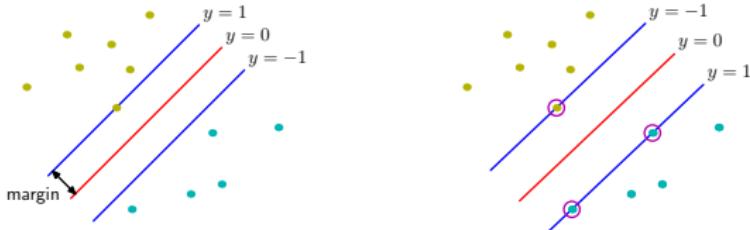
$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b$$

so that a new point  $\mathbf{x}$  is classified according to the sign of  $y(\mathbf{x})$ .

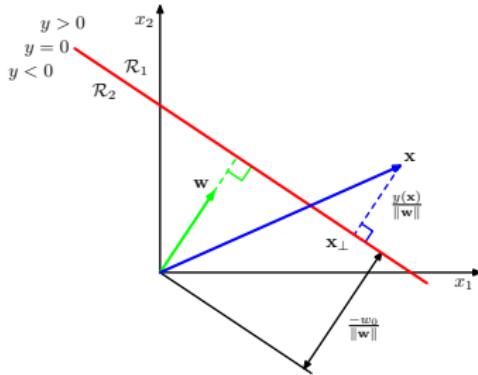
- ▶ Assume that the training set is linearly separable in the feature space (but not necessarily in the input space), i.e.  $t_n y(\mathbf{x}_n) > 0$  for all  $n$ .



- ▶ Aim for the separating hyperplane that maximizes the margin (i.e. the smallest perpendicular distance from any point to the hyperplane) so as to minimize the generalization error.



## Support Vector Machines for Classification



- ▶ The perpendicular distance from any point to the hyperplane is given by

$$\frac{t_n y(\mathbf{x}_n)}{\|\mathbf{w}\|} = \frac{t_n (\mathbf{w}^T \phi(\mathbf{x}_n) + b)}{\|\mathbf{w}\|}$$

- ▶ Then, the maximum margin separating hyperplane is given by

$$\arg \max_{\mathbf{w}, b} \left( \frac{1}{\|\mathbf{w}\|} \min_n (t_n (\mathbf{w}^T \phi(\mathbf{x}_n) + b)) \right)$$

- ▶ Multiply  $\mathbf{w}$  and  $b$  by  $\kappa$  so that  $t_n (\mathbf{w}^T \phi(\mathbf{x}_n) + b) = 1$  for the point closest to the hyperplane. Note that  $t_n (\mathbf{w}^T \phi(\mathbf{x}_n) + b) / \|\mathbf{w}\|$  does not change.

## Support Vector Machines for Classification

- Then, the maximum margin separating hyperplane is given by

$$\arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$$

subject to  $t_n(\mathbf{w}^T \phi(\mathbf{x}_n) + b) \geq 1$  for all  $n$ .

- To minimize the previous expression, we minimize

$$\frac{1}{2} \|\mathbf{w}\|^2 - \sum_n a_n (t_n(\mathbf{w}^T \phi(\mathbf{x}_n) + b) - 1)$$

where  $a_n \geq 0$  are called Lagrange multipliers. Any stationary point of the Lagrangian function is a stationary point of the original function subject to the constraints. Moreover, the original function is concave up and, thus, so is the Lagrangian function. Then, it is "easy" to minimize.

- Setting its derivatives with respect to  $\mathbf{w}$  and  $b$  to zero gives

$$\mathbf{w} = \sum_n a_n t_n \phi(\mathbf{x}_n)$$

$$0 = \sum_n a_n t_n$$

- Replacing these in the Lagrangian function gives the dual representation

$$\sum_n a_n - \frac{1}{2} \sum_n \sum_m a_n a_m t_n t_m \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m) = \sum_n a_n - \frac{1}{2} \sum_n \sum_m a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

subject to  $a_n \geq 0$  for all  $n$ , and  $\sum_n a_n t_n = 0$ .

## Support Vector Machines for Classification

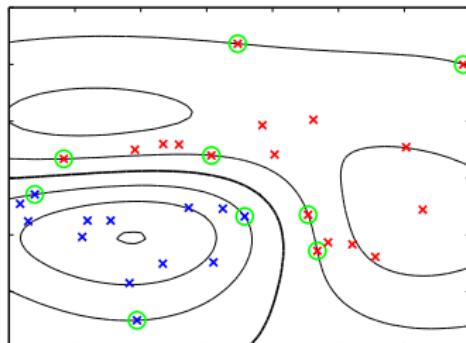
- When maximizing the Lagrangian function, the Karush-Kuhn-Tucker condition holds for all  $n$ :

$$a_n(t_n y(\mathbf{x}_n) - 1) = 0$$

- Then,  $a_n > 0$  if and only if  $t_n y(\mathbf{x}_n) = 1$ . The points with  $a_n > 0$  are called support vectors and they lie on the margin boundaries.
- A new point  $\mathbf{x}$  is classified according to the sign of

$$y(\mathbf{x}) = \sum_n a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b = \sum_{m \in S} a_m t_m k(\mathbf{x}, \mathbf{x}_m) + b$$

where  $S$  are the indexes of the support vectors.



## Support Vector Machines for Classification

- To find  $b$ , consider any support vector  $\mathbf{x}_n$ . Then,

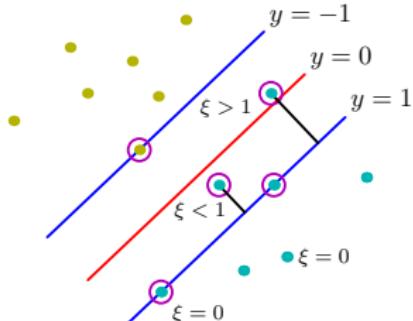
$$1 = t_n y(\mathbf{x}_n) = t_n \left( \sum_{m \in S} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) + b \right)$$

and thus

$$b = t_n - \sum_{m \in S} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

- We now drop the assumption of linear separability in the feature space, e.g. to avoid overfitting. We do so by introducing the slack variables  $\xi_n \geq 0$  to penalize (almost-)misclassified points as

$$\xi_n = \begin{cases} 0 & \text{if } t_n y(\mathbf{x}_n) \geq 1 \\ |t_n - y(\mathbf{x}_n)| & \text{otherwise} \end{cases}$$

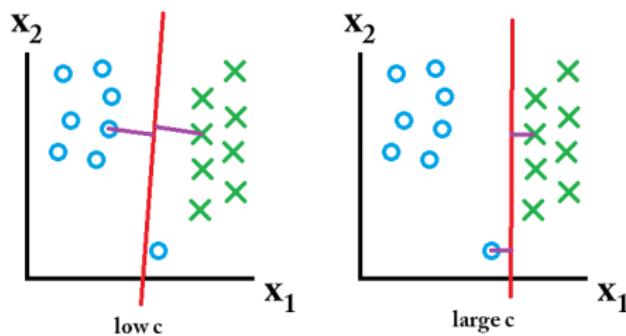


## Support Vector Machines for Classification

- The optimal separating hyperplane is given by

$$\arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_n \xi_n$$

subject to  $t_n y(\mathbf{x}_n) \geq 1 - \xi_n$  and  $\xi_n \geq 0$  for all  $n$ , and where  $C > 0$  controls regularization. Its value can be decided by cross-validation. Note that the number of misclassified points is upper bounded by  $\sum_n \xi_n$ .



- To minimize the previous expression, we minimize

$$\frac{1}{2} \|\mathbf{w}\|^2 + C \sum_n \xi_n - \sum_n a_n (t_n (\mathbf{w}^T \phi(\mathbf{x}_n) + b) - 1 + \xi_n) - \sum_n \mu_n \xi_n$$

where  $a_n \geq 0$  and  $\mu_n \geq 0$  are Lagrange multipliers.

## Support Vector Machines for Classification

- ▶ Setting its derivatives with respect to  $\mathbf{w}$ ,  $b$  and  $\xi_n$  to zero gives

$$\mathbf{w} = \sum_n a_n t_n \phi(\mathbf{x}_n)$$

$$0 = \sum_n a_n t_n$$

$$a_n = C - \mu_n$$

- ▶ Replacing these in the Lagrangian function gives the dual representation

$$\sum_n a_n - \frac{1}{2} \sum_n \sum_m a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

subject to  $a_n \geq 0$  and  $a_n \leq C$  for all  $n$ , because  $\mu_n \geq 0$ .

- ▶ When maximizing the Lagrangian function, the Karush-Kuhn-Tucker conditions hold for all  $n$ :

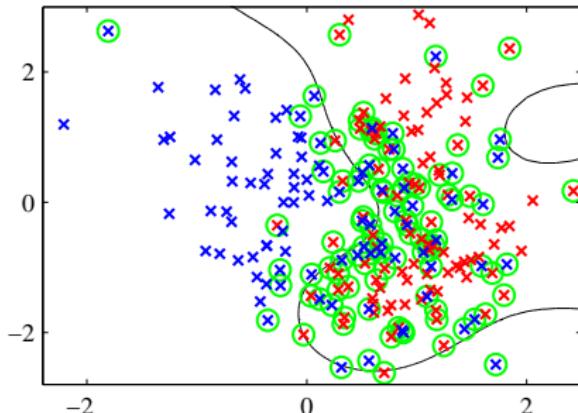
$$a_n(t_n y(\mathbf{x}_n) - 1 + \xi_n) = 0$$

$$\mu_n \xi_n = 0$$

- ▶ Then,  $a_n > 0$  if and only if  $t_n y(\mathbf{x}_n) = 1 - \xi_n$  for all  $n$ . The points with  $a_n > 0$  are called support vectors and they lie
  - ▶ on the margin if  $a_n < C$ , because then  $\mu_n > 0$  and thus  $\xi_n = 0$ , or
  - ▶ inside the margin (even on the wrong side of the decision boundary) if  $a_n = C$ , because then  $\mu_n = 0$  and thus  $\xi_n$  is unconstrained.

## Support Vector Machines for Classification

- Since the optimal  $\mathbf{w}$  takes the same form as in the linearly separable case, classifying a new point is done the same as before. Finding  $b$  is done the same as before by considering any support vector  $\mathbf{x}_n$  with  $0 < a_n < C$ .



- Not covered topics:
  - Classifying into more than two classes.
  - Returning class posterior probabilities.

## Support Vector Machines for Regression

- ▶ Consider regressing an unidimensional continuous random variable on a  $D$ -dimensional continuous random variable.
- ▶ Consider a training set  $\{(\mathbf{x}_n, t_n)\}$ . Consider using the linear model

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b$$

- ▶ Instead of minimizing the classical regularized error function

$$\frac{1}{2} \sum_n (y(\mathbf{x}_n) - t_n)^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

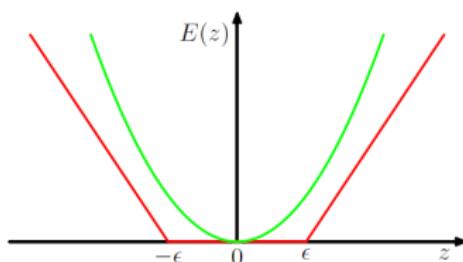
consider minimizing the  $\epsilon$ -insensitive regularized error function

$$C \sum_n E_\epsilon(y(\mathbf{x}_n) - t_n) + \frac{1}{2} \|\mathbf{w}\|^2$$

where  $C > 0$  controls regularization and

$$E_\epsilon(y(\mathbf{x}) - t) = \begin{cases} 0 & \text{if } |y(\mathbf{x}) - t| < \epsilon \\ |y(\mathbf{x}) - t| - \epsilon & \text{otherwise} \end{cases}$$

**Figure 7.6** Plot of an  $\epsilon$ -insensitive error function (in red) in which the error increases linearly with distance beyond the insensitive region. Also shown for comparison is the quadratic error function (in green).



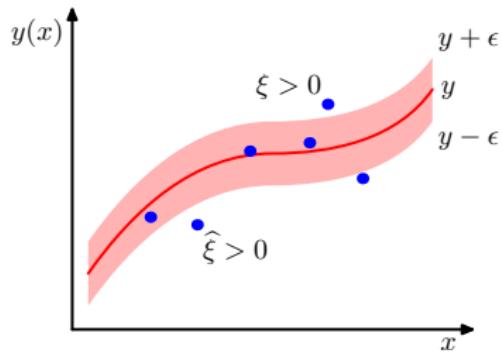
## Support Vector Machines for Regression

- ▶ The values of  $C$  and  $\epsilon$  can be decided by cross-validation.
- ▶ Consider the slack variables  $\xi \geq 0$  and  $\hat{\xi}_n \geq 0$  such that

$$\xi_n = \begin{cases} t_n - y(\mathbf{x}_n) - \epsilon & \text{if } t_n > y(\mathbf{x}_n) + \epsilon \\ 0 & \text{otherwise} \end{cases}$$

and

$$\hat{\xi}_n = \begin{cases} y(\mathbf{x}_n) + \epsilon - t_n & \text{if } t_n < y(\mathbf{x}_n) + \epsilon \\ 0 & \text{otherwise} \end{cases}$$



## Support Vector Machines for Regression

- The optimal regression curve is given by

$$\arg \min_{\mathbf{w}} C \sum_n (\xi_n + \widehat{\xi}_n) + \frac{1}{2} \|\mathbf{w}\|^2$$

subject to  $\xi \geq 0$ ,  $\widehat{\xi}_n \geq 0$ ,  $t_n \leq y(\mathbf{x}_n) + \epsilon + \xi_n$  and  $t_n \geq y(\mathbf{x}_n) - \epsilon - \widehat{\xi}_n$ .

- To minimize the previous expression, we minimize

$$C \sum_n (\xi_n + \widehat{\xi}_n) + \frac{1}{2} \|\mathbf{w}\|^2 - \sum_n (\mu_n \xi_n + \widehat{\mu}_n \widehat{\xi}_n)$$
$$- \sum_n a_n (y(\mathbf{x}_n) + \epsilon + \xi_n - t_n) - \sum_n \widehat{a}_n (t_n - y(\mathbf{x}_n) + \epsilon + \widehat{\xi}_n)$$

where  $\mu_n \geq 0$ ,  $\widehat{\mu}_n \geq 0$ ,  $a_n \geq 0$  and  $\widehat{a}_n \geq 0$  are Lagrange multipliers.

- Setting its derivatives with respect to  $\mathbf{w}$ ,  $b$ ,  $\xi_n$  and  $\widehat{\xi}_n$  to zero gives

$$\mathbf{w} = \sum_n (a_n - \widehat{a}_n) \phi(\mathbf{x}_n)$$

$$0 = \sum_n (a_n - \widehat{a}_n)$$

$$C = a_n + \mu_n$$

$$C = \widehat{a}_n + \widehat{\mu}_n$$

## Support Vector Machines for Regression

- Replacing these in the Lagrangian function gives the dual representation

$$\frac{1}{2} \sum_n \sum_m (a_n - \hat{a}_n)(a_m - \hat{a}_m) k(\mathbf{x}_n, \mathbf{x}_m) - \epsilon \sum_n (a_n + \hat{a}_n) + \sum_n (a_n - \hat{a}_n) t_n$$

subject to  $a_n \geq 0$  and  $a_n \leq C$  for all  $n$ , because  $\mu_n \geq 0$ . Similarly for  $\hat{a}_n$ .

- When maximizing the Lagrangian function, the Karush-Kuhn-Tucker conditions hold for all  $n$ :

$$a_n(y(\mathbf{x}_n) + \epsilon + \xi_n - t_n) = 0$$

$$\hat{a}_n(t_n - y(\mathbf{x}_n) + \epsilon + \widehat{\xi}_n) = 0$$

$$\mu_n \xi_n = 0$$

$$\widehat{\mu}_n \widehat{\xi}_n = 0$$

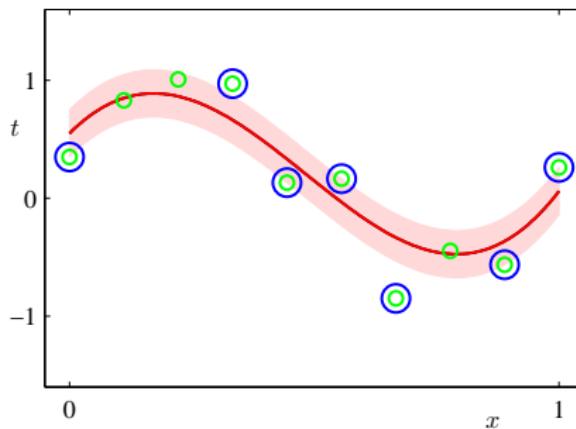
- Then,  $a_n > 0$  if and only if  $y(\mathbf{x}_n) + \epsilon + \xi_n - t_n = 0$ , which implies that  $\mathbf{x}_n$  lies on the upper margin of the  $\epsilon$ -tube or above it. Similarly for  $\hat{a}_n > 0$ .

## Support Vector Machines for Regression

- The prediction for a new point  $\mathbf{x}$  is made according to

$$y(\mathbf{x}) = \sum_{n \in S} (a_n - \hat{a}_n) k(\mathbf{x}, \mathbf{x}_n) + b$$

where  $S$  are the indexes of the support vectors.



- To find  $b$ , consider any support vector  $\mathbf{x}_n$  with  $0 < a_n < C$ . Then,  $\mu_n > 0$  and thus  $\xi_n = 0$  and thus  $0 = t_n - \epsilon - y(\mathbf{x}_n)$ . Then,

$$b = t_n - \epsilon - \sum_{m \in S} (a_m - \hat{a}_m) k(\mathbf{x}_n, \mathbf{x}_m)$$

## Summary

- ▶ Kernel trick: It allows to work in the feature space without constructing it.
- ▶ Quadratic objective function: It allows to obtain the global optimum for a given kernel and  $C/\epsilon$  (which are obtained by cross-validation).
- ▶ Sparse model: Only the support vectors are needed for classification/regression.

# 732A95 Introduction to Machine Learning

## Lecture 6a: Neural Networks

Jose M. Peña  
IDA, Linköping University, Sweden

- ▶ Neural Networks
- ▶ Backpropagation Algorithm
- ▶ Regularization
- ▶ Convolutional Networks
- ▶ Summary

## Neural Networks

- ▶ Consider binary classification with input space  $\mathbb{R}^D$ . Consider a training set  $\{(\mathbf{x}_n, t_n)\}$  where  $t_n \in \{-1, +1\}$ .
- ▶ SVMs classify a new point  $\mathbf{x}$  according to

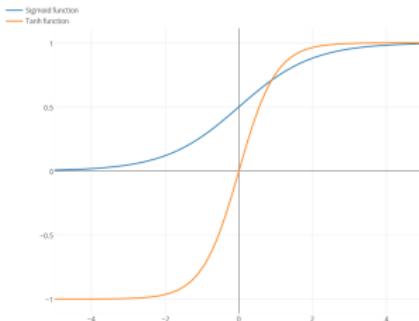
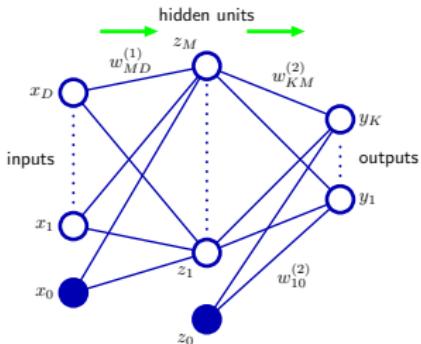
$$y(\mathbf{x}) = \text{sgn} \left( \sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}, \mathbf{x}_m) + b \right)$$

- ▶ Consider regressing an unidimensional continuous random variable on a  $D$ -dimensional continuous random variable. Consider a training set  $\{(\mathbf{x}_n, t_n)\}$
- ▶ For a new point  $\mathbf{x}$ , SVMs predict

$$y(\mathbf{x}) = \sum_{m \in \mathcal{S}} (a_n - \hat{a}_n) k(\mathbf{x}, \mathbf{x}_m) + b$$

- ▶ SVMs imply **data-selected user-defined** basis functions.
- ▶ NNs imply a **user-defined** number of **data-selected** basis functions.

# Neural Networks



- ▶ Activations:  $a_j = \sum_i w_{ji}^{(1)} x_i + w_{j0}^{(1)}$
- ▶ Hidden units and activation function:  $z_j = h(a_j)$
- ▶ Output activations:  $a_k = \sum_j w_{kj}^{(2)} z_j + w_{k0}^{(2)}$
- ▶ Output activation function for regression:  $y_k(\mathbf{x}) = a_k$
- ▶ Output activation function for classification:  $y_k(\mathbf{x}) = \sigma(a_k)$
- ▶ Sigmoid function:  $\sigma(a) = \frac{1}{1+\exp(-a)}$
- ▶ Two-layer NN:

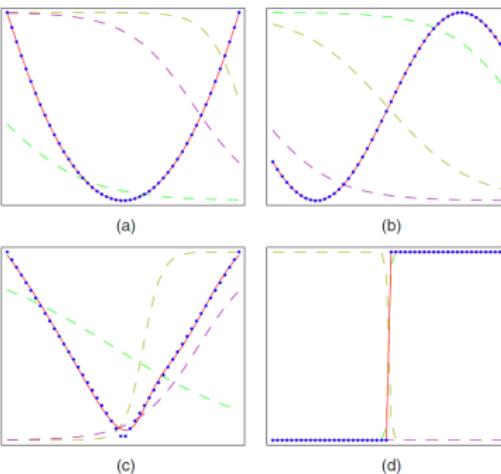
$$y_k(\mathbf{x}) = \sigma \left( \sum_j w_{kj}^{(2)} h \left( \sum_i w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right)$$

- ▶ Evaluating the previous expression is known as forward propagation. The NN is said to have a feed-forward architecture.
- ▶ All the previous is, of course, generalizable to more layers.

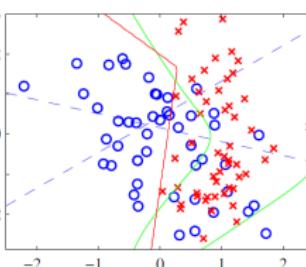
# Neural Networks

- For a large variety of activation function, the two-layer NN can uniformly approximate any continuous function to arbitrary accuracy provided enough hidden units. Easy to fit the parameters ? Overfitting ?!

**Figure 5.3** Illustration of the capability of a multilayer perceptron to approximate four different functions comprising (a)  $f(x) = x^2$ , (b)  $f(x) = \sin(x)$ , (c),  $f(x) = |x|$ , and (d)  $f(x) = H(x)$  where  $H(x)$  is the Heaviside step function. In each case,  $N = 50$  data points, shown as blue dots, have been sampled uniformly in  $x$  over the interval  $(-1, 1)$  and the corresponding values of  $f(x)$  evaluated. These data points are then used to train a two-layer network having 3 hidden units with ‘tanh’ activation functions and linear output units. The resulting network functions are shown by the red curves, and the outputs of the three hidden units are shown by the three dashed curves.

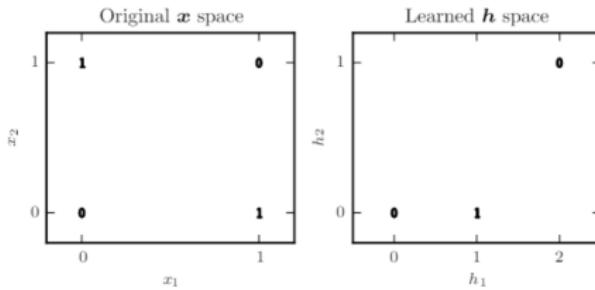
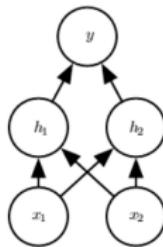


**Figure 5.4** Example of the solution of a simple two-class classification problem involving synthetic data using a neural network having two inputs, two hidden units with ‘tanh’ activation functions, and a single output having a logistic sigmoid activation function. The dashed blue lines show the  $z = 0.5$  contours for each of the hidden units, and the red line shows the  $y = 0.5$  decision surface for the network. For comparison, the green line denotes the optimal decision boundary computed from the distributions used to generate the data.



# Neural Networks

- ▶ Solving the XOR problem with NNs.
- ▶ No line shatters the points in the original space.
- ▶ The NN represents a mapping of the input space to an alternative space where a line can shattered the points. Note that the points  $(0,1)$  and  $(1,0)$  are mapped both to the point  $(1,0)$ .
- ▶ It resembles SVMs.



$$w_{11}^{(1)} = w_{12}^{(1)} = w_{21}^{(1)} = w_{22}^{(1)} = 1$$

$$w_{10}^{(1)} = 0, \quad w_{20}^{(1)} = -1$$

$$z_k = h(a_j) = \max\{0, a_j\}$$

$$w_{11}^{(2)} = 1, \quad w_{12}^{(2)} = -2$$

$$w_{10}^{(2)} = 0$$

$$y_k = a_k$$

## Backpropagation Algorithm

- ▶ Consider regressing an  $K$ -dimensional continuous random variable on a  $D$ -dimensional continuous random variable.
- ▶ Consider a training set  $\{(\mathbf{x}_n, \mathbf{t}_n)\}$ . Consider minimizing the error function

$$E(\mathbf{w}^t) \equiv \sum_n E_n(\mathbf{w}^t) \equiv \sum_n \frac{1}{2} (\mathbf{y}(\mathbf{x}_n) - \mathbf{t}_n)^2$$

- ▶ The weight space is highly multimodal and, thus, we have to resort to approximate iterative methods to minimize the previous expression.
- ▶ Batch gradient descent

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla E(\mathbf{w}^t)$$

where  $\eta > 0$  is the learning rate

- ▶ Sequential gradient descent

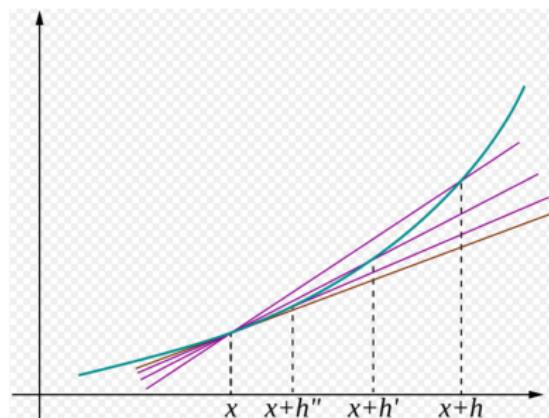
$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla E_n(\mathbf{w}^t)$$

where  $n$  is chosen randomly or sequentially.

- ▶ Sequential gradient descent is less affected by the multimodality problem, as a local minimum of the whole data will not be generally a local minimum of each individual point.

## Backpropagation Algorithm

- Recall that  $f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h)-f(x)}{h}$



- Recall that  $\nabla E_n(\mathbf{w}^t)$  is a vector whose components are the partial derivatives of  $E_n(\mathbf{w}^t)$ .

## Backpropagation Algorithm

- ▶ Since  $E_n$  depends on  $w_{ji}$  only via  $a_j$ , and  $a_j = \sum_i w_{ji} z_i$ , then

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} z_i$$

- ▶ Since  $E_n$  depends on  $a_j$  only via  $a_k$ , then

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial a_k}{\partial a_j}$$

- ▶ Since  $a_k = \sum_j w_{kj} z_j$  and  $z_j = h(a_j)$ , then

$$\frac{\partial a_k}{\partial a_j} = h'(a_j) w_{kj}$$

- ▶ Putting all together, we have that

$$\frac{\partial E_n}{\partial w_{ji}} = z_i h'(a_j) \sum_k \delta_k w_{kj}$$

- ▶ Since  $y_k = a_k$  for regression, then

$$\delta_k \equiv \frac{\partial E_n}{\partial a_k} = y_k - t_k$$

- ▶ Backpropagation algorithm:

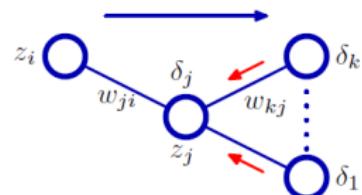
1. Forward propagate to compute activations, and hidden and output units.
2. Compute  $\delta_k$  for the output units.
3. Backpropagate the  $\delta$ 's, i.e. evaluate  $\delta_j$  for the hidden units recursively.
4. Compute the required derivatives.

## Backpropagation Algorithm

- ▶ Backpropagation algorithm:

1. Forward propagate to compute activations, and hidden and output units.
2. Compute  $\delta_k$  for the output units.
3. Backpropagate the  $\delta$ 's, i.e. evaluate  $\delta_j$  for the hidden units recursively.
4. Compute the required derivatives.

**Figure 5.7** Illustration of the calculation of  $\delta_j$  for hidden unit  $j$  by backpropagation of the  $\delta$ 's from those units  $k$  to which unit  $j$  sends connections. The blue arrow denotes the direction of information flow during forward propagation, and the red arrows indicate the backward propagation of error information.

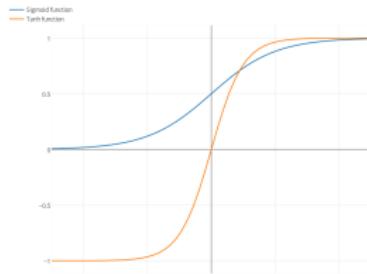
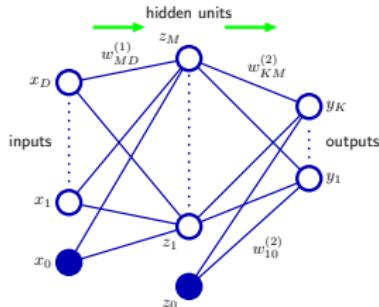


- ▶ Since  $y_k = \sigma(a_k)$  for classification, then

$$\delta_k \equiv \frac{\partial E_n}{\partial a_k} = \sigma(a_k)(1 - \sigma(a_k))$$

- ▶ This is an example of embarrassingly parallel algorithm.

# Backpropagation Algorithm



- Example:  $y_k = a_k$ , and  $z_j = h(a_j) = \tanh(a_j)$  where  $\tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)}$ .
- Note that  $h'(a) = 1 - h(a)^2$ .
- Backpropagation:

1. Forward propagation, i.e. compute

$$a_j = \sum_i w_{ji} x_i \text{ and } z_j = h(a_j) \text{ and } y_k = \sum_j w_{kj} z_j$$

2. Compute

$$\delta_k = y_k - t_k$$

3. Backpropagate, i.e. compute

$$\delta_j = (1 - z_j^2) \sum_k w_{kj} \delta_k$$

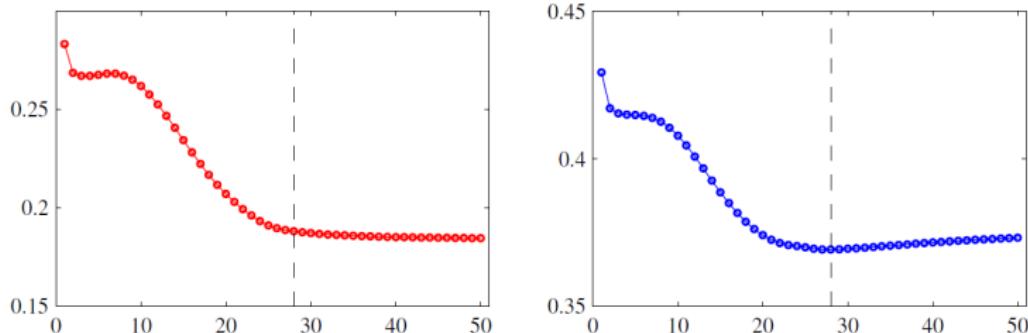
4. Compute

$$\frac{\partial E_n}{\partial w_{kj}} = \delta_k z_j \text{ and } \frac{\partial E_n}{\partial w_{ji}} = \delta_j x_i$$

## Backpropagation Algorithm

- ▶ The weight space is non-convex and has many symmetries, plateaus and local minima. So, the initialization of the weights in the backpropagation algorithm is crucial.
- ▶ Hints based on experimental rather than theoretical analysis:
  - ▶ Initialize the weights to different values, otherwise they would be updated in the same way because the algorithm is deterministic, and so creating redundant hidden units.
  - ▶ Initialize the weights at random, but
    - ▶ too small magnitude values may cause losing signal in the forward or backward passes, and
    - ▶ too big magnitude values may cause the activation function to saturate and loss gradient.
  - ▶ Initialize the weights according to prior knowledge: Almost-zero for hidden units that are unlikely to interact, and bigger magnitude values for the rest.
  - ▶ Initialize the weights to almost-zero values so that the initial model is almost-linear, i.e. the sigmoid function is almost-linear around the zero. Let the algorithm to introduce non-linearities where needed.
    - ▶ Note however that this initialization makes the sigmoid function to fire at around half its saturation level. That is why the hyperbolic tangent function is sometimes preferred in practice.

## Regularization



**Figure 5.12** An illustration of the behaviour of training set error (left) and validation set error (right) during a typical training session, as a function of the iteration step, for the sinusoidal data set. The goal of achieving the best generalization performance suggests that training should be stopped at the point shown by the vertical dashed lines, corresponding to the minimum of the validation set error.

- ▶ Regularization when learning the parameters: Early stopping the backpropagation algorithm according to the error on some validation data.
- ▶ Regularization when learning the structure:
  - ▶ Cross-validation.
  - ▶ Penalizing complexity according to

$$E(\mathbf{w}) + \frac{\lambda_1}{2} \|\mathbf{w}^{(1)}\|^2 + \frac{\lambda_2}{2} \|\mathbf{w}^{(2)}\|^2$$

instead of the classical

$$E(\mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

and choose  $\lambda_1$  and  $\lambda_2$  by cross-validation.

## Regularization

- ▶ To see why, let  $z_j = h(\sum_i w_{ji}x_i + w_{j0})$  and  $y_k = \sum_j w_{kj}z_j + w_{k0}$ .
- ▶ Consider the linear transformation  $X_i \rightarrow ax_i + b$ .
- ▶ Transform  $w_{j0} \rightarrow w_{j0} - \frac{b}{a} \sum_i w_{ji}$  and  $w_{ji} \rightarrow \frac{1}{a}w_{ji}$ .
- ▶ Both NNs define the same mapping, but they may receive different penalty for complexity  $\|\mathbf{w}\|^2$ .
- ▶ Solution: Penalize according to  $\frac{\lambda_1}{2}\|\mathbf{w}^{(1)}\|^2 + \frac{\lambda_2}{2}\|\mathbf{w}^{(2)}\|^2$  and let  $\lambda_1 \rightarrow a^{1/2}\lambda_1$ .
- ▶ Finally, note that the effect of the penalty is simply to add  $\lambda_1 w_{ji}$  and  $\lambda_2 w_{kj}$  to the appropriate derivatives.
- ▶ NNs: Nonlinear mapping from input to output.
- ▶ Extremely expressive.
- ▶ Training: Backpropagation algorithm, and regularization.

# 732A95 Introduction to Machine Learning

## Lecture 6b: Deep Learning

Jose M. Peña  
IDA, Linköping University, Sweden

- ▶ Limitations of Neural Networks
- ▶ Deep Neural Networks
- ▶ Convolutional Networks
- ▶ Rectifier Activation Function
- ▶ Layer-Wise Pre-Training
- ▶ Summary

## Limitations of Neural Networks

### Theorem (Universal approximation theorem)

For every continuous function  $f : [a, b]^D \rightarrow \mathbb{R}$  and for every  $\epsilon > 0$ , there exist a NN with one hidden layer such that

$$\sup_{\mathbf{x} \in [a, b]^D} |f(\mathbf{x}) - y(\mathbf{x})| < \epsilon$$

### Theorem (Universal classification theorem)

Let  $\mathcal{C}^{(k)}$  contain all classifiers defined by NNs of one hidden layer with  $k$  hidden units, and an arbitrary sigmoid function  $\sigma$ . Then, for any distribution  $p(\mathbf{x}, t)$ ,

$$\lim_{k \rightarrow \infty} \inf_{y \in \mathcal{C}^{(k)}} E(y(\mathbf{x})) - E(p(t|\mathbf{x})) = 0$$

- ▶ How many hidden units has such a NN ?
- ▶ How much data do we need to learn such a NN (and avoid overfitting) via the backpropagation algorithm ?
- ▶ How fast converges the backpropagation algorithm to such a NN ?  
Assuming that it does not get trapped in a local minimum
- ▶ The answer to the last two questions depends on the first: More hidden units implies more training time and higher generalization error.

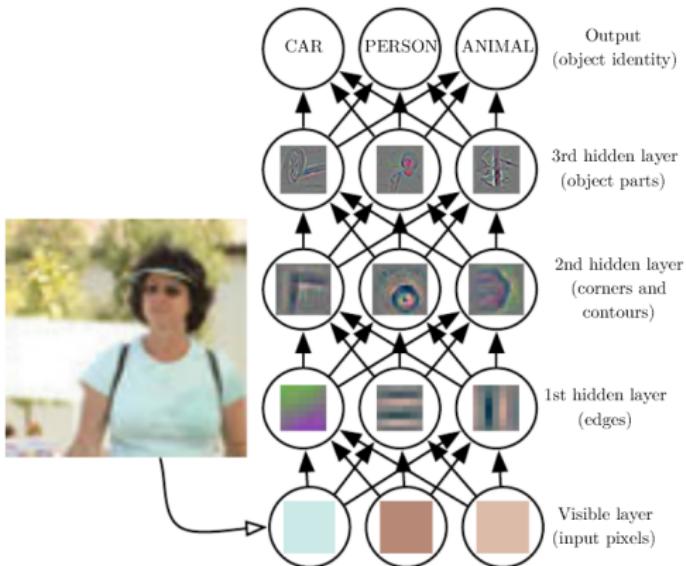
## Limitations of Neural Networks

- ▶ How many hidden units does the NN need ?
- ▶ Any Boolean function can be written in disjunctive normal form (OR of ANDs) or conjunctive normal form (AND of ORs). This is a depth-two logical circuit.
- ▶ For most Boolean functions, the size of the circuit is exponential in the size of the input.
- ▶ However, there are Boolean functions that have a polynomial-size circuit of depth  $k$  and an exponential-size circuit of depth  $k - 1$ .
- ▶ Then, there is no universally right depth. Ideally, we should let the data determine the right depth.

### Theorem (No free lunch theorem)

*For any algorithm, good performance on some problems comes at the expense of bad performance on some others.*

# Deep Neural Networks



- ▶ A deep NN is a function that maps input to output.
- ▶ The mapping is formed by composing many simpler functions.
- ▶ Each layer provides a new representation of the input, i.e. complex concepts are built from simpler ones.
- ▶ The representation is learned automatically from data.

# Deep Neural Networks

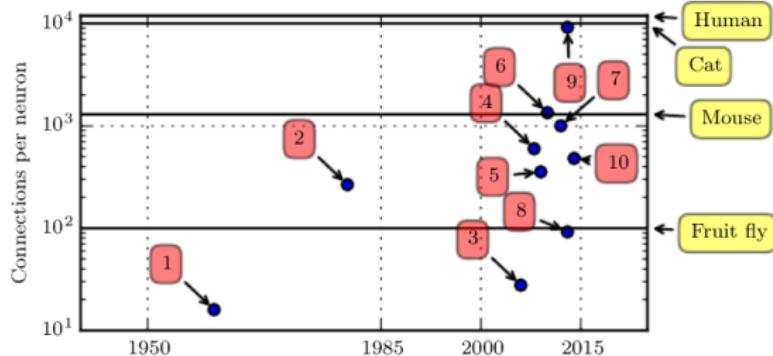


Figure 1.10: Initially, the number of connections between neurons in artificial neural networks was limited by hardware capabilities. Today, the number of connections between neurons is mostly a design consideration. Some artificial neural networks have nearly as many connections per neuron as a cat, and it is quite common for other neural networks to have as many connections per neuron as smaller mammals like mice. Even the human brain does not have an exorbitant amount of connections per neuron. Biological neural network sizes from [Wikipedia \(2015\)](#).

1. Adaptive linear element ([Widrow and Hoff, 1960](#))
2. Neocognitron ([Fukushima, 1980](#))
3. GPU-accelerated convolutional network ([Chellapilla et al., 2006](#))
4. Deep Boltzmann machine ([Salakhutdinov and Hinton, 2009a](#))
5. Unsupervised convolutional network ([Jarrett et al., 2009](#))
6. GPU-accelerated multilayer perceptron ([Ciresan et al., 2010](#))
7. Distributed autoencoder ([Le et al., 2012](#))
8. Multi-GPU convolutional network ([Krizhevsky et al., 2012](#))
9. COTS HPC unsupervised convolutional network ([Coates et al., 2013](#))
10. GoogLeNet ([Szegedy et al., 2014a](#))

# Deep Neural Networks

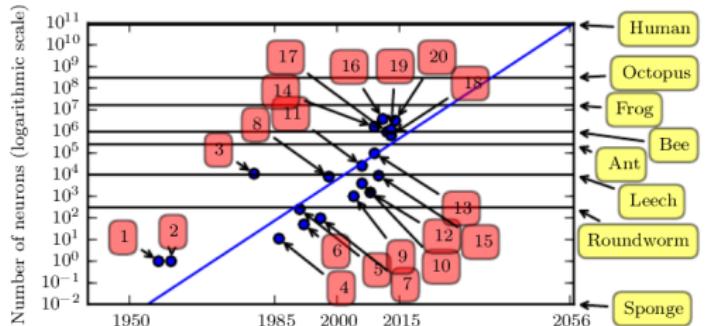


Figure 1.11: Since the introduction of hidden units, artificial neural networks have doubled in size roughly every 2.4 years. Biological neural network sizes from [Wikipedia \(2015\)](#).

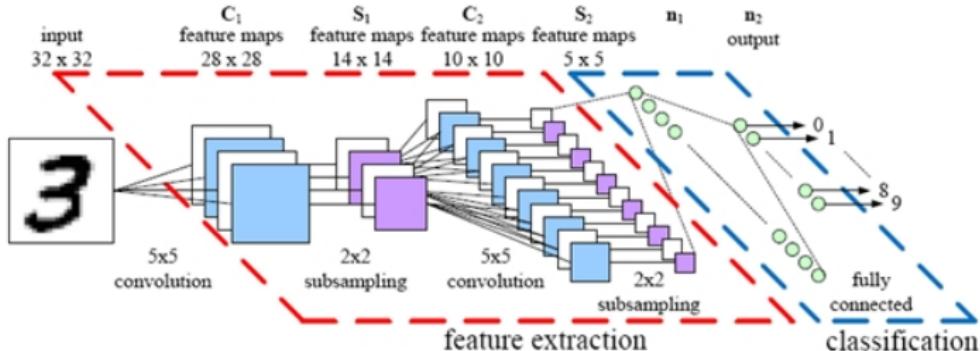
1. Perceptron ([Rosenblatt, 1958, 1962](#))
2. Adaptive linear element ([Widrow and Hoff, 1960](#))
3. Neocognitron ([Fukushima, 1980](#))
4. Early back-propagation network ([Rumelhart \*et al.\*, 1986b](#))
5. Recurrent neural network for speech recognition ([Robinson and Fallside, 1991](#))
6. Multilayer perceptron for speech recognition ([Bengio \*et al.\*, 1991](#))
7. Mean field sigmoid belief network ([Saul \*et al.\*, 1996](#))
8. LeNet-5 ([LeCun \*et al.\*, 1998b](#))
9. Echo state network ([Jaeger and Haas, 2004](#))
10. Deep belief network ([Hinton \*et al.\*, 2006](#))
11. GPU-accelerated convolutional network ([Chellapilla \*et al.\*, 2006](#))
12. Deep Boltzmann machine ([Salakhutdinov and Hinton, 2009a](#))
13. GPU-accelerated deep belief network ([Raina \*et al.\*, 2009](#))
14. Unsupervised convolutional network ([Jarrett \*et al.\*, 2009](#))
15. GPU-accelerated multilayer perceptron ([Ciresan \*et al.\*, 2010](#))
16. OMP-1 network ([Coates and Ng, 2011](#))
17. Distributed autoencoder ([Le \*et al.\*, 2012](#))
18. Multi-GPU convolutional network ([Krizhevsky \*et al.\*, 2012](#))
19. COTS HPC unsupervised convolutional network ([Coates \*et al.\*, 2013](#))
20. GoogLeNet ([Szegedy \*et al.\*, 2014a](#))

22 layers DNN, but  
12 times fewer weights  
than DNN 19

# Deep Neural Networks

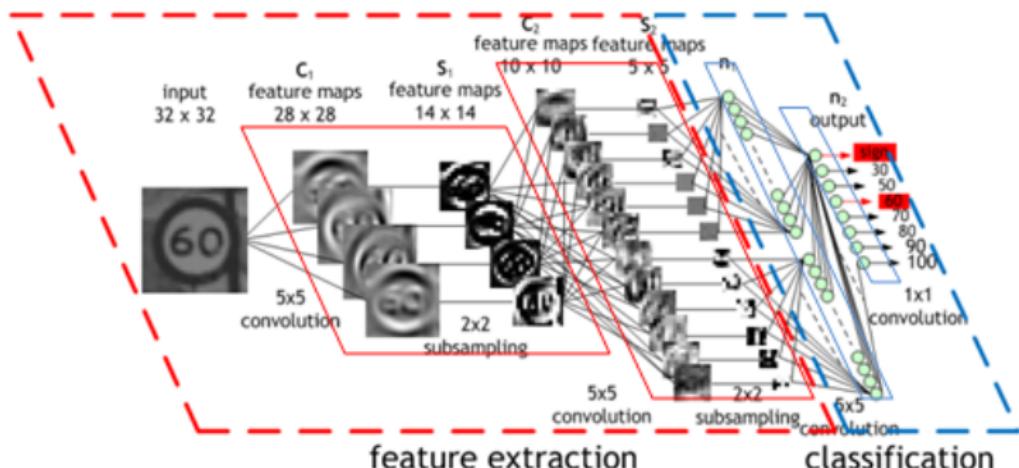
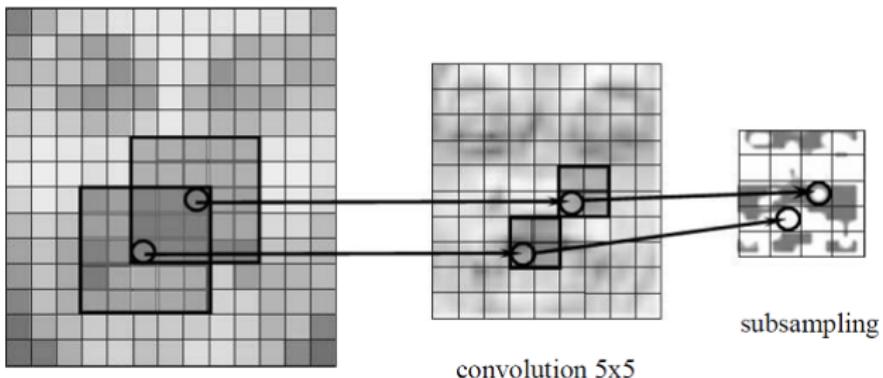
- ▶ Training DNNs is difficult:
  - ▶ Typically, poorer generalization than (shallow) NNs.
  - ▶ The gradient may vanish/explode as we move away from the output layer, due to multiplying small/big quantities. E.g. the gradient of  $\sigma$  and  $\tanh$  is in  $[0, 1]$ . So, they may only suffer the gradient vanishing problem. Other activation functions may suffer the gradient exploding problem.
  - ▶ There may be larger plateaus and many more local minima than with NNs.
- ▶ Training DNNs is doable:
  - ▶ Convolutional networks, particularly suitable for image processing.
  - ▶ Rectifier activation function, a new activation function.
  - ▶ Layer-wise pre-training, to find a good starting point for training.
- ▶ In addition to performance, the computational demands of the training must be considered, e.g. CPU, GPU, memory, parallelism, etc.
  - ▶ The authors state that GoogLeNet was trained "using modest amount of model and data-parallelism. Although we used a CPU based implementation only, a rough estimate suggests that the GoogLeNet network could be trained to convergence using few high-end GPUs within a week, the main limitation being the memory usage".

# Convolutional Networks



- ▶ DNNs suitable for image recognition, since they exhibit invariance to translation, scaling, rotations, and warping.
- ▶ Convolution: Detection of local features, e.g.  $a_j$  is computed from a  $5 \times 5$  pixel patch of the image.
- ▶ To achieve invariance, the units in the convolution layer share the same activation function and weights.
- ▶ Subsampling: Combination of local features into higher-order features, e.g.  $a_k$  is computed from a  $2 \times 2$  pixel patch of the convoluted image.
- ▶ There are several feature maps in each layer, to compensate the reduction in resolution by increasing in the number of features being detected.
- ▶ The final layer is a regular NN for classification.

# Convolutional Networks



## Convolutional Networks

- ▶ DNNs allow increased depth because
  - ▶ they are sparse, which allows the gradient to propagate further, and
  - ▶ they have relatively few weights to fix due to feature locality and weight sharing.
- ▶ The backpropagation algorithm needs to be adapted, by modifying the derivatives with respect to the weights in each convolution layer  $m$ .
- ▶ Since  $E_n$  depends on  $w_i^{(m)}$  only via  $a_j^{(m)}$ , and  $a_j^{(m)} = \sum_{i \in L_j^{(m)}} w_i^{(m)} z_i^{(m-1)}$  where  $L_j^{(m)}$  is the set of indexes of the input units, then

$$\frac{\partial E_n}{\partial w_i^{(m)}} = \sum_j \frac{\partial E_n}{\partial a_j^{(m)}} \frac{\partial a_j^{(m)}}{\partial w_i^{(m)}} = \sum_j \delta_j^{(m)} z_i^{(m-1)}$$

- ▶ Note that  $w_i^{(m)}$  does not depend on  $j$  by weight sharing, whereas  $i \in L_j^{(m)}$  by feature locality.

## Rectifier Activation Function

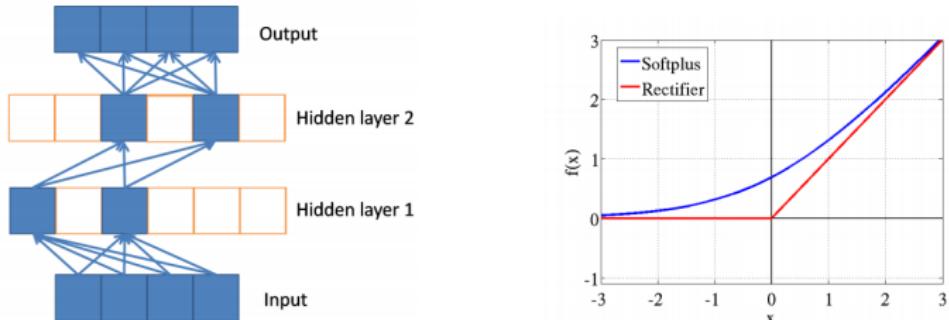


Figure 2: *Left:* Sparse propagation of activations and gradients in a network of rectifier units. The input selects a subset of active neurons and computation is linear in this subset. *Right:* Rectifier and softplus activation functions. The second one is a smooth version of the first.

- ▶  $\text{rectifier}(x) = \max\{0, x\}$ , i.e. hidden units are off or operating in a linear regime.
- ▶ The most popular choice nowadays.
- ▶ Sparsity promoting: Uniform initialization of the weights implies that around 50 % of the hidden units are off.
- ▶ Piece-wise linear mapping: The input selects which hidden units are active, and the output is a liner function of the input in the selected hidden units.

# Rectifier Activation Function

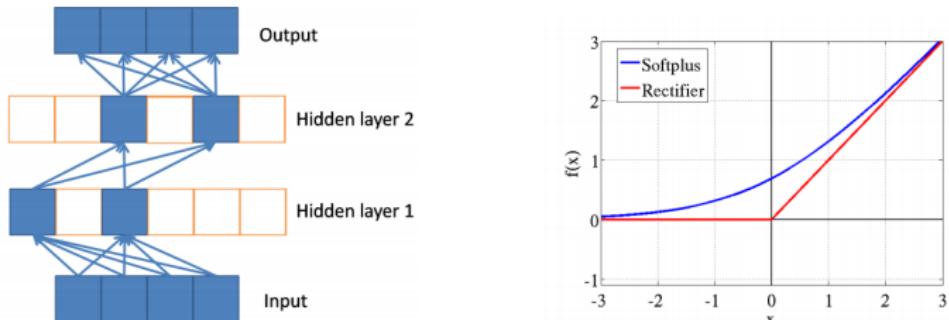


Figure 2: *Left:* Sparse propagation of activations and gradients in a network of rectifier units. The input selects a subset of active neurons and computation is linear in this subset. *Right:* Rectifier and softplus activation functions. The second one is a smooth version of the first.

- ▶ It simplifies the backpropagation algorithm as  $h'(a_j) = 1$  for the selected units. So, there is no gradient vanishing on the paths of selected units.
- ▶ Note that  $h'(0)$  does not exist since  $h'_+(0) \neq h'_-(0)$ . We can get around this problem by simply returning one of two one-sided derivatives. Or using a generalization of the rectifier function.
- ▶ Regularization is typically added to prevent numerical problems due to the activation being unbounded, e.g. when forward propagating.

## Layer-Wise Pre-Training

- ▶ Supervised version:
  1. Train each layer of the DNN as if it was the hidden layer in a depth-two NN. As input, use the output of the last of the previously trained layers.
  2. Run the backpropagation algorithm to fine-tune the weights.
- ▶ The pre-training aims to find a good starting point for the subsequent run of the backpropagation algorithm.
- ▶ Unsupervised version: Similar to the supervised one but the hidden layers (except the last one) are trained to learn an encoding of the output of the previous layer, instead of the original classification or regression function.

## Summary

- ▶ Direct application of the backpropagation algorithm to DNNs produces poor results.
- ▶ Convolutional networks: It makes the backpropagation algorithm more efficient by using local features and weight sharing. This also achieves invariance, which is particularly important for image processing.
- ▶ Rectifier activation function: Free of gradient vanishing problem and it simplifies the backpropagation algorithm.
- ▶ Layer-wise pre-training: Heuristic weight initialization to alleviate the local optima problem.