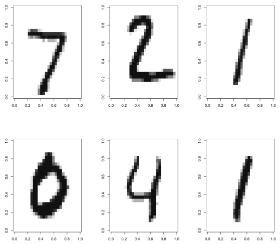
Example: classifying hadwritten digits



Example: classifying hadwritten digits

Training data: 60000 images.
Test data: 10000 images.
Features: intensities (0-255, scaled to 0-1) in the $28 \times 28 = 784$ pixels as features.

Methods:

- Multinomial regression with LASSO prior
- Support vector machines
- Neural Networks (deep?)

Example: classifying hadwritten digits

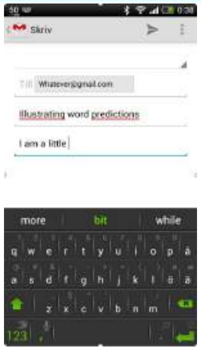
- Confusion matrix

PREDICTION

TRUTH

	0	1	2	3	4	5	6	7	8	9
0	966	0	8	1	1	7	9	2	4	6
1	0	1121	1	1	0	2	3	13	7	7
2	2	2	957	13	5	4	4	21	7	0
3	0	2	9	947	0	29	1	3	12	10
4	0	0	12	1	940	5	5	9	8	32
5	6	1	3	19	1	816	9	1	24	9
6	4	4	13	1	7	12	926	0	10	1
7	1	0	9	10	2	2	0	954	5	13
8	1	4	17	11	2	10	1	3	892	4
9	0	1	3	6	24	5	0	22	5	927

Example: smartfone typing predictions



Till Whatever@gmail.com

Illustrating word predictions

I am a little

Example: smartfone typing predictions

- Assume a simple (Markov) model of a sentence:

$$p(w_1, \dots, w_n) = p(w_1)p(w_2|w_1) \dots p(w_n|w_{n-1})$$

- Intuition:
  - $p(person|crazy) = 0.1$
  - $p(horse|crazy) = 0.0001$

Highest P(?|Donald) ?

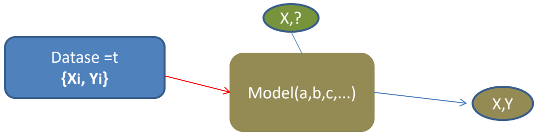
- Probability for sentence depends only on $p(w_n|w_{n-1})$
- How to compute ? Investigate a lot of data!

$$p(w_k|w_{k-1}) = \frac{\#\ cases\ w_k\ follows\ w_{k-1}}{\#\ cases\ w_k}$$

- In practice, more advanced model used
  - Neural networks for ex.

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 (→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 ingridients

- **Data** D: observations (cases)
  - Features $X_1, .. X_p$
  - Targets $Y_1, \dots, Y_r$

Case	X_1	X_2	Y
1			
2			
…			

- **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 $\widehat{w}$ or $p(w|D)$)
  - Maximum likelihood, Bayesian estimation…
- **Prediction** of new data X^{new} by using the fitted model

Types of data sets

- **Training data** (training set D): used for fitting the model
  - Supervised learning: w_i in $P(y|\boldsymbol{x}, w_1, \dots w_k)$ estimated using D

X	Y
1.1	M
2.3	F

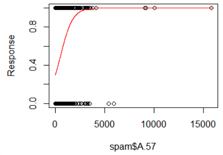
- **Test data** (test set T): used for predictions
  - Supervised learning: estimate $p(Y)$ or $\hat{Y}$ for new $\boldsymbol{x}$

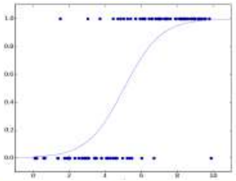
X	Y
1.3	?
2.9	?

Logistic regression

- Data $Y_i \in \{Spam, Not\ Spam\}, X_i = \#of\ a\ word$
- Model: $p(Y = Spam|w, x) = \frac{1}{1+e^{-w_0 - w_1 x}}$
- Fitting: maximum likelihood
- Prediction : $p(spam) = p(Y = spam|x)$

We can also make point predictions -how?

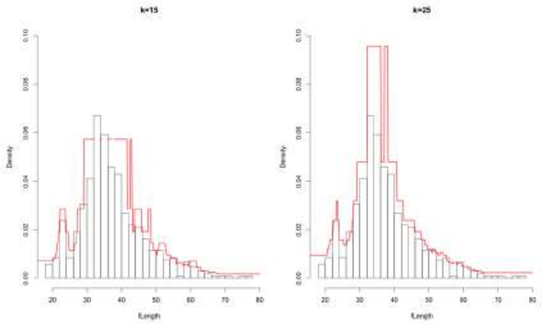




K-nearest neighbor density estimation

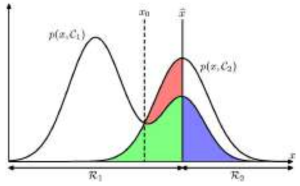
- Data: Fish length $X_1, \dots X_N$
- Model $p(x|K) = \frac{K}{N \cdot \Delta}$
  - K: #neighbors in training data
  - Δ: length of the interval containing K neighbors
- Learning: Fix some K or find an appropriate K
- Prediction: predict $p(x|K)$

K-nearest neighbor density estimation



K-nearest neighbor density estimation

- Why estimating a density can be interesting:
  1. Estimate **class-conditional densities** $p(x|y = C_i)$
  2. Predict



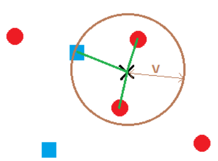
K-nearest neighbor classification

- Given N observations $(\boldsymbol{X}_j, Y_j)$
  - $Y_j = C_i$, 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 $Y = C_i$

3-NN method



Bayesian classification

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

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

- Bayes theorem

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

- We get

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

K-nearest neighbor classification

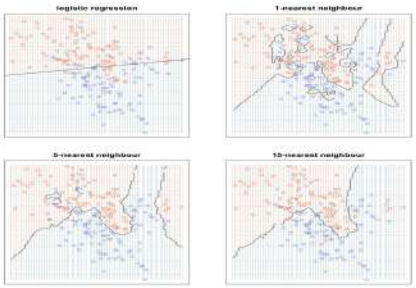
Algorithm

```
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 $l = \arg \max_{i \in \{1,\dots,m\}} p'(C_i|\boldsymbol{x})$
    3. Predict $\hat{Y}(x) = C_l$
```

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

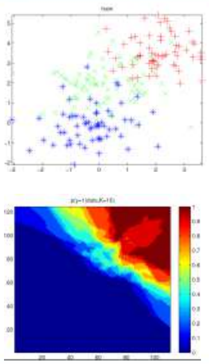
K-nearest neigbor example

Why classification results are so different for K-NN?



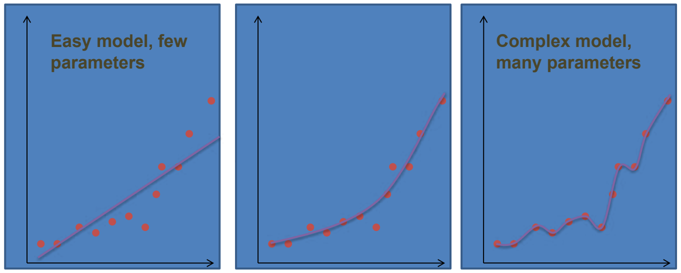
Model types

- 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



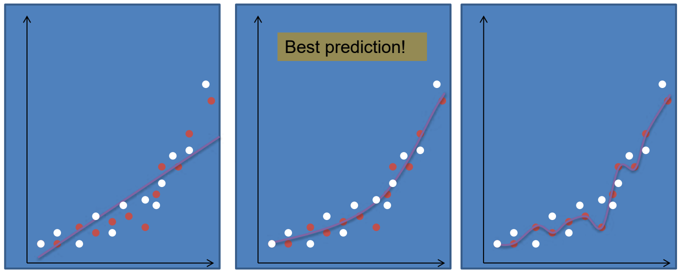
Overfitting

- Which model feels appropriate?



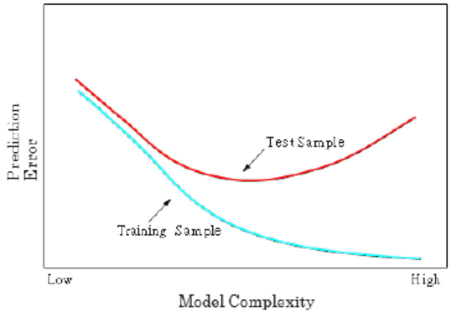
Overfitting

Now new data from the same process



Overfitting

- Observed:



Model selection

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

Training	Test

- 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

Typical error functions

- Regression, **MSE** :

$$R(Y, \hat{Y}) = \frac{1}{N} \sum_{i=1}^{N} (Y_i - \hat{Y}_i)^2$$

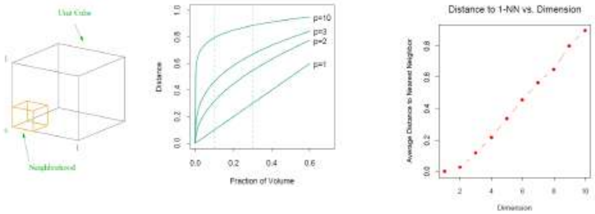
- Classification, **misclassification rate**

$$R(Y, \hat{Y}) = \frac{1}{N} \sum_{i=1}^{N} I(Y_i \neq \hat{Y}_i)$$

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 misleadning
  - 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

Probability

How likely it is that some event will happen?

Idea:

- Experiment
- Outcomes (sample points) $O_1, O_2, \ldots O_n$
- Sample space Ω
- Event A
- Probability function P: Events →[0,1]

Probability

Example: Tossing a coin two times

http://cdn.toonvectors.com/images/35/10267/toonvectors-10267-940.jpg

Example:

- $p(A)$ frequency of observing A
- $p(A,B)$ frequency of observing A and B
- $p(B|A)$ frequency of observing B given A

Properties and definitions

- One can think of events as sets
  - Set operations are defined: $A \cup B, A \cap B, \bar{A} \backslash B$
- $P(A \cup B) = P(A) + P(B)$ if $A \cap B = \emptyset$
- Independence $P(A,B) \equiv P(A \cap B) = P(A)P(B)$
- Conditional probability $P(A|B) = \frac{P(A,B)}{P(B)}$

Bayes theorem

Example:

- We have constructed spam filter that
  - identifies spam mail as spam with probability 0.95
  - Identifies usual mail as spam with probability 0.005
- This kind of spam occurs once in 100,000 mails
- If we found that a letter is a spam, what is the probability that it is actually a spam?

Bayes theorem

- We have some knowledge about event B
  - Prior probability P(B) of B
- We get new information A
  - P(A)
  - P(A|B) probability of A can occur given B has occured
- New (updated) knowledge about B
  - Posterior probability P(B|A)

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

Random variables

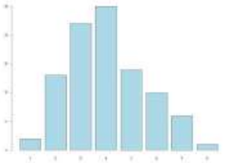
- Instead of having events, we can have a variable X:
  - Events→ℝ Continuous random variables
  - Events→ℕ Discrete random variables

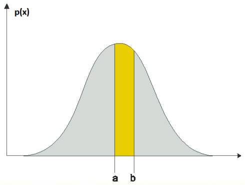
Examples:

- X={amount of times the word ”crisis” can be found in financial documents}
  - P(X=3)
- X={Time to download a specific file to a specific computer}
  - P(X=0.36 min)

Distributions

- Discrete
  - Probability mass function P(x) for all feasible x
- Coninuous
  - Probability density function p(x)
    - $p(x \in [a,b]) = \int_a^b p(x)dx$
    - $p(x) \geq 0, \int_{-\infty}^{+\infty} p(x)dx = 1$
  - Cumulative distribution function $F(x) = \int_0^x p(t)dt$





Expected value and variance

- Expected value = mean value
  - $E(X) = \sum_{i=1}^{n} X_i P(X_i)$
  - $E(X) = \int X p(X) dX$
- Variance how much values of random variable can deviate from mean value
  - $Var(X) = E\left(X - E(X)\right)^2 = E(X^2) - E(X)^2$

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

For random variables:

Bayes Theorem

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

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

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

Some conventional distributions

Bernoulli distribution

- Events: Success (X=1) and Failure (X=0)
- P(X=1)=p, P(X=0)=1-p
- $E(X) = p$
- $Var(X) = 1 - p$

Examples: Tossing coin, vinning a lottery,..

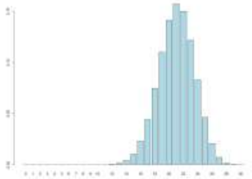
Some conventional distributions

Binomial distribution

- Sequence of *n* Bernoulli events
- X={Amount of successes among these events}, X=0,…,n

$$P(X = r) = \frac{n!}{(n-r)!\,r!}p^r(1-p)^{n-r}$$

- $EX = np$
- $Var(X) = np(1-p)$



Poisson distribution

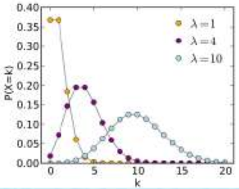
- Customers of a bank **n** (in theory, endless population)
- Probability that a specific person will make a call to the bank between 13.00 and 14.00 a certain day is ***p***
  - ***p*** can be very small if population is large (rare event)
  - Still, some people will make calls between 13.00 and 14.00 that day, and their amount may be quite big
  - A known quantity **λ=*np*** is mean amount of persons that call between 13.00 and 14.00
  - **X**={amount of persons that have called between 13.00 and 14.00}

Poisson distribution

- $P(X = r) = \lim_{n\to\infty} \frac{n!}{(n-r)!r!}p^r(1-p)^{n-r}$
- It can be shown that

$$P(X = r) = \frac{\lambda^r e^{-\lambda}}{r!}$$

- $E(X) = \lambda$
- $Var(X) = \lambda$



Poisson distribution

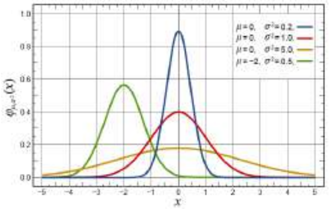
- Further properties:
  - Poisson distribution is a good approximation of the binomial distribution if n >20 and *p* < 0.05
  - Excellent approximation if *n* ≥ 100 and *np* ≤ 10

Normal distribution

- Appears in almost all applications
  - Difference between the times required to download two specific documents to a specific computer

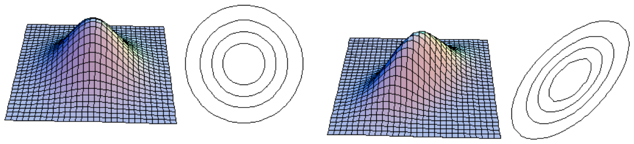
$$f(x) = \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \sigma > 0$$

- $E(X) = \mu$
- $Var(X) = \sigma^2$



Multivariate distributions

- Probability of two variables having certain values at the same time
  - P.D.F. p(x,y)
  - Correlation



Basic ML ingridients

- Data D: observations
  - Features $X_1, .. X_p$
  - Targets $Y_1, \dots, Y_r$

Case	X_1	X_2	Y
1			
2			
…			

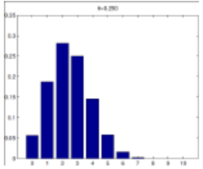
- Model $P(x|\, w_1, \dots w_k)$ or $P(y|x, w_1, \dots w_k)$
  - Example: Linear regression $p(y|x,w) = N(w_0 + w_1 x, \sigma^2)$
- Learning procedure (data→get parameters $\hat{w}$ or $p(w|D)$)
  - Maximum likelihood, Bayesian estimation
- Predict new data X^{new} by using the fitted model

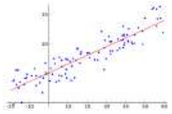
Probabilistic models

- A distribution $p(x|w)$ or $p(y|x,w)$
- Example:
  - $x \sim Bin(n,\theta)$

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

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





Source: Wikipedia

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

Fitting a model

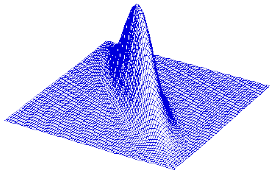
- Given dataset D and model $p(\boldsymbol{x}|w)$ or $p(y|\boldsymbol{x},w)$
  - Frequentist approach: which combination of parameter values fits my data best?
  - Bayesian approach: parameters are random variables, all feasible values are acceptable
    - Different parameter values have different probabilities

Fitting a model

- Frequenist principle: **Maximum likelihood** principle
  - Compute likelihood $p(\boldsymbol{D}|\, w)$

$$p(\boldsymbol{D}|\, w) = \prod_{i=1}^{n} p(X_i|w)$$

$$p(\boldsymbol{D}|\, w) = \prod_{i=1}^{n} p(Y_i|X_i,w)$$



  - Maximize the likelihood and find the optimal w^*

Fitting a model

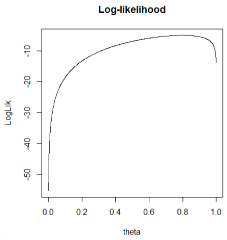
Remarks:

- 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

Example: tossing a coin.

$D = \{0,1,1,0,1,1,1,1,1,1,1,1\}$,
$p(x=1|\theta) = \theta, p(x=0|\theta) = 1-\theta$



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

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

$$\boxed{p(w|D) = \frac{p(D|w)p(w)}{p(D)} \propto p(D|w)p(w)}$$

- $p(D)$ 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 posterior mean θ^*

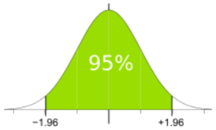
Fitting a model

- 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

      http://en.wikipedia.org/wiki/Conjugate_prior
  - Conjugacy
    - 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 $N \to \infty$, data overwhelms the prior

Measuring uncertainty

- **Confidence interval** (frequentist)
  1. Model $p(x|w)$ is known
  2. $\hat{w}$ is a function of x by ML
  3. Derive distribution of $\hat{w}$
  4. Compute quantiles
- **Credible interval** (Bayes)
- **Prediction interval** (models)



- Example: Prediction interval for $Y \sim N(2x+4, 1)$ at $x = 5$

Regression and regularization

Lecture 1d

Overview

- Linear regression
- Ridge Regression
- Lasso
- Variable selection

Simple linear regression

Model:

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

or

$$y = w_0 + w_1 x + \epsilon,$$
$$\epsilon \sim N(0, \sigma^2)$$

or

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

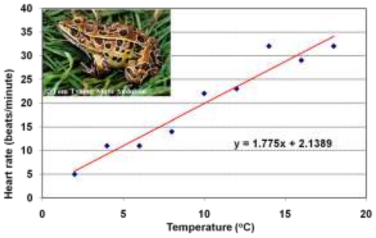
Terminology:

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)

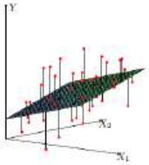
Model:

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

where

$$\boldsymbol{w} = \{w_0, \dots w_d\}$$
$$\boldsymbol{x} = \{1, x_1, \dots x_d\}$$

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

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

Matrix formulation of OLS regression

Optimality condition:

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

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

Parameter estimates and predictions

- Least squares estimates of the parameters

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

- Predicted values $\hat{\boldsymbol{y}} = \boldsymbol{X}\hat{w} = \boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} = \boldsymbol{P}\boldsymbol{y}$

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

Hat matrix

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 $\hat{Y} = S(X)Y$

$$df = trace(S)$$

- For linear regression, degrees of freedom is

$$df = trace(P) = p$$

Different types of features

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

Basis function expansion:

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:

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

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

- In general $\phi_1(\dots)$ may be a function of several x components
- Having data given by **X,** compute new data
- $\Phi = \begin{pmatrix} 1 & \phi_1(x_{11,}\dots,x_{1p}) & .. & \phi_p(x_{11,}\dots,x_{1p}) \\ & \dots & \dots & \dots \\ 1 & \phi_1(x_{n1,}\dots,x_{np}) & \dots & \phi_p(x_{n1,}\dots,x_{np}) \end{pmatrix}$
- If doing a basis function in a model, replace **X** by Φ everywhere where **X** is used:

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

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

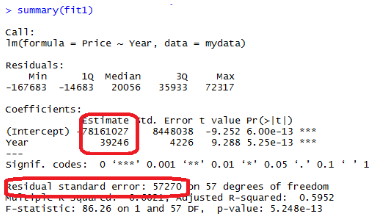
```
summary(fit)
predict(fit, newdata, se.fit, interval)
coefficients(fit) # model coefficients
confint(fit, level=0.95) # CIs for model parameters
fitted(fit) # predicted values
residuals(fit) # residuals
```

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)
```

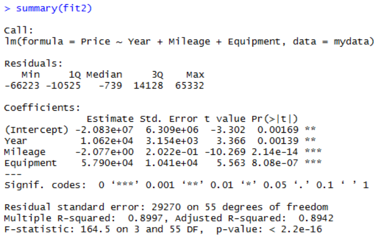
Response variable:
Requested price of used Porsche cars (1000 SEK)

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



```
> 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
```

An example of ordinary least squares regression

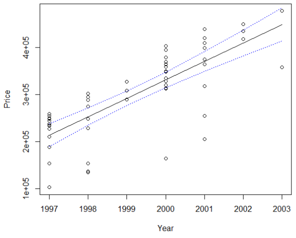


```
> summary(fit2)

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

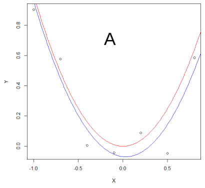
- Prediction

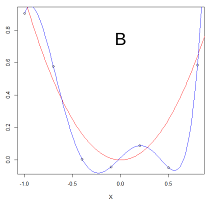
```
fitted <- predict(fit1, interval =
"confidence")

# 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 $Y \coloneqq Y, X_1 = X, X_2 = X^2, \dots, X_p = X^p$ → polinomial 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

minimize $-loglikelihood + \lambda_0\|w\|_2^2$

→ **l_2 regularization**

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

$$\hat{w}^{ridge} = \operatorname{argmin}\left\{\sum_{i=1}^{N}(y_i - w_0 - w_1x_{1j} - \dots - w_px_{pj})^2 + \lambda\sum_{j=1}^{p}w_j^2\right\}$$

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

Ridge regression

Equivalent form

$$\hat{w}^{ridge} = \operatorname{argmin}\sum_{i=1}^{N}(y_i - w_0 - w_1x_{1j} - \dots - w_px_{pj})^2$$

$$\textbf{subject to} \sum_{j=1}^{p} w_j^2 \le s$$

Solution

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

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

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 ortogonal (not realistic), $X^TX = I \rightarrow$

  $\widehat{\boldsymbol{w}}^{\text{ridge}} = \frac{1}{1+\lambda}\mathbf{w}^{\mathbf{linreg}}$ → 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 → shrunk
- Degrees of freedom decrease when λ 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 λ?
  - cross-validation

Ridge regression

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

$$y \sim N(y|w_0 + Xw, \sigma^2 I)$$
$$w \sim N\left(0, \frac{\sigma^2}{\lambda} I\right)$$

Theorem MAP estimate to the Bayesian Ridge is equal to solution in frequenist Ridge

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

- In Bayesian version, we can also make inference about λ

Ridge regression

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

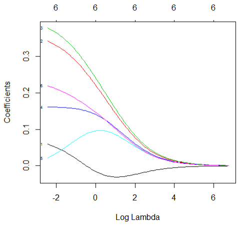
- Cycle time
- Memory
- Channels
- …

Build model predicting performance

Ridge regression

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

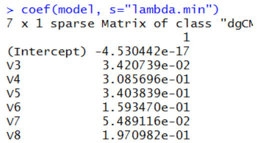
```
data=read.csv("machine.csv", header=F)
covariates=scale(data[,3:8])
response=scale(data[, 9])

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



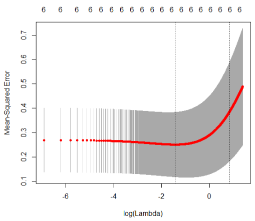
Ridge regression

- 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
```



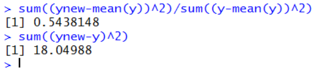
```
> model$lambda.min
 [1] 0.046
```

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)
```

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



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

LASSO

- **Idea**: Similar idea to Ridge
- Minimize minus loglikelihood plus **linear** penalty factor→ **l_1 regularization**
  - Given that model is Gaussian, we get **LASSO** (least absolute shrinkage and selection operator):

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

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

LASSO

- Equivalently

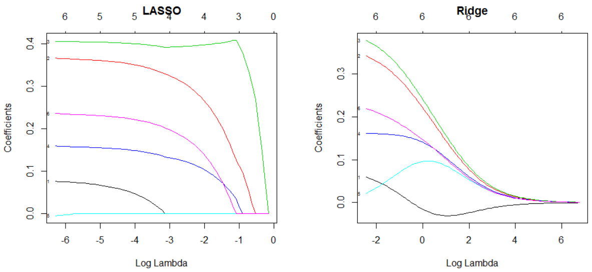
$$\hat{w}^{lasso} = \text{argmin}\sum_{i=1}^{N}(y_i - w_0 - w_1 x_{1j} - \ldots - w_p x_{pj})^2$$

subject to $\sum_{j=1}^{p}|w_j| \le s$

LASSO vs Ridge

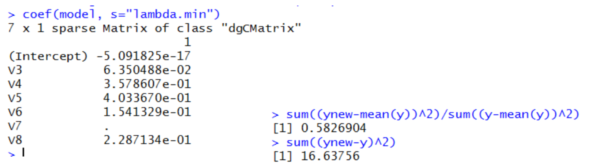
- **LASSO yields sparse solutions!**

Example Computer hardware data



LASSO vs Ridge

- Only 5 variables selected by LASSO

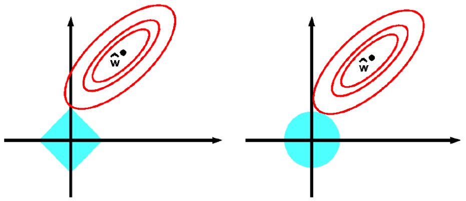


```
> coef(model, s="lambda.min")
7 x 1 sparse Matrix of class "dgCMatrix"
                        1
(Intercept) -5.091825e-17
V3           6.350488e-02
V4           3.578607e-01
V5           4.033670e-01
V6           1.541329e-01
V7           .
V8           2.287134e-01
> 
```

```
> 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 properies

- **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,

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

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

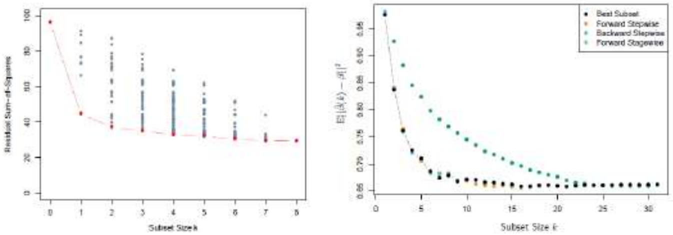
- Model can become overfitted (recall polynomial regression)
- Large number of predictors → model is difficult to use and interpret

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”)

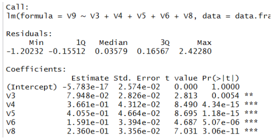
RSS and MSE depend on k



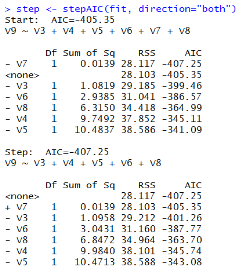
Variable selection in R

- Use stepAIC() in MASS

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



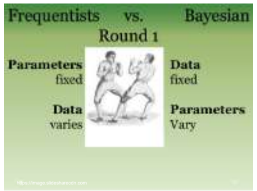
```
Call:
lm(formula = V9 ~ V3 + V4 + V5 + V6 + V8, data = data.fra

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
```

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 $p(w)$??


Frequentists vs. Bayesian

Round 1

Parameters fixed

Data varies

Data fixed

Parameters Vary

https://image.slidesharecdn.com

Example: Linear regression, what are parameters here?

$$y \sim w_0 + \boldsymbol{wx} + e, e \sim N(0, \sigma^2)$$

$$y \sim N(w_0 + \boldsymbol{wx}, \sigma^2)$$

An estimator

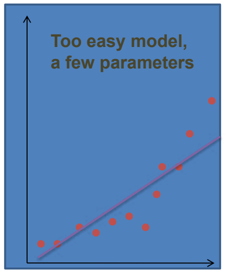
- $\widehat{\boldsymbol{w}} = \delta(D)$ (some function of your data) – an **estimator**
- Optimal parameter values?→there can be many ways to compute them (MLE, shrinkage…)
  - Compare Bayesian: given estimators $\boldsymbol{w}^{\mathbf{1}}$ and $\boldsymbol{w}^{\mathbf{2}}$, we **can** compare them!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

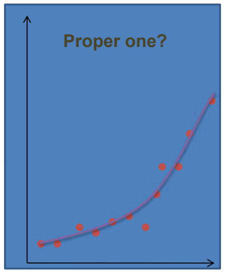
Example: Linear regression

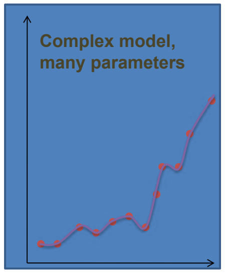
- Estimator 1: $\boldsymbol{w} = (X^T X)^{-1} X^T Y$ (maximum likelihood)
- Estimator 2: $\boldsymbol{w} = (0, \dots, 0, 1)$
- Which one is better?
  - A comparison strategy is needed!

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

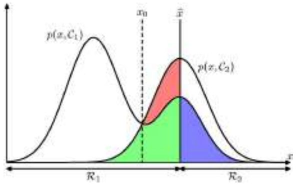
- Given a model, choose the optimal parameter values
  - Decision theory
- Define loss $L(Y, \hat{y})$
  - How much we loose in guessing true Y incorrectly
- If we know the true distribution $p(y, x|w)$ then we choose $\hat{y}$

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

Model selection

Example: Spam classification

- Loss for incorrect classifying mails and spams
  - $L_{12} = 100, L_{21} = 1$



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:

$$L(Y, \hat{y}) = \begin{cases} 10 - \frac{\hat{y}}{Y}, \hat{y} \le Y \\ 1000, \hat{y} > Y \end{cases}$$

Guess why such loss function was chosen

Example: Compute loss function related to

- Normal distribution

Loss functions

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

Model selection

- **Problem**: true model and true $\boldsymbol{w}$ 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

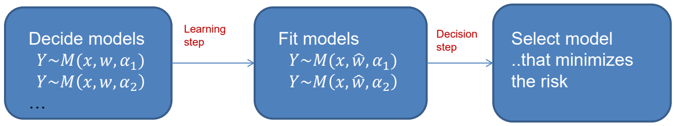
- Simplify the risk estimation:
  - Fix D as a particular training set T
  - Fix Y,X as a particular validation set V
- Risk becomes (**empirical risk**)

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

  - 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 $D = \{\boldsymbol{X}_i, Y_i, i = 1 \dots n\}$



Decide models $Y\sim M(x,w,\alpha_1)$ $Y\sim M(x,w,\alpha_2)$ …

Learning step

Fit models $Y\sim M(x,\hat{w},\alpha_1)$ $Y\sim M(x,\hat{w},\alpha_2)$

Decision step

Select model ..that minimizes the risk

- When fitting data, Maximum Likelihood is usually used
- α_i can be different things:
  - Type of distribution
  - Number of variables in the model
  - Regulatization parameter value
  - …

Holdout method

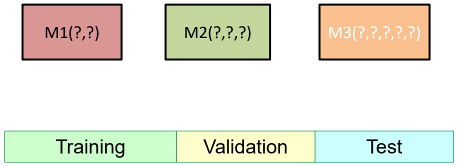
Divide into training, validation and test sets

Training Validation Test

- Choose proportions in some way

Holdout method

- Given: training, validation, test sets and models to select between

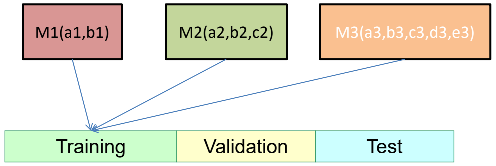


M1(?,?) M2(?,?,?) M3(?,?,?,?,?)

Training Validation Test

Holdout method

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

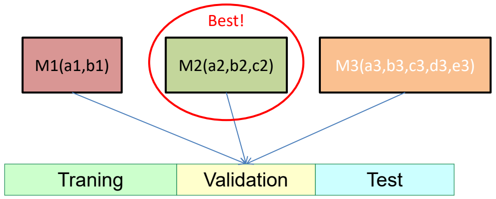


M1(a1,b1) M2(a2,b2,c2) M3(a3,b3,c3,d3,e3)

Training Validation Test

Holdout method

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



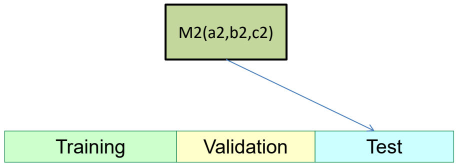
Best!

M1(a1,b1) M2(a2,b2,c2) M3(a3,b3,c3,d3,e3)

Traning Validation Test

Holdout method

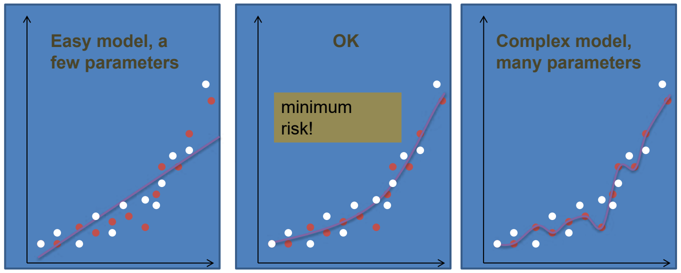
- Test set is used to test a performance on a new data



M2(a2,b2,c2)

Training Validation Test

Holdout method



Easy model, a few parameters

OK

minimum risk!

Complex model, many parameters

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,]
```

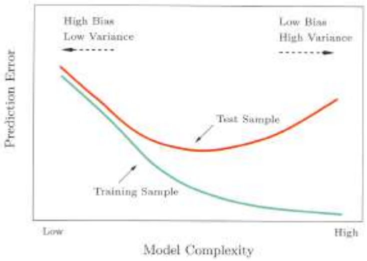
Bias-variance tradeoff

- 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 $L(Y,\hat{y}) = (Y-\hat{y})^2$

$$R\big(Y(x_0),\hat{y}(x_0)\big) = \sigma^2 + Bias^2\big(\hat{y}(x_0)\big) + Var\big(\hat{y}(x_0)\big)$$



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)?

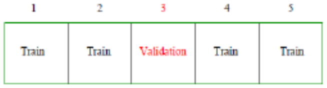
Cross-validation (Estimates Err)

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):

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

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 $R_{train} = \frac{1}{N}\sum_{X_i,Y_i \in T} L(Y_i,\hat{y}_i)$

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

$$AIC \equiv -2loglik(D) + 2df(model)$$

Model selection

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

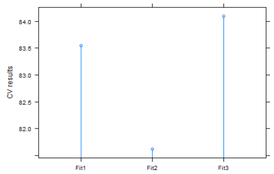
- Cycle time
- Memory
- Channels
- …

Build model predicting performance

Cross-validatation

- 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)
```



Linear classification methods

Lecture 2a

Overview

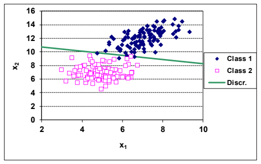
- Elements of decision theory
- Logistic regression
- Discriminant Analysis models

Classification

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

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 $P(Y = C_i|X), i = 1 \dots K$

Disanvantages 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(Spam|A)=0.99, P(Spam|B)=0.45 → better decision can be made

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
  - $p(Y = Spam|x) = 0.83$→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

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

- Example 1: 0/1-loss

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

- Example 2: Spam

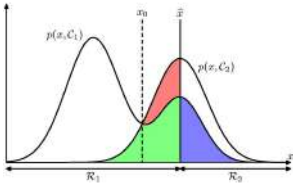
$$L = \begin{pmatrix} 0 & 100 \\ 1 & 0 \end{pmatrix}$$

Loss and decision

- Expected loss minimization
  - R_j : classify to C_j

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

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

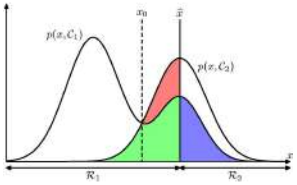


$$EL = \int_{R_1} L_{21} p(x, C_2) dx + \int_{R_2} L_{12} p(x, C_1)\, dx$$

Loss and decision

- Loss minimization

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



When loss is

$$\begin{cases} 1, wrongly\ classified \\ 0, correctly\ classified \end{cases}$$

Classify Y as

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

Loss and decision

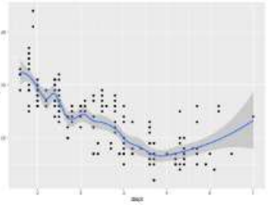
- How to minimize *EL with two classes?*
- Rule:
  - $L_{12} p(x, C_1) > L_{21} p(x, C_2)$ →predict y as C_1
- 0/1 Loss: classify to the class which is more probable!

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

Loss and decision

- Continuous targets: squared loss
  - Given a model $p(x,y)$, minimize

$$EL = \int L\left(y, \hat{Y}(x)\right) p(x,y) dxdy$$



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

		PREDICTED		
T R U E		1	0	Total
	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

$$TPR = TP/N_+$$

- **False Positive Rates** (FPR)
  - Probability of false alarm: system alarms (1) when nothing happens (true=0)

$$FPR = FP/N_-$$

- **Specificity**

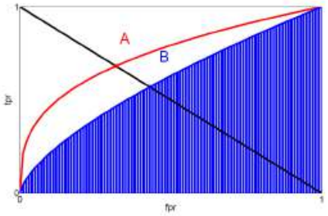
$$Specificity = 1 - FPR$$

- **Precision**

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

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**)



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,

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

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

Generative vs Discriminative

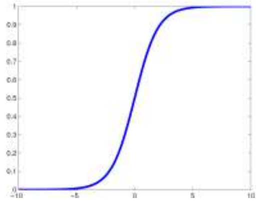
- Generative can be used to generate new data
- Generative normally easier to fit (check Logistic vs K-NN)
- Generative: each class estimated separately→ do not need to retrain when a new class added
- Discriminative models: can replace X with $\phi(X)$ (preprocessing), method will still work
  - Not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

Logistic regression

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

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

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



- Alternatively

$$Y \sim Bernoulli(sigm(a)), a = \boldsymbol{w}^T\boldsymbol{x}$$

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

Logistic regression

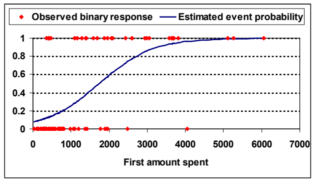
- Logistic model- yet another form

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

The log of the odds is linear in *x*

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

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



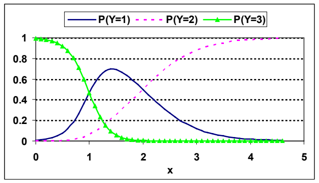
Logistic regression

- When Y is categorical,

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

- Alternatively

$$Y \sim Multinoulli\left(softmax(\boldsymbol{w}_1^T\boldsymbol{x}), \ldots softmax(\boldsymbol{w}_K^T\boldsymbol{x})\right)$$



Logistic regression

Fitting logistic regression

- In binary case,

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

  - 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 $p(y) = [p_1, \ldots p_C]$ and classify as arg $\max_i p_i$

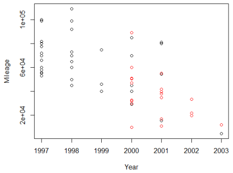
Decision boundaries of logistic regression are linear

Logistic regression

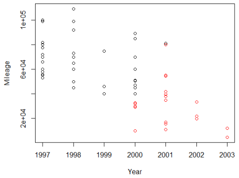
- In R, use glm() with family="binomial"
  - Predicted probabilities: predict(fit,newdata, type="response")

Example Equipment=f(Year, mileage)

Original data



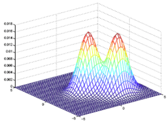
Classified data



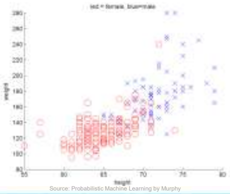
Quadratic discriminant analysis

- Generative classifier
- Main assumptions:
  - $\boldsymbol{x}$ is now **random** as well as y

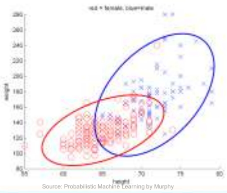
$$p(\boldsymbol{x}|y = C_i, \theta) = N(\boldsymbol{x}|\boldsymbol{\mu_i}, \boldsymbol{\Sigma_i})$$



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



Source: Probabilistic Machine Learning by Murphy

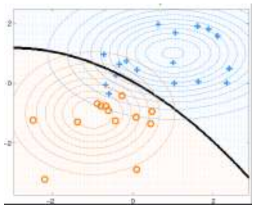


Source: Probabilistic Machine Learning by Murphy

Quadratic discriminant analysis

- If parameters are estimated, classify:

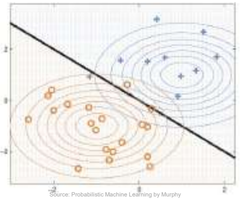
$$\hat{y}(\boldsymbol{x}) = \arg\max_c p(y = c|\boldsymbol{x}, \theta)$$



Source: Probabilistic Machine Learning by Murphy

Linear discriminant analysis (LDA)

- Assumtion $\Sigma_i = \Sigma, i = 1, \ldots K$
- Then $p(y = c_i|x) = softmax(w_i^T x + w_{0i})$ →exactly the same form as the logistic regression
  - $w_{0i} = -\frac{1}{2}\boldsymbol{\mu}_i^T \Sigma^{-1} \boldsymbol{\mu}_i + \log \pi_i$
  - $w_i = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i$



Source: Probabilistic Machine Learning by Murphy

- Decision boundaries are linear
  - **Discriminant function**:

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

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 probabiliies
- 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)
```

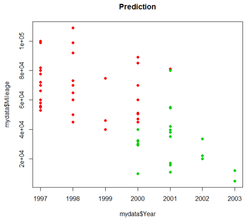
```
> print(resLDA)
Call:
lda(Equipment ~ Mileage + Year, data = mydata)

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

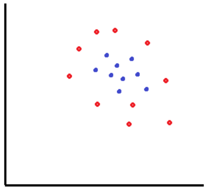
```
plot(mydata$Year, mydata$Mileage,
col=as.numeric(Pred$class)+1, pch=21,
bg=as.numeric(Pred$class)+1,
main="Prediction")
```

```
> 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 Σ → 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.

Naïve Bayes classifiers
Decision trees

Lecture 2b

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=No,1=Yes)
    - X_1 corr. "space", X_2 corr. "fun",…

Acme Article
Technology
Sports
Entertainment

Källa: kdnuggets.com

- Want to classify a new document

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

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

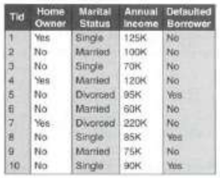
- How many parameters now?
  - $P(X_i = x_i|Y = y), i = 1, \dots p, x_i = \{0,1\}, y = \{0,1\}$ $2 * p$
- Is Naive Bayes assumption always valid?
  - P(Space,ship|SciFi)= P(Space|SciFi)*P(Ship|SciFi) ?

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 θ_{ijk} and π_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 classifiers - discrete inputs

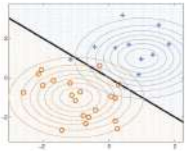
- **Example** Loan decision
  - Classify a person: Home Owner=No, Single=Yes



Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

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 $p(\boldsymbol{x}|y = C_i, \theta) = N(\boldsymbol{x}|\boldsymbol{\mu_i}, \boldsymbol{\Sigma_i})$
  - $\boldsymbol{\Sigma_i}$=diag$(\sigma_{i1}^2, \dots, \sigma_{ip}^2)$

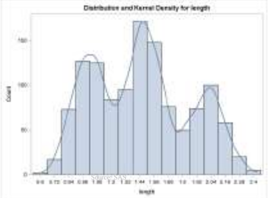


- Naive bayes is a special case of LDA (given A)
  - →MLE are means and variances (per class)

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



Source: SAS

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
```

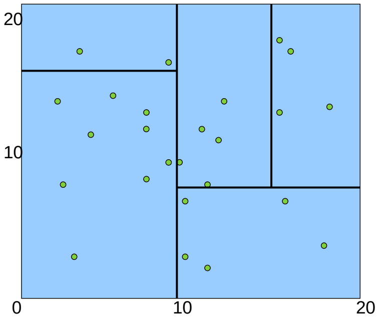
Decision trees

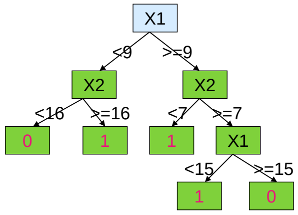
Idea

Split the domain of feature 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

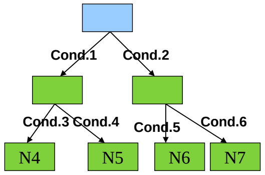
Classification tree toy example





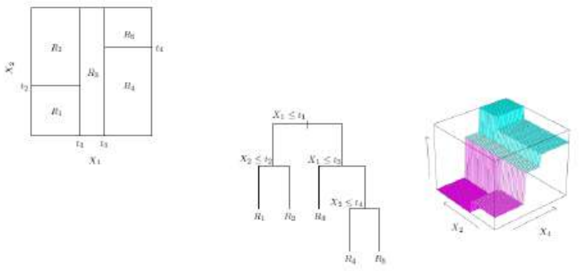
Definitions

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



Cond.1
Cond.2
Cond.3
Cond.4
Cond.5
Cond.6
N4
N5
N6
N7

Regression tree toy example

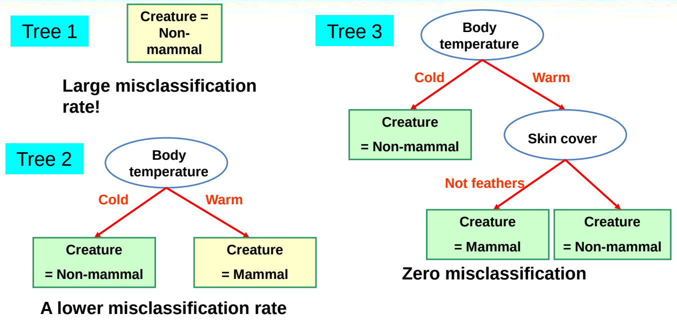


A classification problem

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 2
Body temperature
Cold
Warm
Creature = Non-mammal
Creature = Mammal
A lower misclassification rate

Tree 3
Body temperature
Cold
Warm
Creature = Non-mammal
Skin cover
Not feathers
Creature = Mammal
Creature = Non-mammal
Zero misclassification

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

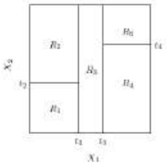
Decision trees

- A tree $T =< r_i, s_{r_i}, R_j, i = 1 \ldots S, j = 1 \ldots L >$
  - $x_{r_i} \le s_{r_i}$ splitting rules (conditions), S- their amount
  - R_j-terminal nodes, L- their amount
  - labels μ_j in each terminal node

Model:

- $Y|T$ for R_j comes from exponential family with mean μ_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 $Y_j|T \sim Multinomial(p_{j1}, \dots p_{jc})$

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_i from C and find a variable X_j and value *s* that split C_j into two hypercubes

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

and solve

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

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**

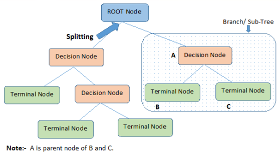
Weakest link pruning:

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

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

$|T|$ =#leaves

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



Note:- A is parent node of B and C.

Source: http://www.analyticsvidhya.com

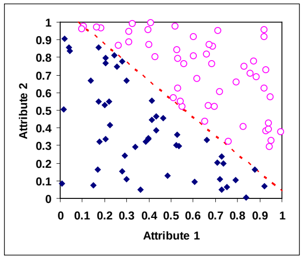
How to find the optimal α? Cross validation!

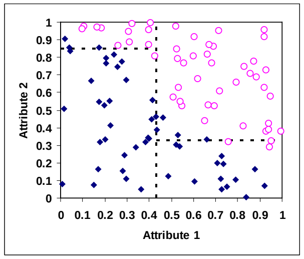
Decision trees: comments

- Similar algorithms work for regression trees – replace $N \cdot Q(R)$ by $SSE(R)$
- Easy to interpret
- Easy to handle all types of features 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"), …)

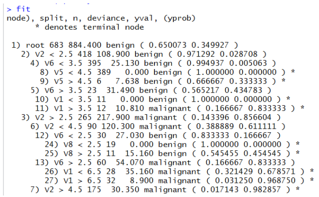
print(), summary(), plot(), text()

Example: breast cancer as a function av biological measurements

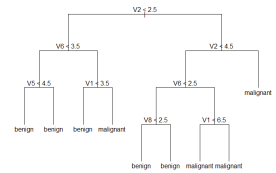
```
library(tree)
n=dim(biopsy)[1]
fit=tree(class~., data=biopsy)
plot(fit)
text(fit, pretty=0)
fit
summary(fit)
```

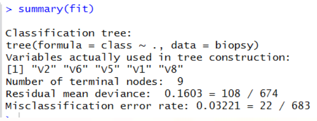
Decision trees in R

- 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 ) *
```



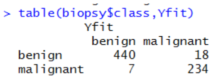


```
> summary(fit)

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)
```

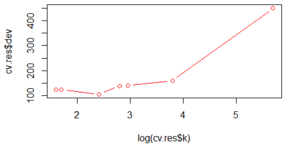


```
> table(biopsy$class,Yfit)
           Yfit
            benign malignant
  benign       440        18
  malignant      7       234
```

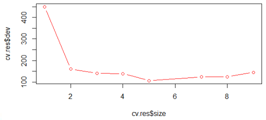
Decision trees in R

- 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")
```



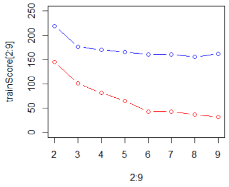
What is optimal number of leaves?



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")
```

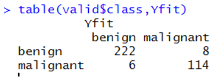


What is optimal number of leaves?

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
```

Generalized Linear Models. Uncertainty estimation

Lecture 2c

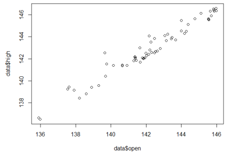
Moving beyond typical distributions

- We know how to model
  - Normally distributed targets -> linear regression
  - Bernoulli and Multinomial targets→logistic regression
  - What if target 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…

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

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

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

- If f is a canonical link, then

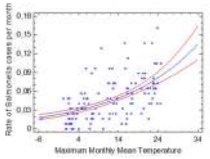
$$p(y|w,s) = h(y,s)g(\boldsymbol{w},\boldsymbol{x})e^{\frac{(\boldsymbol{w}^T\boldsymbol{x})y}{s}}$$

Generalized linear models

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

Example: Poisson regression

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



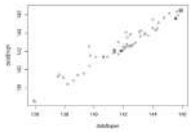
Generalized linear model: software

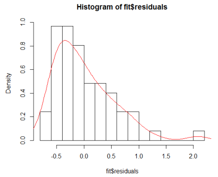
- Use **glm**(formula, family, data) in R

Example: Daily Stock prices NASDAQ

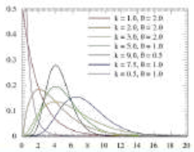
- Open
- High (within day)

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



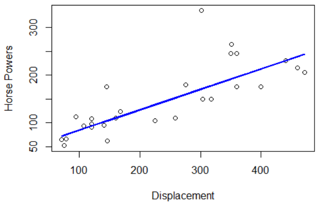


Gamma distribution: Wikipedia



Least absolute deviation regression

- Model $Y \sim Laplace(w^TX, b)$
  - Member of exponential family
- Equivalent to minimizing sum of absolute deviations
- Properties
  - Robust to outliers
  - Sensitive to changes in data
  - Multiple solutions possible



- R: package **L1pack**

Probabilistic models

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

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

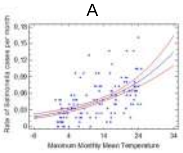
- f is known, w is unknown
- Fit model to data with least squares, optimization or ad hoc→ find w

Probabilistic models

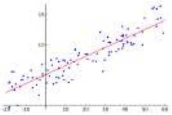
Arguments against deterministic models:

- 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 …” 😮

A



B



Probabilistic models

Probabilistic model

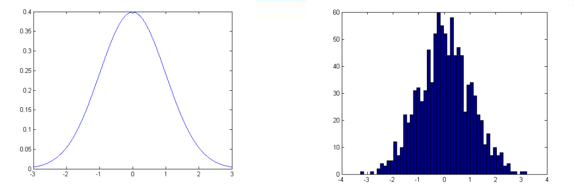
$$Y \sim Distribution(f(x,w),\theta)$$

- 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 $\hat{\boldsymbol{\alpha}} = \delta(D)$), 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

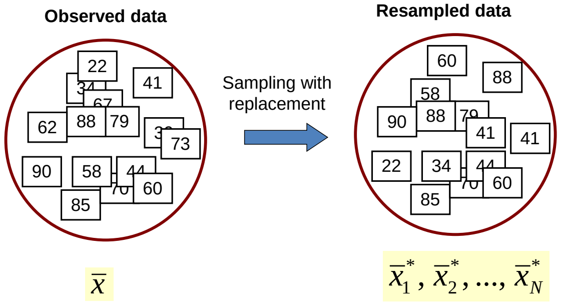


We want to determine uncertainty of $\hat{f}(D, X)$

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



Observed data

Sampling with replacement

Resampled data

$\overline{x}$

$\overline{x}_1^*, \overline{x}_2^*, \ldots, \overline{x}_N^*$

Nonparametric bootstrap

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

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

1. Estimate $\widehat{w}$ from data **D**=(X_1,…X_n)
2. Generate $\mathbf{D_1}$ =(X^*_1,…X^*_n) by sampling with replacement
3. Repeat step 2 B times
4. The distribution of w is given by $\hat{f}(D_1), \ldots \hat{f}(D_B)$

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

Parametric bootstrap

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

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

1. Estimate $\widehat{w}$ from data **D**=(X_1,…X_n)
2. Generate $\mathbf{D_1}$ =(X^*_1,…X^*_n) by generating from $F(X, \widehat{w})$
3. Repeat step 2 B times
4. The distribution of w is given by $\hat{f}(D_1), \ldots \hat{f}(D_B)$

Parametric bootstrap is **more** precise if the distribution form is correct

Uncertainty estimation

1. Get D_1 , … D_B by bootstrap
2. Use $\hat{f}(D_1$), … $\hat{f}(D_B$) to estimate the uncertainty
   - Boostrap 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

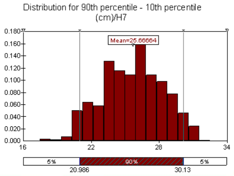
- To estimate 100(1-α) confidence interval for w

Bootstrap percentile method

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

$$\left(w_{A_1}, w_{A_2}\right)$$

Look at the plot…


Distribution for 90th percentile - 10th percentile (cm)/H7

Bootstrap: regression context

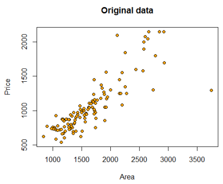
- Model $Y \sim F(X, w)$
- Data $\mathrm{D} = \{(Y_i, X_i), i = 1, \ldots, n\}$
- Idea: produce several bootstrap sets that are similar to D

Nonparametric bootstrap:

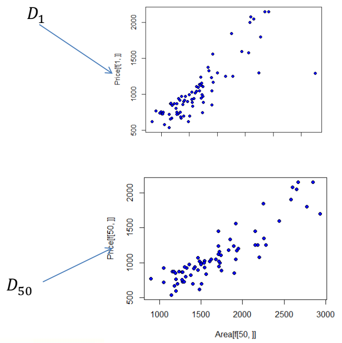
1. Using observation set **D**, sample **pairs** (X_i, Y_i) with replacement and get bootstrap sample $\mathbf{D_1}$
2. Repeat step 1 B times→ get $\mathbf{D_{1,\ldots}}$ $\mathbf{D_B}$

Uncertainty estimation

Example: Albuquerque dataset:
Y=Price of House
X=Area (sqft)



We sample data index, from {1…N}



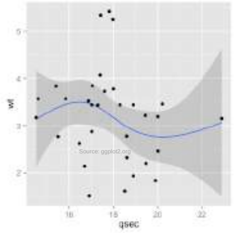
Bootstrap: regression context

Parametric bootstrap

1. Fit a model to D → 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

Confindence intervals in regression

- Given $Y \sim Distribution(y|x,w), EY|X = \mu|x = f(x,w)$
  - 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?
  - $\mu|x = f(x,w)$

Confindence intervals in regression

Estimation

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

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,…)

Bootstrap: R

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

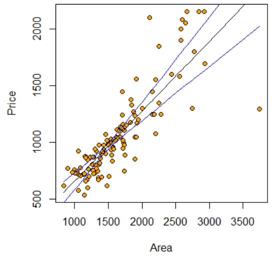
Bootstrap

```
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 model
  #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 cofidence 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 cofidence 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 $Y|X$

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

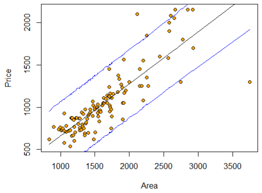
Prediction band for parametric bootstrap

1. Run parametric bootstrap and get $D_1, \dots D_B$
2. Fit the model to the data and get $\widehat{w}(D_1), \dots \widehat{w}(D_B)$
3. For each X, generate from $Distribution\big(X,\widehat{w}(D_1)\big), \dots$ $Distribution\big(X,\widehat{w}(D_B)\big)$ 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?

Lecture 2d

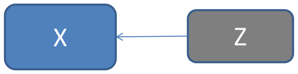
Latent variable models

Overview

- Principal Component Analysis (PCA)
- Probabilistic PCA
- Independent component analysis (ICA)

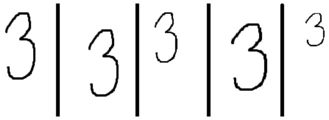
Latent variables

- Sometimes data depends on the variables 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



Latent variables

- Latent factor discovered→ data storage may decrease a lot



- Latent factors
  - Center
  - Scaling
- Original vs compressed
  - 100x100x5=50000
  - 100x100+2*5+2*5=10020

Principal Component Analysis (PCA)

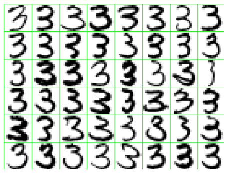
- *PCA* is a technique for reducing the complexity of high dimensional data
- It can be used to approximate high dimensional data with a few dimensions (latent features) –> much less data to store
- New variables might have a special interpretation

Applications

- Image recognition
- Information compression
- Subspace clustering
- …

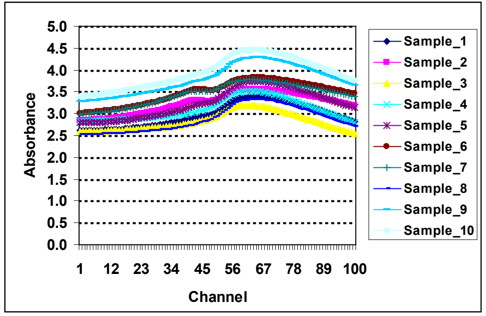
Principal Component Analysis (PCA)

- Example 1: Hadwritten digits
  - Can we get a more compact summary?



Absorbance records for ten samples of chopped meat

Parallel coordinate plot for “FAT”



1 target (fat)

100 features (absorbance at 100 wavelengths or channels)

The features are strongly correlated to each other

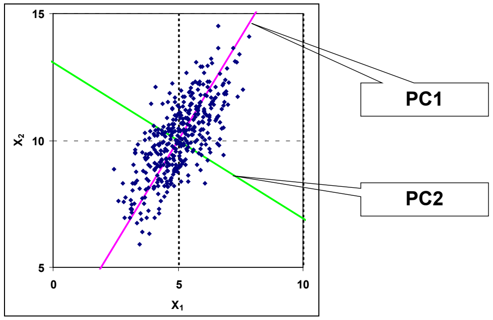
Principal components analysis

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

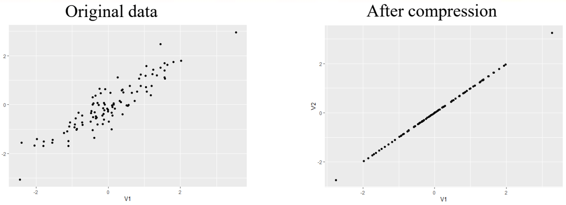
- The first principal component (PC1) is the direction that maximizes the variance of the projected data
- The second principal component (PC2) is the direction that maximizes the variance of the projected data after the variation along PC1 has been removed
- The third principal component (PC3) is the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

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

Principal Component Analysis - two inputs



PCA- after reducing dimensionality



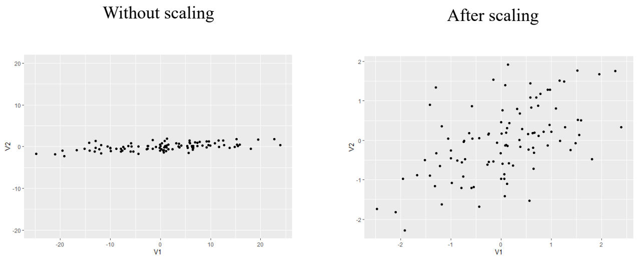
Original data

After compression

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

PCA and scaling

- Do we need to scale features?



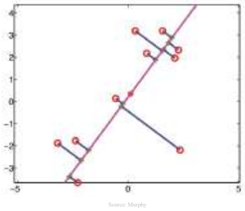
Without scaling

After scaling

PCA: another view

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

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



PCA: computations

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

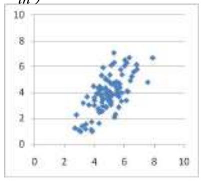
1. Centred data

$$X = \left\| \mathbf{x}_1 - \overline{\mathbf{x}}_1 \ \mathbf{x}_2 - \overline{\mathbf{x}}_2 \dots \mathbf{x}_p - \overline{\mathbf{x}}_p \right\|,$$

2. Covariance matrix

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

3. Search for eigenvectors and eigenvalues of **S**

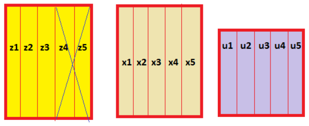


	Column 1	*Column 2*
Column 1	0.951	0.905
Column 2	0.905	1.883

PCA: computations

4. Coordinates 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:*
$Z = X\ U_M$

Store: N x M+ p x M
instead N x p

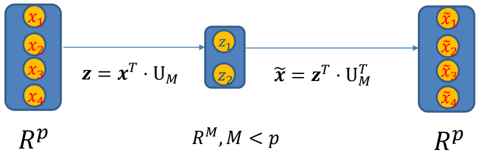
6. New data will have dimensions N x M instead of N x p

Getting approximate original data:
$\tilde{X} = ZU_M^T$

100*50 vs
100*4+50*4

PCA: computations

- PCA makes a **linear** compression of features



$\boldsymbol{z} = \boldsymbol{x}^T \cdot \mathrm{U}_M$

$\tilde{\boldsymbol{x}} = \boldsymbol{z}^T \cdot \mathrm{U}_M^T$

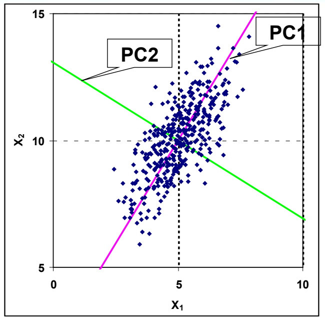
R^p

$R^M, M < p$

R^p

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

Principal Component Analysis



Eigenanalysis of the Covariance Matrix

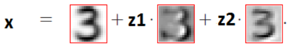
Eigenvalue	2.8162	0.3835
Proportion	0.880	0.120
Cumulative	0.880	1.000

Variable	PC1	PC2
X1	0.523	0.852
X2	0.852	-0.523

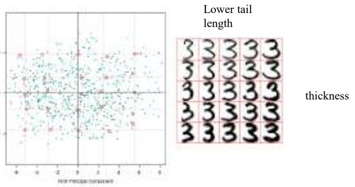
Loadings (U)

Principal Component Analysis

- Digits: two eigenvectors extracted



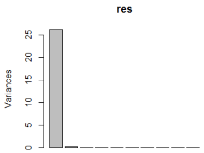
- Interptretation of eigenvectiors



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-
 [7] 2.052212e-04 1.084213e-04 2.077326e-05 1.150359e-
```

```
> 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

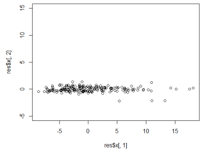
- Principal component **loadings (U)**

```
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 0.08184896 0.1227491 0.07125048 -0.1156
```

- Data in (PC1, PC2) – **scores (Z)**

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

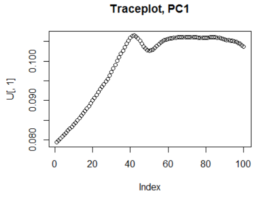


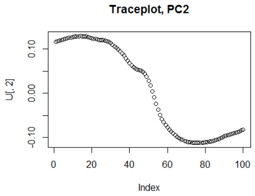
Do we need second dimension?

PCA in R

- Trace plots

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

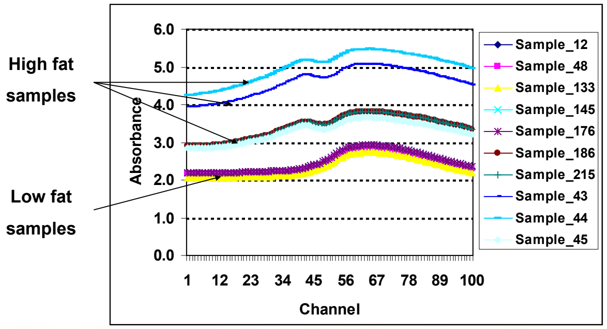




Which components contribute to PC1-2?

Absorbance records for ten samples of chopped meat

PCA2 captures the most of remaining variation



Probabilistic PCA

- z_i-latent variables, x_i- observed variables

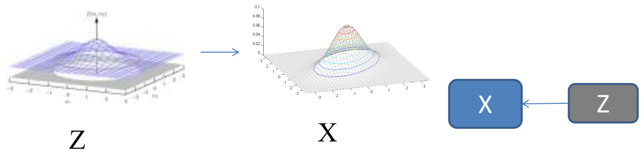
$$\boldsymbol{z} \sim \boldsymbol{N}(0, \boldsymbol{I})$$

$$\boldsymbol{x}|\boldsymbol{z} \sim \boldsymbol{N}(\boldsymbol{x}|\boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu}, \sigma^2 \boldsymbol{I})$$

- Alternatively

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

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



Probabilistic PCA

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

$$\boldsymbol{x} \sim \boldsymbol{N}(\boldsymbol{\mu}, \boldsymbol{C})$$

$$\boldsymbol{C} = \boldsymbol{W}\boldsymbol{W}^{\boldsymbol{T}} + \boldsymbol{\sigma}^2 \boldsymbol{I}$$

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

$$\boldsymbol{x}|\boldsymbol{z}' \sim \boldsymbol{N}(\boldsymbol{x}|\boldsymbol{W}\boldsymbol{z}' + \boldsymbol{\mu}, \sigma^2 \boldsymbol{I})$$

  - **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

$$\mu_{ML} = \bar{x}$$

$$W_{ML} = U_M (L_M - \sigma_{ML}^2 I)^{\frac{1}{2}} R$$

$$\sigma_{ML}^2 = \frac{1}{p-M} \sum_{i=M+1}^{p} \lambda_i$$

- 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 $R = I, \sigma^2 = 0$ → 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<<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

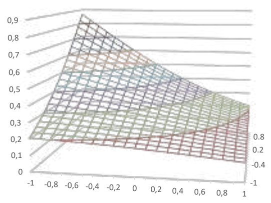
- Use **pcaMethods** from Bioconductor
- Install
  - source("https://bioconductor.org/biocLite.R")
  - biocLite("pcaMethods")

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_i)
- Assuming latent features are **independent**



$$p(z) = \prod_{i=1}^{M} p(z_i) \qquad p(z_i) = \frac{2}{\pi(e^{z_i} + e^{-z_i})}$$

ICA

- Model

$$x = \mu + Wz + \epsilon, \qquad \epsilon \sim \boldsymbol{N}(0, \Sigma)$$

- **Estimation : Maximum likelihood** $(V = W^{-1})$
  - Assuming noise-free x

$$\max_{V} \sum_{i=1}^{n} \sum_{j=1}^{p} \log\left(p_j(v_j^T x_i)\right)$$

Subject to $||v_i|| = 1$

ICA: estimation algorithm

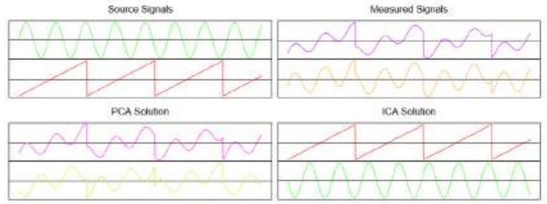
1. Estimate V by maximum likelihood
2. Compute $Z = X'V$

- **With prewhitening**
  1. Convert X into PCA coordinate system (do not remove dimensions): $X' = XU$
  2. Estimate V by maximum likelihood in ICA
  3. Estimate final scores $Z = X'V$

  – Note: full transformation matrix is $U_{ICA} = U \cdot V$

ICA

- Example

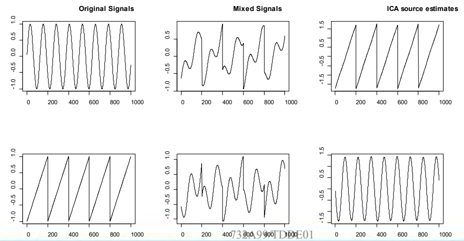


Source: Elem of stat learn by Hastie

Independent component analysis: R

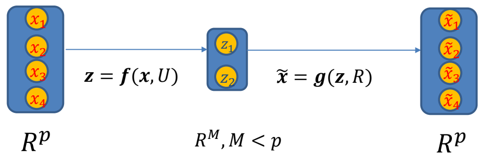
R package: fastICA

```
S <- cbind(sin((1:1000)/20), rep((((1:200)-100)/100), 5))
A <- matrix(c(0.291, 0.6557, -0.5439, 0.5572), 2, 2)
X <- S %*% A #mixing signals
a <- fastICA(X, 2) #now separate them
```



Autoencoders (nonlinear PCA)

- Why linear transformations? Take nonlinear instead!
- $f()$ and $g()$ are typically Neural Networks



x_1
x_2
x_3
x_4
$\boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x}, U)$
z_1
z_2
$\tilde{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{z}, R)$
$\tilde{x}_1$
$\tilde{x}_2$
$\tilde{x}_3$
$\tilde{x}_4$
R^p
$R^M, M < p$
R^p

$$\min_{U,R} \sum_{i=1}^{N} \|x_n - \tilde{x}_n\|^2$$

…or some other loss function

Histogram Classification

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

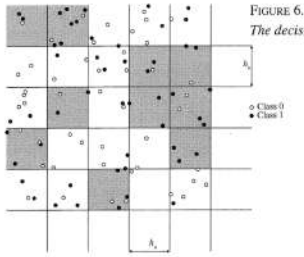


FIGURE 6.1. *A cubic histogram rule: The decision is 1 in the shaded area.*

- In other words,

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

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.

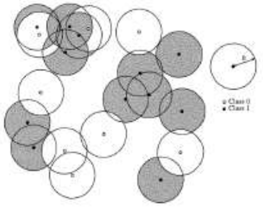


FIGURE 10.1. *The moving window rule in $\mathcal{R}^2$. The decision is 1 in the shaded area.*

- In other words,

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

where $S(\boldsymbol{x}, h)$ is a D-dimensional closed ball of radius h centered at $\boldsymbol{x}$.

Kernel Classification

- The moving window rule gives equal weight to all the points in the ball, which may be counterintuitive. Then,

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

where $k : \mathbb{R}^D \to \mathbb{R}$ 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.

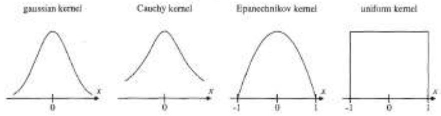


FIGURE 10.3. *Various kernels on $\mathcal{R}$.*

- Gaussian kernel: $k(u) = exp(-||u||^2)$ where $||\cdot||$ is the Euclidean norm.
- 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

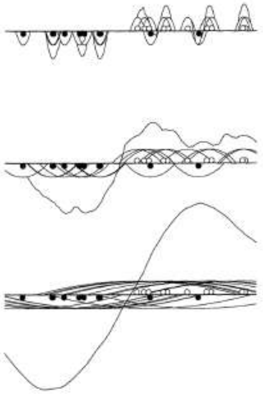


FIGURE 10.2. *Kernel rule on the real line. The figure shows $\sum_{i=1}^n (2Y_i - 1)K((x - X_i)/h)$ for $n = 20$, $K(u) = (1 - u^2)I_{\{|u| \leq 1\}}$ (the Epanechnikov kernel), and three smoothing factors h. One definitely undersmooths and one oversmooths. We took $p = 1/2$, and the class-conditional 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 $y^*(\boldsymbol{x}) = \mathbb{E}_Y[y|\boldsymbol{x}]$.
- Since $\boldsymbol{x}$ may not appear in the finite training set $\{(\boldsymbol{x}_n, t_n)\}$ available, then we average over the points in $C(\boldsymbol{x}, h)$ or $S(\boldsymbol{x}, h)$, or kernel-weighted average over all the points.
- In other words,

$$y_C(\boldsymbol{x}) = \frac{\sum_{\boldsymbol{x}_n \in C(\boldsymbol{x},h)} t_n}{|\{\boldsymbol{x}_n \in C(\boldsymbol{x}, h)\}|}$$

or

$$y_S(\boldsymbol{x}) = \frac{\sum_{\boldsymbol{x}_n \in S(\boldsymbol{x},h)} t_n}{|\{\boldsymbol{x}_n \in S(\boldsymbol{x}, h)\}|}$$

or

$$y_k(\boldsymbol{x}) = \frac{\sum_n k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right) t_n}{\sum_n k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right)}$$

Histogram, Moving Window, and Kernel Density Estimation

- Consider density estimation for a D-dimensional continuous random variable.
- Let $R \subseteq \mathbb{R}^D$ and $\boldsymbol{x} \in R$. Then,

$$P = \int_R p(\boldsymbol{x}) d\boldsymbol{x} \simeq p(\boldsymbol{x}) Volume(R)$$

and the number of the N training points $\{\boldsymbol{x}_n\}$ that fall inside R is

$$|\{\boldsymbol{x}_n \in R\}| \simeq P\, N$$

and thus

$$p(\boldsymbol{x}) \simeq \frac{|\{\boldsymbol{x}_n \in R\}|}{N\, Volume(R)}$$

- Then,

$$p_C(\boldsymbol{x}) = \frac{|\{\boldsymbol{x}_n \in C(\boldsymbol{x}, h)\}|}{N\, Volume(C(\boldsymbol{x}, h))}$$

or

$$p_S(\boldsymbol{x}) = \frac{|\{\boldsymbol{x}_n \in S(\boldsymbol{x}, h)\}|}{N\, Volume(S(\boldsymbol{x}, h))}$$

or

$$p_k(\boldsymbol{x}) = \frac{1}{N} \sum_n k\left(\frac{\boldsymbol{x} - \boldsymbol{x}_n}{h}\right)$$

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 Δ are shown for various values of Δ.

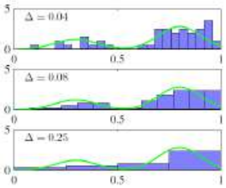
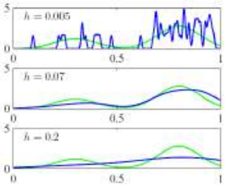


Figure 2.25 Illustration of the kernel density model (2.250) applied to the same data set used to demonstrate the histogram approach in Figure 2.24. We see that h acts as a smoothing parameter and that if it is set too small (top panel), the result is a very noisy density model, whereas if it is set 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

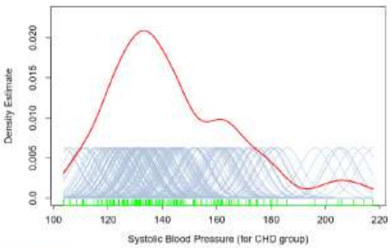


FIGURE 6.13. *A kernel density estimate for systolic blood pressure (for the CHD group). The density estimate at each point is the average contribution from each of the kernels at that point. We have scaled the kernels down by a factor of 10 to make the graph readable.*

Histogram, Moving Window, and Kernel Density Estimation

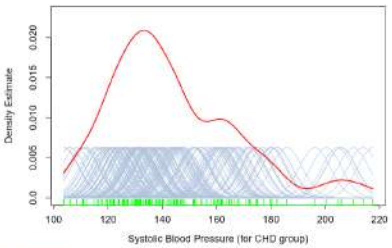


FIGURE 6.13. *A kernel density estimate for systolic blood pressure (for the CHD group). The density estimate at each point is the average contribution from each of the kernels at that point. We have scaled the kernels down by a factor of 10 to make the graph readable.*

- From kernel density estimation to kernel classification:
  1. Estimate $p(\boldsymbol{x}|y = 0)$ and $p(\boldsymbol{x}|y = 1)$ using the methods just seen.
  2. Estimate $p(y)$ as class proportions.
  3. Compute $p(y|\boldsymbol{x}) \propto p(\boldsymbol{x}|y)p(y)$ by Bayes theorem.

Histogram, Moving Window, and Kernel Density Estimation

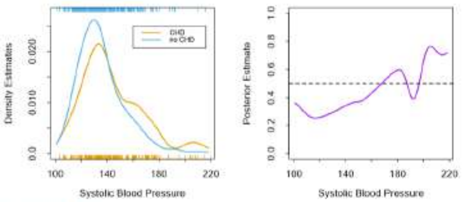
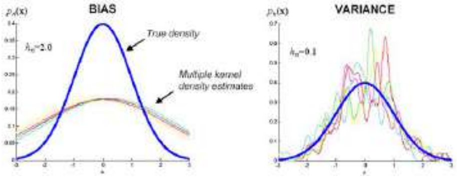


FIGURE 6.14. *The left panel shows the two separate density estimates for systolic blood pressure in the CHD versus no-CHD groups, using a Gaussian kernel density estimate in each. The right panel shows the estimated posterior probabilities for CHD, using (6.25).*

Kernel Selection

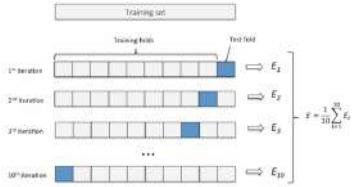
- How to choose the right kernel and width ? E.g., by cross-validation.
- What does “right” mean ? E.g., minimize loss function.
- Note that the width of the kernel corresponds to a bias-variance trade-off.



- 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 $\boldsymbol{x}$.
- 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.

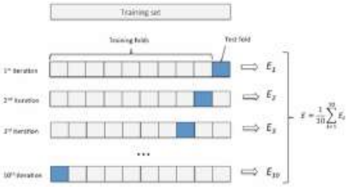


- If the training set contains N points, note that cross-validation estimates the prediction error when the model is trained on $N - N/K$ points.
- Note that the model returned is trained on N points. So, cross-validation overestimates the prediction error of the model returned.
- This seems to suggest 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.

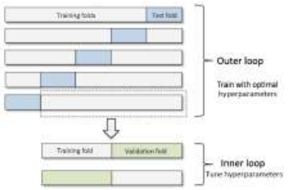
Kernel Selection

- Model: For example, ridge regression with a given value for the penalty factor λ. Only the parameters (weights) need to be determined (closed-form solution).
- Model selection: For example, determine the value for the penalty factor λ. 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



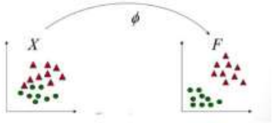
Nested cross-validation for estimating model **selection** prediction error



- 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

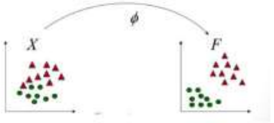
- The kernel function $k\left(\frac{\boldsymbol{x}-\boldsymbol{x}'}{h}\right)$ is invariant to translations, and it can be generalized as $k(\boldsymbol{x}, \boldsymbol{x}')$. For instance,
  - Polynomial kernel: $k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T\boldsymbol{x}' + c)^M$
  - Gaussian kernel: $k(\boldsymbol{x}, \boldsymbol{x}') = exp(-||\boldsymbol{x} - \boldsymbol{x}'||^2/2\sigma^2)$
- If the matrix
$$\begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \vdots & \dots & \vdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{pmatrix}$$
is symmetric and positive semi-definite for all choices of $\{\boldsymbol{x}_n\}$, then $k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T\phi(\boldsymbol{x}')$ 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,
$$\phi(\boldsymbol{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2c}x_1, \sqrt{2c}x_2, c)$$
for the polynomial kernel with $M = D = 2$.

Kernel Trick

- Consider again moving window classification, regression, and density estimation.
- Note that $\boldsymbol{x}_n \in S(\boldsymbol{x}, h)$ if and only if $||\boldsymbol{x} - \boldsymbol{x}_n|| \le h$.
- Note that
$$||\boldsymbol{x} - \boldsymbol{x}_n|| = \sqrt{(\boldsymbol{x} - \boldsymbol{x}_n)^T(\boldsymbol{x} - \boldsymbol{x}_n)} = \sqrt{\boldsymbol{x}^T\boldsymbol{x} + \boldsymbol{x}_n^T\boldsymbol{x}_n - 2\boldsymbol{x}^T\boldsymbol{x}_n}$$
- Then,
$$\begin{aligned} ||\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}_n)|| &= \sqrt{\phi(\boldsymbol{x}^T)\phi(\boldsymbol{x}) + \phi(\boldsymbol{x}_n^T)\phi(\boldsymbol{x}_n) - 2\phi(\boldsymbol{x}^T)\phi(\boldsymbol{x}_n)} \\ &= \sqrt{k(\boldsymbol{x}, \boldsymbol{x}) + k(\boldsymbol{x}_n, \boldsymbol{x}_n) - 2k(\boldsymbol{x}, \boldsymbol{x}_n)} \end{aligned}$$
- So, the distance is now computed in a (hopefully) more convenient space.



- Note that we do not need to compute $\phi(\boldsymbol{x})$ and $\phi(\boldsymbol{x}_n)$.

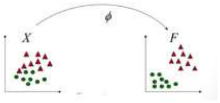
Support Vector Machines for Classification

- Consider binary classification with input space $\mathbb{R}^D$.
- Consider a training set $\{(\boldsymbol{x}_n, t_n)\}$ where $t_n \in \{-1,+1\}$.
- Consider using the linear model

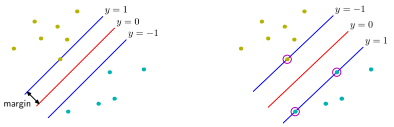
$$y(\boldsymbol{x}) = \boldsymbol{w}^T\phi(\boldsymbol{x}) + b$$

so that a new point $\boldsymbol{x}$ is classified according to the sign of $y(\boldsymbol{x})$.

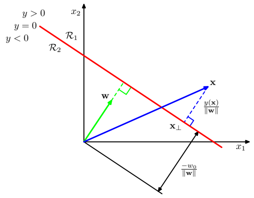
- Assume that the training set is linearly separable in the feature space (but not necessarily in the input space), i.e. $t_n y(\boldsymbol{x}_n) > 0$ for all n.



- 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

$$\frac{t_n y(\boldsymbol{x}_n)}{\|\boldsymbol{w}\|} = \frac{t_n(\boldsymbol{w}^T\phi(\boldsymbol{x}_n) + b)}{\|\boldsymbol{w}\|}$$

- Then, the maximum margin separating hyperplane is given by

$$\arg\max_{\boldsymbol{w},b}\Big(\min_n \frac{t_n(\boldsymbol{w}^T\phi(\boldsymbol{x}_n) + b)}{\|\boldsymbol{w}\|}\Big)$$

- Multiply $\boldsymbol{w}$ and b by κ so that $t_n(\boldsymbol{w}^T\phi(\boldsymbol{x}_n) + b) = 1$ for the point closest to the hyperplane. Note that $t_n(\boldsymbol{w}^T\phi(\boldsymbol{x}_n) + b)/\|\boldsymbol{w}\|$ does not change.

Support Vector Machines for Classification

- Then, the maximum margin separating hyperplane is given by

$$\arg\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2$$

subject to $t_n(\boldsymbol{w}^T\phi(\boldsymbol{x}_n) + b) \geq 1$ for all n.

- To minimize the previous expression, we minimize

$$\frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_n a_n\big(t_n(\boldsymbol{w}^T\phi(\boldsymbol{x}_n) + b) - 1\big)$$

where $a_n \geq 0$ are called Lagrange multipliers.

- Note that any stationary point of the Lagrangian function is a stationary point of the original function subject to the constraints. Moreover, the Lagrangian function is a quadratic function subject to linear inequality constraints. Then, it is concave, actually concave up because of the +1/2 and, thus, "easy" to minimize.
- Note that we are now minimizing with respect to $\boldsymbol{w}$ and b, and maximizing with respect to a_n.
- Setting its derivatives with respect to $\boldsymbol{w}$ and b to zero gives

$$\boldsymbol{w} = \sum_n a_n t_n \phi(\boldsymbol{x}_n)$$

$$0 = \sum_n a_n t_n$$

Support Vector Machines for Classification

- Replacing the previous expressions in the Lagrangian function gives the dual representation of the problem, in which we maximize

$$\sum_n a_n - \frac{1}{2}\sum_n\sum_m a_n a_m t_n t_m \phi(\boldsymbol{x}_n)^T\phi(\boldsymbol{x}_m) = \sum_n a_n - \frac{1}{2}\sum_n\sum_m a_n a_m t_n t_m k(\boldsymbol{x}_n, \boldsymbol{x}_m)$$

subject to $a_n \geq 0$ for all n, and $\sum_n a_n t_n = 0$.

- Again, this "easy" to maximize.
- Note that the dual representation makes use of the kernel trick, i.e. it allows working in a more convenient feature space without constructing it.

Support Vector Machines for Classification

- When the Lagrangian function is maximized, the Karush-Kuhn-Tucker condition holds for all n:

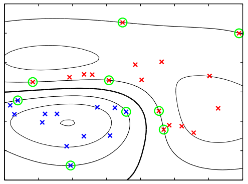
$$a_n(t_n y(\boldsymbol{x}_n) - 1) = 0$$

- Then, $a_n > 0$ if and only if $t_n y(\boldsymbol{x}_n) = 1$. The points with $a_n > 0$ are called support vectors and they lie on the margin boundaries.
- A new point $\boldsymbol{x}$ is classified according to the sign of

$$y(\boldsymbol{x}) = \boldsymbol{w}^T\phi(\boldsymbol{x}) + b = \sum_n a_n t_n \phi(\boldsymbol{x}_n)^T\phi(\boldsymbol{x}) + b = \sum_n a_n t_n k(\boldsymbol{x}, \boldsymbol{x}_n) + b$$

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

where $\mathcal{S}$ are the indexes of the support vectors. Sparse solution!



Support Vector Machines for Classification

- To find b, consider any support vector $\boldsymbol{x}_n$. Then,

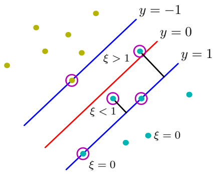
$$1 = t_n y(\boldsymbol{x}_n) = t_n\Big(\sum_{m\in\mathcal{S}} a_m t_m k(\boldsymbol{x}_n, \boldsymbol{x}_m) + b\Big)$$

and multiplying both sides by t_n, we have that

$$b = t_n - \sum_{m\in\mathcal{S}} a_m t_m k(\boldsymbol{x}_n, \boldsymbol{x}_m)$$

- 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

$$\xi_n = \begin{cases} 0 & \text{if } t_n y(\boldsymbol{x}_n) \geq 1 \\ |t_n - y(\boldsymbol{x}_n)| & \text{otherwise} \end{cases}$$

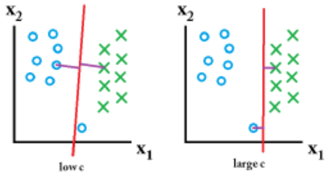


Support Vector Machines for Classification

- The optimal separating hyperplane is given by

$$\arg\min_{\boldsymbol{w},b,\{\xi_n\}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_n \xi_n$$

subject to $t_n y(\boldsymbol{x}_n) \geq 1 - \xi_n$ 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

$$\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_n \xi_n - \sum_n a_n\big(t_n(\boldsymbol{w}^T\phi(\boldsymbol{x}_n) + b) - 1 + \xi_n\big) - \sum_n \mu_n \xi_n$$

where $a_n \geq 0$ and $\mu_n \geq 0$ are Lagrange multipliers.

Support Vector Machines for Classification

- Setting its derivatives with respect to $\boldsymbol{w}$, b and ξ_n to zero gives

$$\boldsymbol{w} = \sum_n a_n t_n \phi(\boldsymbol{x}_n)$$

$$0 = \sum_n a_n t_n$$

$$a_n = C - \mu_n$$

- Replacing these in the Lagrangian function gives the dual representation of the problem, in which we maximize

$$\sum_n a_n - \frac{1}{2}\sum_n\sum_m a_n a_m t_n t_m k(\boldsymbol{x}_n, \boldsymbol{x}_m)$$

subject to $a_n \geq 0$ and $a_n \leq C$ for all n, because $\mu_n \geq 0$.

- When the Lagrangian function is maximized, the Karush-Kuhn-Tucker conditions hold for all n:

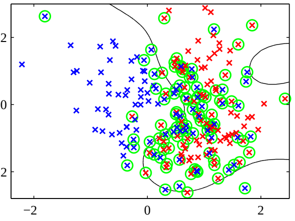
$$a_n(t_n y(\boldsymbol{x}_n) - 1 + \xi_n) = 0$$

$$\mu_n \xi_n = 0$$

- Then, $a_n > 0$ if and only if $t_n y(\boldsymbol{x}_n) = 1 - \xi_n$ 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 ξ_n is unconstrained.

Support Vector Machines for Classification

- Since the optimal $\boldsymbol{w}$ 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 $\boldsymbol{x}_n$ 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 $\{(\boldsymbol{x}_n, t_n)\}$. Consider using the linear model

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \phi(\boldsymbol{x}) + b$$

- To get a sparse solution, instead of minimizing the classical regularized error function

$$\frac{1}{2}\sum_n (y(\boldsymbol{x}_n) - t_n)^2 + \frac{\lambda}{2}||\boldsymbol{w}||^2$$

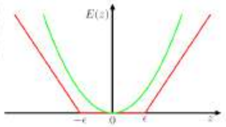
consider minimizing the ϵ-insensitive regularized error function

$$C\sum_n E_\epsilon(y(\boldsymbol{x}_n) - t_n) + \frac{1}{2}||\boldsymbol{w}||^2$$

where $C > 0$ controls regularization and

$$E_\epsilon(y(\boldsymbol{x}) - t) = \begin{cases} 0 & \text{if } |y(\boldsymbol{x}) - t| < \epsilon \\ |y(\boldsymbol{x}) - t| - \epsilon & \text{otherwise} \end{cases}$$

Figure 7.6 Plot of an ϵ-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).



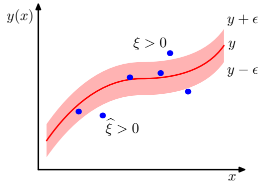
Support Vector Machines for Regression

- The values of C and ϵ can be decided by cross-validation.
- Consider the slack variables $\xi_n \geq 0$ and $\widehat{\xi}_n \geq 0$ such that

$$\xi_n = \begin{cases} t_n - y(\boldsymbol{x}_n) - \epsilon & \text{if } t_n > y(\boldsymbol{x}_n) + \epsilon \\ 0 & \text{otherwise} \end{cases}$$

and

$$\widehat{\xi}_n = \begin{cases} y(\boldsymbol{x}_n) - \epsilon - t_n & \text{if } t_n < y(\boldsymbol{x}_n) - \epsilon \\ 0 & \text{otherwise} \end{cases}$$



Support Vector Machines for Regression

- The optimal regression curve is given by

$$\underset{\boldsymbol{w}, b, \{\xi_n\}, \{\widehat{\xi}_n\}}{\arg\min} \; C\sum_n (\xi_n + \widehat{\xi}_n) + \frac{1}{2}||\boldsymbol{w}||^2$$

subject to $\xi \geq 0$, $\widehat{\xi}_n \geq 0$, $t_n \leq y(\boldsymbol{x}_n) + \epsilon + \xi_n$ and $t_n \geq y(\boldsymbol{x}_n) - \epsilon - \widehat{\xi}_n$.

- To minimize the previous expression, we minimize

$$C\sum_n (\xi_n + \widehat{\xi}_n) + \frac{1}{2}||\boldsymbol{w}||^2 - \sum_n (\mu_n \xi_n + \widehat{\mu}_n \widehat{\xi}_n)$$
$$- \sum_n a_n (y(\boldsymbol{x}_n) + \epsilon + \xi_n - t_n) - \sum_n \widehat{a}_n (t_n - y(\boldsymbol{x}_n) + \epsilon + \widehat{\xi}_n)$$

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 $\boldsymbol{w}$, b, ξ_n and $\widehat{\xi}_n$ to zero gives

$$\boldsymbol{w} = \sum_n (a_n - \widehat{a}_n)\phi(\boldsymbol{x}_n)$$
$$0 = \sum_n (a_n - \widehat{a}_n)$$
$$C = a_n + \mu_n$$
$$C = \widehat{a}_n + \widehat{\mu}_n$$

Support Vector Machines for Regression

- Replacing these in the Lagrangian function gives the dual representation of the problem, in which we maximize

$$\frac{1}{2}\sum_n \sum_m (a_n - \widehat{a}_n)(a_m - \widehat{a}_m) k(\boldsymbol{x}_n, \boldsymbol{x}_m) - \epsilon \sum_n (a_n + \widehat{a}_n) + \sum_n (a_n - \widehat{a}_n) t_n$$

subject to $a_n \geq 0$ and $a_n \leq C$ for all n, because $\mu_n \geq 0$. Similarly for $\widehat{a}_n$.

- When the Lagrangian function is maximized, the Karush-Kuhn-Tucker conditions hold for all n:

$$a_n(y(\boldsymbol{x}_n) + \epsilon + \xi_n - t_n) = 0$$
$$\widehat{a}_n(t_n - y(\boldsymbol{x}_n) + \epsilon + \widehat{\xi}_n) = 0$$
$$\mu_n \xi_n = 0$$
$$\widehat{\mu}_n \widehat{\xi}_n = 0$$

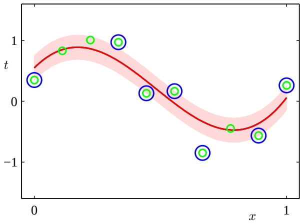
- Then, $a_n > 0$ if and only if $y(\boldsymbol{x}_n) + \epsilon + \xi_n - t_n = 0$, which implies that $\boldsymbol{x}_n$ lies on or above the upper margin of the ϵ-tube. Similarly for $\widehat{a}_n > 0$.

Support Vector Machines for Regression

- The prediction for a new point $\boldsymbol{x}$ is made according to

$$y(\boldsymbol{x}) = \sum_{m \in \mathcal{S}} (a_m - \widehat{a}_m) k(\boldsymbol{x}, \boldsymbol{x}_m) + b$$

where $\mathcal{S}$ are the indexes of the support vectors. Sparse solution!



- To find b, consider any support vector $\boldsymbol{x}_n$ with $0 < a_n < C$. Then, $\mu_n > 0$ and thus $\xi_n = 0$ and thus $0 = t_n - \epsilon - y(\boldsymbol{x}_n)$. Then,

$$b = t_n - \epsilon - \sum_{m \in \mathcal{S}} (a_m - \widehat{a}_m) k(\boldsymbol{x}_n, \boldsymbol{x}_m)$$

Neural Networks

- Consider binary classification with input space $\mathbb{R}^D$. Consider a training set $\{(\boldsymbol{x}_n, t_n)\}$ where $t_n \in \{-1, +1\}$.
- SVMs classify a new point $\boldsymbol{x}$ according to

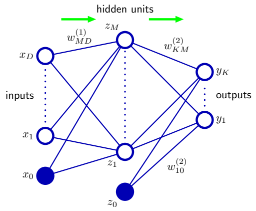
$$y(\boldsymbol{x}) = \text{sgn}\Big(\sum_{m \in \mathcal{S}} a_m t_m k(\boldsymbol{x}, \boldsymbol{x}_m) + b\Big)$$

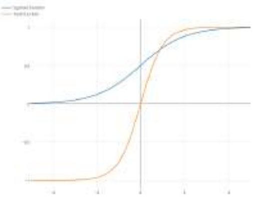
- Consider regressing an unidimensional continuous random variable on a D-dimensional continuous random variable. Consider a training set $\{(\boldsymbol{x}_n, t_n)\}$
- For a new point $\boldsymbol{x}$, SVMs predict

$$y(\boldsymbol{x}) = \sum_{m \in \mathcal{S}} (a_n - \widehat{a}_n) k(\boldsymbol{x}, \boldsymbol{x}_m) + b$$

- SVMs imply data-selected user-defined basis functions.
- NNs imply a user-defined number of data-selected basis functions.

Neural Networks





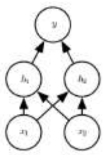
- Activations: $a_j = \sum_i w_{ji}^{(1)} x_i + w_{j0}^{(1)}$
- Hidden units and activation function: $z_j = h(a_j)$
- Output activations: $a_k = \sum_j w_{kj}^{(2)} z_j + w_{k0}^{(2)}$
- Output activation function for regression: $y_k(\boldsymbol{x}) = a_k$
- Output activation function for classification: $y_k(\boldsymbol{x}) = \sigma(a_k)$
- Sigmoid function: $\sigma(a) = \frac{1}{1+\exp(-a)}$
- Two-layer NN:

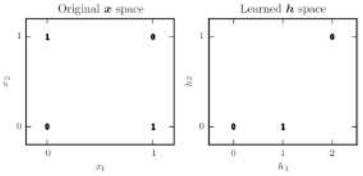
$$y_k(\boldsymbol{x}) = \sigma\Big(\sum_j w_{kj}^{(2)} h\Big(\sum_i w_{ji}^{(1)} x_i + w_{j0}^{(1)}\Big) + w_{k0}^{(2)}\Big)$$

- 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

- 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 shatter the points. Note that the points (0,1) and (1,0) are mapped both to the point (1,0).
- **It resembles SVMs.**





$w_{11}^{(1)} = w_{12}^{(1)} = w_{21}^{(1)} = w_{22}^{(1)} = 1$
$w_{10}^{(1)} = 0,\ w_{20}^{(1)} = -1$
$h_j = z_j = h(a_j) = \max\{0, a_j\}$
$w_{11}^{(2)} = 1,\ w_{12}^{(2)} = -2$
$w_{10}^{(2)} = 0$
$y = y_k = a_k$

Backpropagation Algorithm

- Consider regressing an K-dimensional continuous random variable on a D-dimensional continuous random variable.
- Consider a training set $\{(\boldsymbol{x}_n, \boldsymbol{t}_n)\}$. Consider minimizing the sum-of-squares error function

$$E(\boldsymbol{w}) = \sum_n E_n(\boldsymbol{w}) = \sum_n \frac{1}{2} \|\boldsymbol{y}(\boldsymbol{x}_n) - \boldsymbol{t}_n\|^2 = \sum_n \sum_k \frac{1}{2} (y_k(\boldsymbol{x}_n) - t_{nk})^2$$

- This error function can be justified from a maximum likelihood approach to learning $\boldsymbol{w}$. To see it, **assume** that

$$p(t_k | \boldsymbol{x}, \boldsymbol{w}, \sigma) = \mathcal{N}(t_k | y_k(\boldsymbol{x}), \sigma)$$

- Then, the likelihood function is

$$p(\{\boldsymbol{t}_n\} | \{\boldsymbol{x}_n\}, \boldsymbol{w}, \sigma) = \prod_n \prod_k \mathcal{N}(t_{nk} | y_k(\boldsymbol{x}_n), \sigma) = \prod_n \prod_k \frac{1}{(2\pi\sigma^2)^{1/2}} e^{-\frac{1}{2\sigma^2}(t_{nk} - y_k(\boldsymbol{x}_n))^2}$$

and thus

$$-\ln p(\{\boldsymbol{t}_n\} | \{\boldsymbol{x}_n\}, \boldsymbol{w}, \sigma) = \sum_n \sum_k \frac{1}{2\sigma^2} (t_{nk} - y_k(\boldsymbol{x}_n))^2 + \frac{N}{2} \ln \sigma^2 + \frac{N}{2} \ln 2\pi$$

which is equivalent to the sum-of-squares error function for a given σ.

- If σ is not given, then we can find the ML estimates of $\boldsymbol{w}$, plug them into the log likelihood function, and maximize it with respect to σ.

Backpropagation Algorithm

- The weight space is highly multimodal and, thus, we have to resort to approximate iterative methods to minimize the previous expression.
- Batch gradient descent

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta_t \nabla E(\boldsymbol{w}^t)$$

where $\eta_t > 0$ is the learning rate ($\sum_t \eta_t = \infty$ and $\sum_t \eta_t^2 < \infty$ to ensure convergence, e.g. $\eta_t = 1/t$).

- Sequential, stochastic or online gradient descent

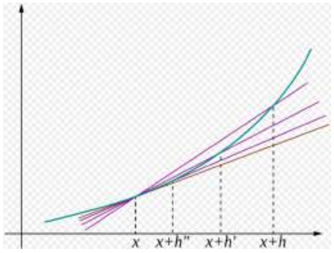
$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta_t \nabla E_n(\boldsymbol{w}^t)$$

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 $f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$



- Recall that $\nabla E_n(\boldsymbol{w}^t)$ is a vector whose components are the partial derivatives of $E_n(\boldsymbol{w}^t)$.

Backpropagation Algorithm

- Backpropagation algorithm:
  1. Forward propagate to compute activations, and hidden and output units.
  2. Compute δ_k for the output units.
  3. Backpropagate the δ's, i.e. evaluate δ_j for the hidden units recursively.
  4. Compute the required derivatives.

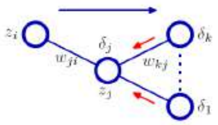


Figure 5.7 Illustration of the calculation of δ_j for hidden unit j by backpropagation of the δ's from those units k to which unit j sends connections. The blue arrow denotes the direction of information flow during forward propagation, and the red arrows indicate the backward propagation of error information.

- For classification, we minimize the negative log likelihood function, a.k.a. cross-entropy error function:

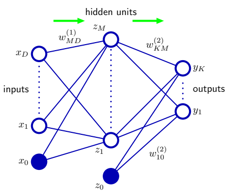
$$E_n(\boldsymbol{w}) = -\sum_k [t_{nk} \ln y_k(\boldsymbol{x}_n) + (1 - t_{nk}) \ln(1 - y_k(\boldsymbol{x}_n))]$$

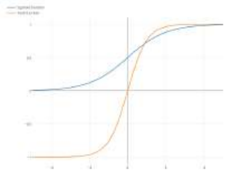
  with $t_{nk} \in \{0, 1\}$ and $y_k(\boldsymbol{x}_n) = \sigma(a_k)$. Then, again

$$\frac{\partial E_n}{\partial w_{kj}} = \delta_k z_j \text{ and } \delta_k = \frac{\partial E_n}{\partial a_k} = y_k - t_k$$

- 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 lose 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 take a value around half its saturation level. That is why the hyperbolic tangent function is sometimes preferred in practice.

Regularization

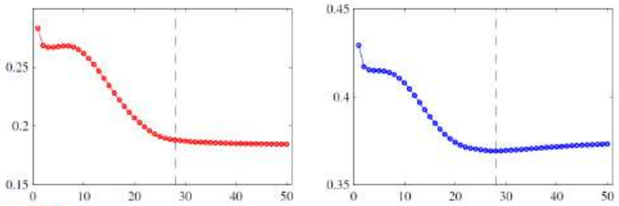


Figure 5.12 An illustration of the behaviour of training set error (left) and validation set error (right) during a typical training session, as a function of the iteration step, for the sinusoidal data set. The goal of achieving the best generalization performance suggests that training should be stopped 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
$$E(\boldsymbol{w}) + \frac{\lambda}{2} ||\boldsymbol{w}||^2 \text{ or } E(\boldsymbol{w}) + \frac{\lambda_1}{2} ||\boldsymbol{w}^{(1)}||^2 + \frac{\lambda_2}{2} ||\boldsymbol{w}^{(2)}||^2$$
    and choose λ, or λ_1 and λ_2 by cross-validation. Note that the effect of the penalty is simply to add λw_{ji} and λw_{kj}, or $\lambda_1 w_{ji}$ and $\lambda_2 w_{kj}$ to the appropriate derivatives.

Limitations of Neural Networks

Theorem (Universal approximation theorem)

For every continuous function $f : [a,b]^D \to \mathbb{R}$ and for every $\epsilon > 0$, there exists 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 the sigmoid activation function. Then, for any distribution $p(\mathbf{x}, t)$,

$$\lim_{k\to\infty} \inf_{y\in\mathcal{C}^{(k)}} L(y(\mathbf{x})) - L(p(t|\mathbf{x})) = 0$$

where $L()$ is the 0/1 loss function.

- 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 does the backpropagation algorithm converge 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.

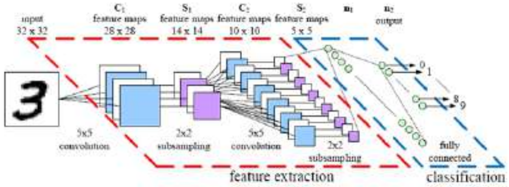
Theorem (No free lunch theorem)

For any algorithm, good performance on some problems comes at the expense of bad performance on some others.

Deep Neural Networks

- 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 σ and tanh is in $[0,1)$. So, they may only suffer the gradient vanishing problem. Other activations 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 5x5 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 compute from a 2x2 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

- DNNs allow increased depth because
  - they are sparse, which allows the gradient to propagate further, and
  - they have relatively few weights to fit 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

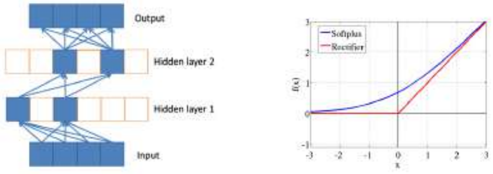


Figure 2: *Left:* **Sparse propagation of activations and gradients in a network of rectifier units.** The input selects a subset of active neurons and computation is linear in this subset. *Right:* **Rectifier and softplus activation functions.** The second one is a smooth version of the first.

- *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

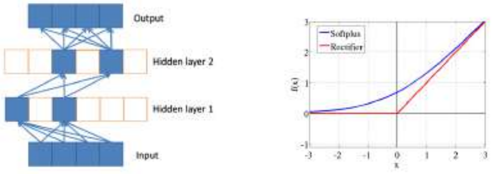


Figure 2: *Left:* **Sparse propagation of activations and gradients in a network of rectifier units.** The input selects a subset of active neurons and computation is linear in this subset. *Right:* **Rectifier and softplus activation functions.** The second one is a smooth version of the first.

- 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. Compare with the sigmoid or hyperbolic tangent, for which
  - the gradient is smaller than one, or
  - even zero due to saturation.
- 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.