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 $\hat{w}$ 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

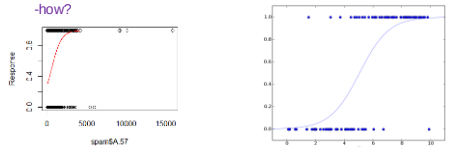
- **Test data** (test set T): used for predictions
  - Supervised learning: estimate $p(Y)$ or $\hat{Y}$ for new $\boldsymbol{x}$

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 : $p(spam) = p(Y = spam|x)$

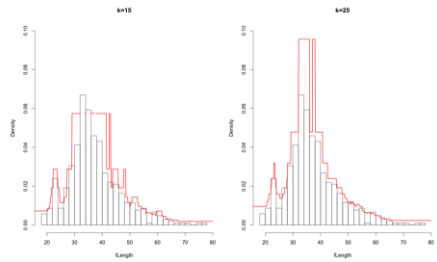
We can also make point predictions -how?



K-nearest neighbor density estimation

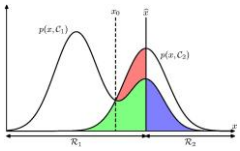
- Data: Fish length $X_1, \dots X_N$
- Model $p(x|K) = \frac{K}{N \cdot \Delta}$
  - K: #neighbors in training data
  - Δ: length of the interval containing K neighbors
- Learning: Fix some K or find an appropriate K
- Prediction: predict $p(x|K)$

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



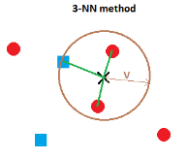
25

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 $Y = C_i$



26

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

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

27

K-nearest neighbor classification

Algorithm

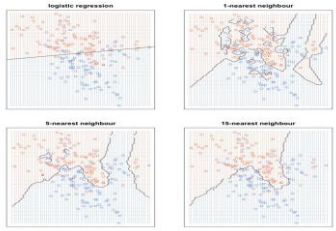
1. Given training set D,number K, and test set T
2. For each $x \in T$
   1. For each $i = 1, \dots M$
      1. $p'(C_i|x) = \frac{K_i}{K}$
   2. Compute $l = \arg \max_{i \in \{1,\dots,m\}} p'(C_i|\boldsymbol{x})$
   3. Predict $\hat{Y}(x) = C_l$

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

28

K-nearest neigbor example

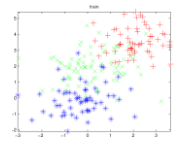
Why classification results are so different for K-NN?

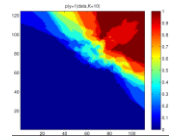


29

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

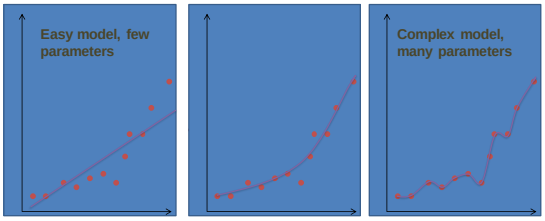




30

Overfitting

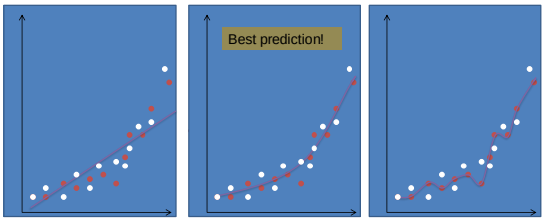
- Which model feels appropriate?



31

Overfitting

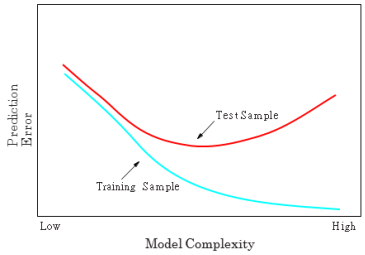
Now new data from the same process



32

Overfitting

- Observed:



33

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

34

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**

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

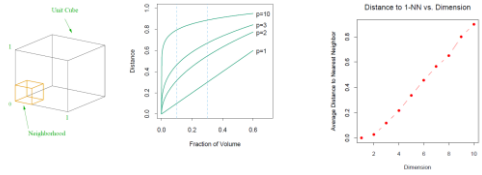
35

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

36

Curse of dimensionality



37

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

38

Basics of Statistics

Lecture 1b

39

Probability

How likely it is that some event will happen?

Idea:

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

40

Probability

Example: Tossing a coin two times

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

41

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

42

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

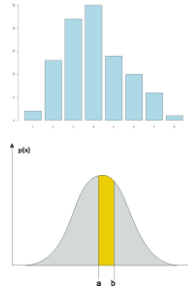
- Instead of having events, we can have a variable X:
  - Events→$\mathbb{R}$ Continuous random variables
  - Events→$\mathbb{N}$ Discrete random variables

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

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

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

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

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

49

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,..

50

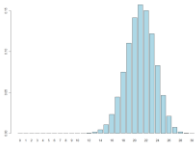
Some conventional distributions

Binomial distribution

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

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

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



51

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}

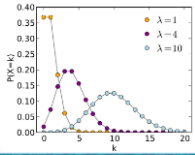
52

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

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

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



53

Poisson distribution

- Further properties:
  - Poisson distribution is a good approximation of the binomial distribution if n >20 and $p < 0.05$
  - Excellent approximation if $n \geq 100$ and $np \leq 10$

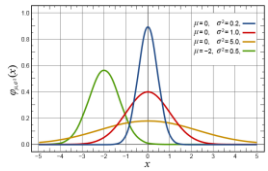
54

Normal distribution

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

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

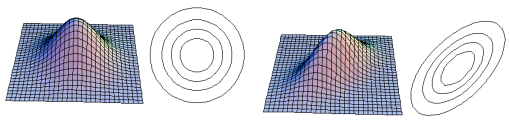
- $E(X) = \mu$
- $Var(X) = \sigma^2$



55

Multivariate distributions

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



56

Basic ML ingridients

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

Case	X_1	X_2	Y
1			
2			
…			

- Model $P(x|\, w_1, \ldots w_k)$ or $P(y|x, w_1, \ldots w_k)$
  - 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

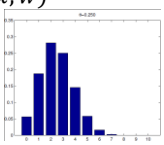
57

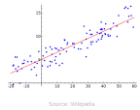
Probabilistic models

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

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

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





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

58

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

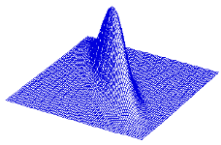
59

Fitting a model

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

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

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



  - Maximize the likelihood and find the optimal w^*

60

Fitting a model

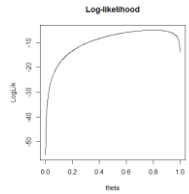
Remarks:

- 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.

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



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

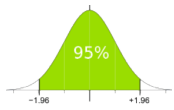
    http://en.wikipedia.org/wiki/Conjugate_prior
  - Conjugacy
    - Distribution of the posterior is the same type as the distribution of the likelihood or prior
- Prior is the most controversial about Bayesian methods, but
  - When $N \to \infty$, data overwhelms the prior

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

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 $x = 5$

Fitting a model

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

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

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

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

Vectors

- Create a vector
  x<-c(1,3)
- See the result
  x
  print(x)

```
> x<-c(1,3)
> x
[1] 1 3
> print(x)
[1] 1 3
>
```

- Create an empty vector
  Y=numeric(10)
  Y

```
> Y=numeric(10)
> Y
 [1] 0 0 0 0 0 0 0 0 0 0
```


73

Sequence

- Either': ' or seq()

R Console

```
> f<-3:5
> f
[1] 3 4 5
> g<-seq(from=3, to=7, by=0.5)
> g
[1] 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0
>
```


74

Operation with vectors

- indexing
- Element-wise: +-*/^
- log exp sin cos
- length –number of elements
- sum - sum of all elements
- max min sort order
- which.min which.max

Logicals:
TRUE or FALSE:
A=TRUE;

== > >= < <= != & (and) | (or)

```
> a=1:5
> b=c(1,4,-1,3,0)
> a+b
[1] 2 6 2 7 5
> a*b
[1]  1  8 -3 12  0
> b+4
[1] 5 8 3 7 4
> length(a)
[1] 5
> sum(a^2)
[1] 55
> max(b)
[1] 4
> which.max(b)
[1] 2
> order(b)
[1] 3 5 1 4 2
> sort(b)
[1] -1  0  1  3  4
> b[1]
[1] 1
> b[2:4]
[1]  4 -1  3
> b[-2]
[1]  1 -1  3  0
```


75

Matrices

Use `matrix()`

`a<-matrix(values,nrow=m,ncol=n)`

Values should be listed columnvise

`nrow=` and `ncol=` can be skipped

R Console

```
> a<-matrix(c(2,1,1,-1),nrow=2,ncol=2)
> a
     [,1] [,2]
[1,]    2    1
[2,]    1   -1
>
```

- Create empty matrix

```
> m=matrix(0,nrow=2,ncol=3)
> m
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
>
```


76

Matrix operations

```
> x<-c(1,2)
> a<-matrix(c(2,1,1,-1),2,2)
> b<-matrix(c(1,0,1,1),2,2)
> y=a%*%x
> y
     [,1]
[1,]    4
[2,]   -1
> c=a%*%b
> c
     [,1] [,2]
[1,]    2    3
[2,]    1    0
>
```

Usual vector operations can also be applied:

```
> m4=matrix(c(1,2,0,1), nrow=2)
> m5=matrix(c(2,2,5,1), nrow=2)
> m4
     [,1] [,2]
[1,]    1    0
[2,]    2    1
> m5
     [,1] [,2]
[1,]    2    5
[2,]    2    1
> m4*m5
     [,1] [,2]
[1,]    2    0
[2,]    4    1
```


77

Matrix operations

- Matrix operators/functions:
- transpose `b=t(a)`

  $b = a^T$
- Inverse $b = a^{-1}$

  `b=solve(a)`
- Solve $d=a^{-1}b$

  `d=solve(a,b)`

```
> a
     [,1] [,2]
[1,]    2    1
[2,]    1   -1
> t(a)
     [,1] [,2]
[1,]    2    1
[2,]    1   -1
> solve(a)
           [,1]       [,2]
[1,] 0.3333333  0.3333333
[2,] 0.3333333 -0.6666667
>
```


78

Indexing for matrices

- Positive index

`x[1,6]` `x[2:10,]`

- Negative index

`x[2,-(1:5)]` row 2 and all columns except 1:5

- Entire column or row

`y=x[2,]` entire row 2

- Extraction

`x[x>5]`

```
> b
[1]  1  4 -1  3  0
> d=b[b>0]
> d
[1] 1 4 3
```


79

Replication

- Replication for vectors
  - rep(what, times)
- Replication for matrices
  - matrix()

```
> v1=rep(3,5)
> v1
[1] 3 3 3 3 3
> v2=rep(c(3,4),2)
> v2
[1] 3 4 3 4
> m1=matrix(1,nrow=2,ncol=2)
> m1
     [,1] [,2]
[1,]    1    1
[2,]    1    1
> m2=matrix(v2,nrow=4,ncol=2)
> m2
     [,1] [,2]
[1,]    3    3
[2,]    4    4
[3,]    3    3
[4,]    4    4
> m3=matrix(v2,nrow=2,ncol=4, byrow=T)
> m3
     [,1] [,2] [,3] [,4]
[1,]    3    4    3    4
[2,]    3    4    3    4
```


80

Matrix operations

- Dimension
  - dim(mat)
- Row/column stististics
  - colMeans, rowMeans, colSums,rowSums

```
> m4=matrix(c(1,3,5,2,4,6),nrow=3)
> m4
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
> colMeans(m4)
[1] 3 4
> rowSums(m4)
[1]  3  7 11
```

- Apply a function over vector/matrix
  - Sapply()
  - Normally used when function works only element-wise

```
> m2
     [,1] [,2]
[1,]    3    3
[2,]    4    4
[3,]    3    3
[4,]    4    4
> ns=dim(m2)
> ns[2]
[1] 2
```

```
> sapply(v2,log)
[1] 1.098612 1.386294 1.098612 1.386294
> log(v2)
[1] 1.098612 1.386294 1.098612 1.386294
```


81

Vector/matrix operations

- Create confusion matrix (classification)
  - table(X,Y)
- Extract diagonal
  - Diag(X)

```
> X=c(1,3,1,1,2,3,1,2,2,2,1,1,3)
> Xfit=c(2,3,2,1,2,3,1,2,2,2,1,1,1)
> t1=table(Xfit,X)
> t1
    X
Xfit 1 2 3
   1 4 0 1
   2 2 4 0
   3 0 0 2
> t1[1,1]
[1] 4
> diag(t1)
1 2 3
4 4 2
>
```


82

Factors

- Text values

```
> f1=c("Man", "Woman")
> f1
[1] "Man"   "Woman"
> f2=c("Man", "Woman", "Man")
> table(f2)
f2
  Man Woman
    2     1
> f3=factor(c(1,0,1,1,0), levels=c(0,1), labels=c("Man", "Woman"))
> f3
[1] Woman Man   Woman Woman Man
Levels: Man Woman
>
```


83

Lists

- List is a collection of objects

```
> d<-15;
> a<-matrix(c(1,2,3,4),2,2);
> a
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> b<-list(first=d, second=a, x="mary")
> b
$first
[1] 15

$second
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$x
[1] "mary"
```

```
> b$x
[1] "mary"
> b[[3]]
[1] "mary"
```


84

Data frame

Vectors and matrices of the row length can be collected into a data frame

- Used to store the data of different types into a single table

Use `data.frame` (*object 1*, *object 2*, … , *object k*)

```
> x<-c(1,3)
> y<-c("M", "F")
> z<-data.frame(x,y)
> z
  x y
1 1 M
2 3 F
```


Data frame

- Any column in the data frame can be retrieved by

  dataframe$object

```
> z$x
[1] 1 3
> z[[1]]
[1] 1 3
> z$y
[1] M F
Levels: F M
```

- Any row in the data frame can be extracted by using matrix notation, for ex: **z[1,]**

Read data from Excel file

1. Save as "comma-separated file"(csv)
2. Change current directory, Session→ Set Working Directory or setwd()
3. Use

```
Dataframe=read.csv2(file_name)
```

```
Dataframe=read.csv(file_name)
```


Conversion between types

- Data frame to matrix
- Matrix to data frame
- Numeric to factor
- Factor to numeric
- List to vector
- Vector to list

```
> df1=data.frame(X=c(1,2,3), Y=c(1,0,1))
> df1
  X Y
1 1 1
2 2 0
3 3 1
> m5=as.matrix(df1)
> m5
     X Y
[1,] 1 1
[2,] 2 0
[3,] 3 1
> df2=as.data.frame(m5)
> df2$X
[1] 1 2 3
```

```
> v6=c(1,4,2,2,1)
> f4=as.factor(v6)
> f4
[1] 1 4 2 2 1
Levels: 1 2 4
> f5=c("1", "0", "1", "1", "1")
> f5
[1] "1" "0" "1" "1" "1"
> as.list(f5)
[[1]]
[1] "1"

[[2]]
[1] "0"

[[3]]
[1] "1"

[[4]]
[1] "1"

[[5]]
[1] "1"

> l1=list(a=1, b=3)
> l1
$a
[1] 1

$b
[1] 3

> as.numeric(l1)
[1] 1 3
```


Loops

```
for (name  in expr1 )
{
…
}

while (condition)
{
…
}
```

```
> for (i in 1:5) {
+ y=seq(i,8)
+ print(y) }
[1] 1 2 3 4 5 6 7 8
[1] 2 3 4 5 6 7 8
[1] 3 4 5 6 7 8
[1] 4 5 6 7 8
[1] 5 6 7 8
>
```


Conditioning and loops

```
If (x==3) {
…
…
} else {
…
}

for (i in 2:99) {
…
}

while(x!=29) {
…
}
```

```
> m4=matrix(c(1,2,0,1), nrow=2)
> m4
     [,1] [,2]
[1,]    1    0
[2,]    2    1
> n=dim(m4)[1]
> I=numeric(n)
> for (i in 1:n){
+   if(max(m4[i,]>1)) I[i]=1
+ }
> I
[1] 0 1
```

Random number generation

- Random are not random
  - Use set.seed(12345) to get identical results
- A plenty of random number generators
  - Rnorm
  - Runif
  - …
- Use d for density p for CDF q for quantiles and r for simulation:

(ex: rnorm pnorm dnorm qnorm)

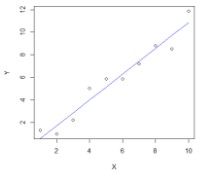
Distribution	R name	additional arguments
beta	beta	shape1, shape2, ncp
binomial	binom	size, prob
Cauchy	cauchy	location, scale
chi-squared	chisq	df, ncp
exponential	exp	rate
F	f	df1, df2, ncp
gamma	gamma	shape, scale
geometric	geom	prob
hypergeometric	hyper	m, n, k
log-normal	lnorm	meanlog, sdlog
logistic	logis	location, scale
negative binomial	nbinom	size, prob
normal	norm	mean, sd
Poisson	pois	lambda
Student's t	t	df, ncp
uniform	unif	min, max
Weibull	weibull	shape, scale
Wilcoxon	wilcox	m, n

91

Using a function

- Use *?name_of_function* to see function parameters
  - For ex. *?lm*
- There are some obligatory parameters and optional parameters
- The optional parameters can be specified in different order

```
X=1:10
Y=1:10+rnorm(10)
W=c(rep(1,5), rep(2,5))
mydata=data.frame(X,Y)

result=lm(Y~X,  weights=W,data=mydata)
?predict.lm
Fit=predict(result)

plot(X,Y)
points(X,Fit, type="l", col="blue")
```



92

Writing your own functions

- Function writing must always end with writing the value which should be returned!
- You may also use "**return(value)**" to show what value the function should return

```
> myfun <-function(x=20, y, z)
+ {
+ if(z)
+ result=x+y
+ else
+ {
+ result=log(x)*y
+ }
+ result
+ }
> r<-myfun(1,2, TRUE)
> r
[1] 3
> r<-myfun(z=FALSE, y=0)
> r
[1] 0
>
```

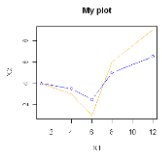

93

Graphical procedures

Some common procedures:

- plot(x,..) plots time series
- plot(x,y) scatter plot
- plot(x,y) followed by points(x,y) plots several scatterplots in one coordinate system
- hist(x,..) plots a hitogram
- persp(x,y,z,…) creates surface plots
- cloud(formula,data..) creates 3D scatter plot

```
x<-c(1,4,7 8, 12);
y<-c(4,3,1,6,9);

plot(x,y, type="l", col="orange",
     main="My plot", xlab="X1", ylab="X2");
points(x, y/2+2, type="b", col="blue");
```



94

Graphical parameters

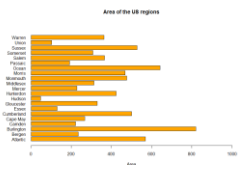
- Adjust color of a graphical object by specifying
  - col=
- Other typical parameters for graphical functions
  - main="text" Title"text"
  - sub="text" Footnote "text"
  - xlab="text" X-axis label
  - ylab="text" Y-axis label

```
mydata<-read.csv2("Counties.csv");
barplot(mydata$Area, names.arg=mydata$County, main="Area of the US regions",
xlab="County", ylab="Area");
```


95

Graphical parametes

- Some parameters need to be specified either in the plotting function or inside par(…)
  - Pch=number – symbol that is plotted
  - Lty=number – linetype
  - Las=0 1 eller 2 Direction of axis values
  - mai=c(bottom, left, top, right) – margins (inch)
  - adj=between 0 and 1, horizontal justification



```
barplot(mydata$Area,
names.arg=mydata$County, horiz=TRUE, las=1,
xlim=c(0,1000), col="orange", main="Area of
the US regions", xlab="Area");
```


96

Some more examples

- Dividing training/test

```
data=data.frame(X=c(1,1,2,2,3), Y=c("M","F","M","M","F"))
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]
```

```
> train
  X Y
4 2 M
5 3 F
> test
  X Y
1 1 M
2 1 F
3 2 M
```

- Computing misclassification rate

```
missclass=function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}
```

```
> X=c(1,3,1,1,2,3,1,2,2,2,1,1,3)
> Xfit=c(2,3,2,1,2,3,1,2,2,2,1,1,1)
> missclass(X,Xfit)
[1] 0.2307692
```


97

Regression and regularization

Lecture 1d

98

Overview

- Linear regression
- Ridge Regression
- Lasso
- Variable selection

99

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

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

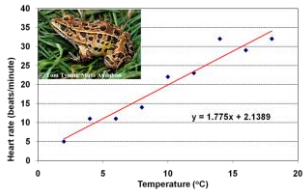
Terminology:

w_0: **intercept (or bias)**

w_1: **regression coefficient**

Response

The target responds directly and linearly to changes in the feature



100

Ordinary least squares regression (OLS)

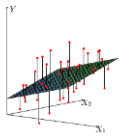
Model:

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

where

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

Why is "1" here?



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

101

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

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

102

Matrix formulation of OLS regression

Optimality condition:

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

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

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

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

Hat matrix

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

Why is it called so?

Degrees of freedom

Definition:

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

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

$$df=trace(S)$$

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

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:

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

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:  0.6851,    Adjusted R-squared:  0.5952
F-statistic: 86.26 on 1 and 57 DF,  p-value: 5.248e-13
```


109

An example of ordinary least squares regression

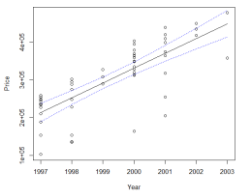
```
> summary(fit2)

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
```


110

An example of ordinary least squares regression

- Prediction

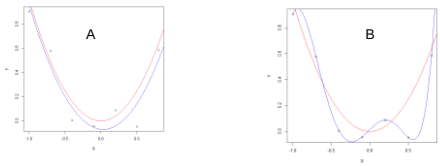
```
fitted <- predict(fit1, interval =
"confidence")

# 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)
```



111

Ridge regression

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



112

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:**

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

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

113

Ridge regression

Equivalent form

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

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

Solution

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

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

Hat matrix

How do we compute degrees of freedom here?

114

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), $X^T X = I \rightarrow$

  $\hat{\mathbf{w}}^{\text{ridge}} = \frac{1}{1+\lambda}\mathbf{w}^{\mathbf{linreg}}$ → 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:

$$\boldsymbol{y} \sim N(\boldsymbol{y}|\boldsymbol{w}_o + \boldsymbol{X}\boldsymbol{w}, \boldsymbol{\sigma^2 I})$$

$$\boldsymbol{w} \sim \boldsymbol{N}\left(\mathbf{0}, \frac{\sigma^2}{\lambda}\boldsymbol{I}\right)$$

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

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

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

Ridge regression

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

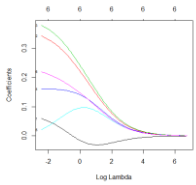
- Cycle time
- Memory
- Channels
- …

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)
```

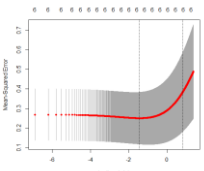


Ridge regression

- 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 "dgC
                        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
>
```


121

LASSO

- **Idea**: Similar idea to Ridge
- Minimize minus loglikelihood plus **linear** penalty factor→ $\mathbf{l_1}$ **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**

122

LASSO

- Equivalently

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

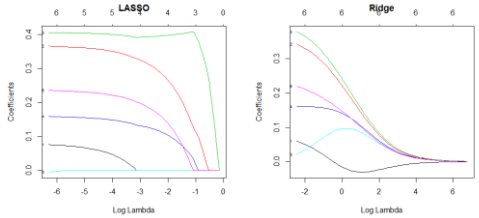
$$\textbf{subject to}\ \sum_{j=1}^{p}|w_j| \le s$$

123

LASSO vs Ridge

- **LASSO yields sparse solutions!**

Example Computer hardware data



124

LASSO vs Ridge

- Only 5 variables selected by LASSO

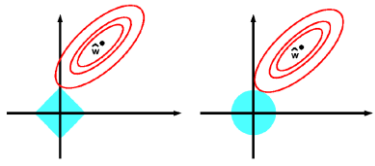
```
> coef(model, s="lambda.min")
7 x 1 sparse Matrix of class "dgCMatrix"
                      1
(Intercept) -5.091825e-17
V3           6.350488e-02
V4           3.578607e-01
V5           4.033670e-01
V6           1.541329e-01
V7           .
V8           2.287134e-01
>
```

```
> sum((ynew-mean(y))^2)/sum((y-mean(y))^2)
[1] 0.5826904
> sum((ynew-y)^2)
[1] 16.63756
```


125

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)



126

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,

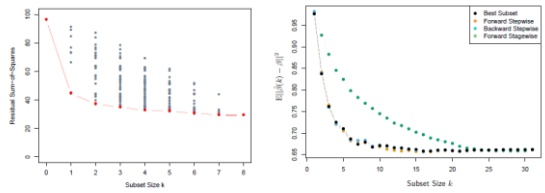
$$\hat{w}_i^{\text{lasso}} = sign\left(w_i^{linreg}\right)\left(\left|w_i^{linreg}\right| - \frac{\lambda}{2}\right)_+$$

- No explicit formula for $\hat{\boldsymbol{w}}^{\boldsymbol{lasso}}$
  - Optimization algorithms used

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

127

RSS and MSE depend on k



130

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

128

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  4.666e-02   8.693 1.18e-15 ***
V6           1.591e-01  3.394e-02   4.687 5.07e-06 ***
V8           2.360e-01  3.350e-02   7.043 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
```


131

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")

129

Model selection

Lecture 1e

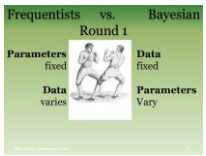
132

Overview

- Model fitting
- Model selection

Frequentist vs Bayesian

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



Example: Linear regression, what are parameters here?

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

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

An estimator

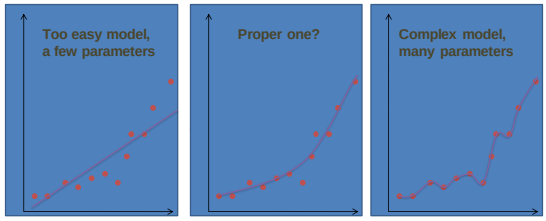
- $\hat{\boldsymbol{w}} = \delta(D)$ (some function of your data) – an **estimator**
- Optimal parameter values?→there can be many ways to compute them (MLE, shrinkage…)
  - Compare Bayesian: given estimators $\boldsymbol{w^1}$ and $\boldsymbol{w^2}$, we **can** compare them! $p(\boldsymbol{w^1}|\boldsymbol{D}) > \boldsymbol{p}(\boldsymbol{w^2}|\boldsymbol{D})$
  - 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

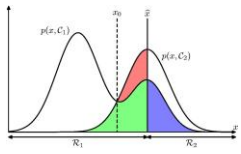
- Given a model, choose the optimal parameter values
  - Decision theory
- Define loss $L(Y, \hat{y})$
  - How much we loose in guessing true Y incorrectly
- If we know the true distribution $p(y, x|w)$ then we choose $\hat{y}$

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

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:

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

Example: Compute loss function related to

- Normal distribution

Guess why such loss function was chosen

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

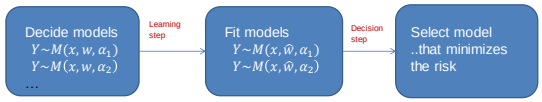
- **Problem**: true model and true $\boldsymbol{w}$ are unknown→can not compute expected loss!
- How to find an optimal model?
  - Consider what expected loss (**risk**) depends on
    $$R(Y,\hat{y}) = E[L(Y,\hat{y}(X,D))]$$
- Random factors:
  - D – **training set**
  - Y, X – data to be predicted **(validation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

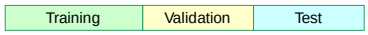
- Given data $D = \{\boldsymbol{X}_i, Y_i, i = 1 \dots n\}$



- 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

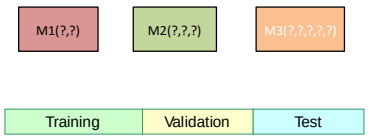


- Choose proportions in some way

145

Holdout method

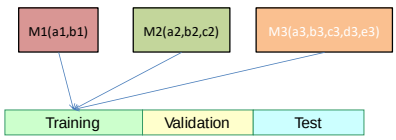
- Given: training, validation, test sets and models to select between



146

Holdout method

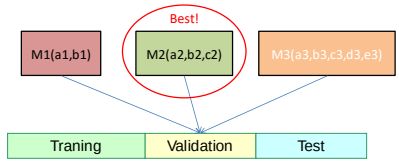
- Training set is to used for fitting models to the dataset by using maximum likelihood



147

Holdout method

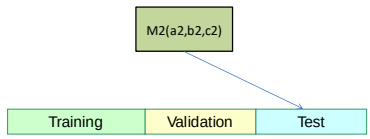
- Validation set is used to choose the best model (lowest risk)



148

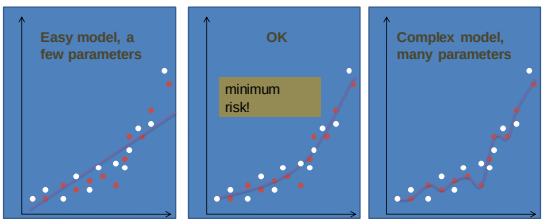
Holdout method

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



149

Holdout method



150

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,]
```


151

Bias-variance tradeoff

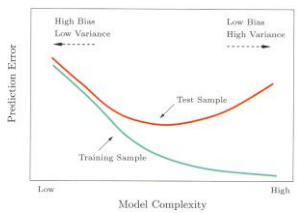
- Bias of an estimator $Bias(\hat{y}(x_0)) = E[\hat{y}(x_0) - f(x_0)]$, $f(x_0)$ is expected response
  - If $Bias(\hat{y}(x_0)) = 0$, the estimator is **unbiased**
  - ML estimators are asymptotically unbiased if the model is enough complex
  - However, unbiasedness does not mean a good choice!

152

Bias-variance tradeoff

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

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



When loss is not quadratic, no such nice formula exist

153

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)

154

Cross-validation

Cross-validation

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

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

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?

155

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

156

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):

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

157

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,

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

158

Model selection

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

- Cycle time
- Memory
- Channels
- …

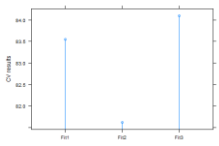
Build model predicting performance

159

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)
```



160

Linear classification methods

Lecture 2a

161

Overview

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

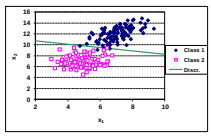
162

Classification

- Given data $D = \{(X_i, Y_i), i = 1 \dots N\}$
  - $Y_i = Y(X_i) = C_j \in \boldsymbol{C}$
  - 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

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

- Example 1: 0/1-loss

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

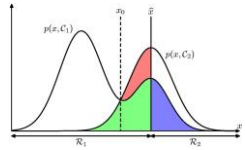
- Example 2: Spam

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

Loss and decision

- Expected loss minimization
  - R_j : classify to C_j

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



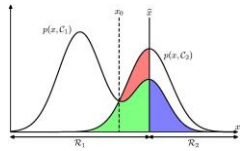
- **Choose such R_j that EL is minimized**
- Two classes

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

Loss and decision

- Loss minimization

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



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!

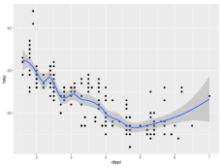
$$\frac{p(C_1|x)}{p(C_2|x)} > \frac{L_{21}}{L_{12}} \rightarrow predict\ y\ as\ C_1$$

169

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

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

170

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**

$$Specificity = 1 - FPR$$

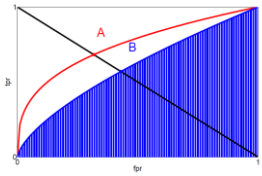
- **Precision**

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

172

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**)



173

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		
		1	0	Total
TRUE	1	TP	FN	N_+
	0	FP	TN	N_-

171

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
  - $p(Y = 1|\boldsymbol{w}, \boldsymbol{x}) = \frac{1}{1+e^{-w^T x}}$

174

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

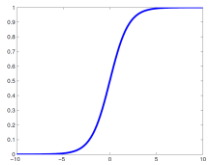
175

Logistic regression

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

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

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



- Alternatively

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

176

Logistic regression

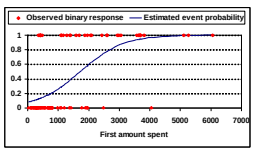
- Logistic model- yet another form

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

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 $\boldsymbol{w^T x}$ via logit link

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



177

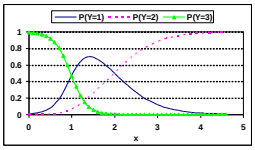
Logistic regression

- When Y is categorical,

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

- Alternatively

$$Y \sim Multinoulli\left(softmax(\boldsymbol{w_1^T x}), \ldots softmax(\boldsymbol{w_K^T x})\right)$$



178

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

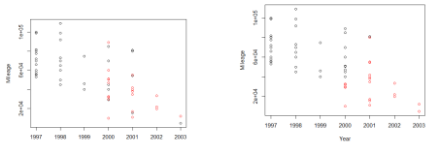
179

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**

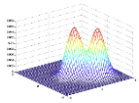


180

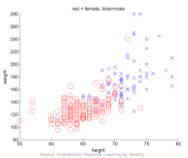
Quadratic discriminant analysis

- 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 $\boldsymbol{\theta} = \{\boldsymbol{\mu_i}, \boldsymbol{\Sigma_i}\}$

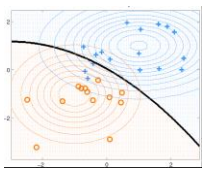


181

Quadratic discriminant analysis

- If parameters are estimated, classify:

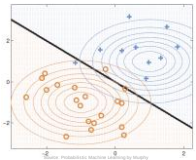
$$\hat{y}(\boldsymbol{x}) = \arg\max_c p(y = c|\boldsymbol{x}, \theta)$$



182

Linear discriminant analysis (LDA)

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



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

183

Linear discriminant analysis (LDA)

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

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

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

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

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

184

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(..)

185

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
```

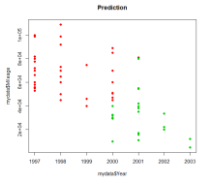

186

LDA: output

- Misclassified items

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

    0  1
0 31  6
1  7 15
```

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



187

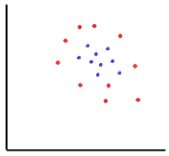
Naïve Bayes classifiers
Decision trees

Lecture 2b

190

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



188

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",...
- Want to classify a new document

191

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.

189

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

192

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!

193

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) ?

194

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: $\hat{Y} = \arg\max_y p(Y = y|X)$

195

Naive Bayes classifiers - discrete inputs

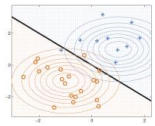
- **Example** Loan decision
  - Classify a person: Home Owner=No, Single=Yes

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

196

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$(\sigma_{i1}^2, \dots, \sigma_{ip}^2)$



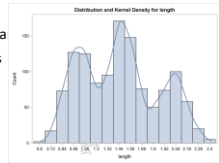
- Naive bayes is a special case of LDA (given A)
  - →MLE are means and variances (per class)

197

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 nonpa
2. Estimate $p(Y = y_k)$ as class proportions
3. Use Bayes rule and 0-1 loss to classify



198

Naive Bayes in R

- naiveBayes in package **e1071**

Example: Satisfaction of householders with their present housing circumstances

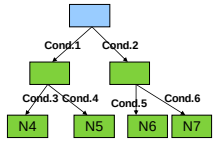
```
library(MASS)
library(e1071)
n=dim(housing)[1]
ind=rep(1:n, housing[,5])
housing1=housing[ind,-5]

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
```


199

Definitions

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



202

Decision trees

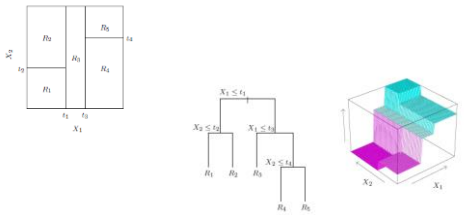
Idea

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

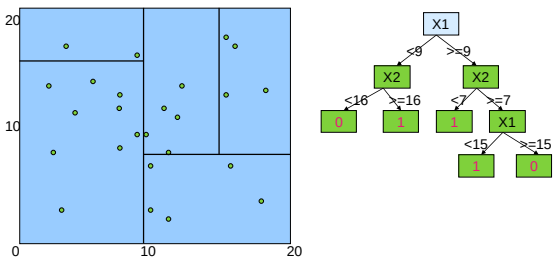
200

Regression tree toy example



203

Classification tree toy example



201

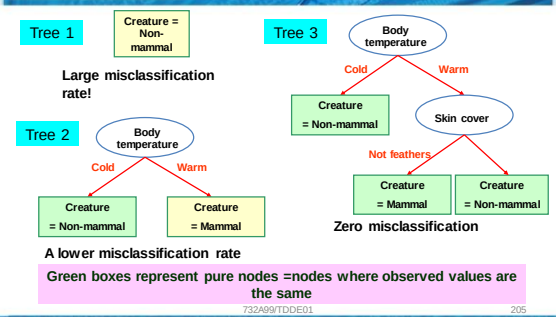
A classification problem

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

204

Several solutions



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

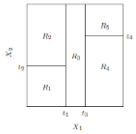
205

Decision trees

- A tree $T = < r_i, s_{r_i}, R_j, i = 1 \dots S, j = 1 \dots L >$
  - $x_{r_i} \leq s_{r_i}$ 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



206

Decision trees

Example:

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

207

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

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

208

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 $Y_j|T \sim Multinomial(p_{j1}, \dots p_{jc})$

209

Fitting regression trees: CART

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

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

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

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)

210

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?

211

Optimal trees

- **Postpruning**

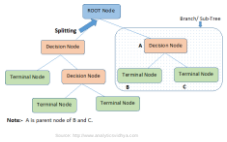
Weakest link pruning:

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

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

$|T|$ =#leaves

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



How to find the optimal α? Cross validation!

212

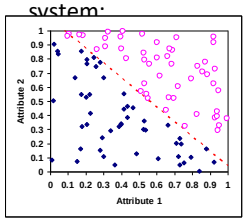
Decision trees: comments

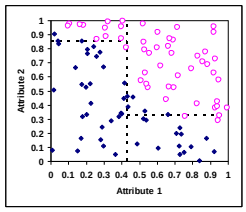
- Similar algorithms work for regression trees – replace $N \cdot Q(R)$ by $SSE(R)$
- Easy to interpret
- Easy to handle all types of features in one model
- **Automatic variable selection**
- Relatively robust to outliers
- Handle large datasets
- Trees have high variance: a small change in response→ totally different tree
- Greedy algorithms → fit may be not so good
- Lack of smoothness

213

Decision trees: issues

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





214

Decision trees in R

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

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

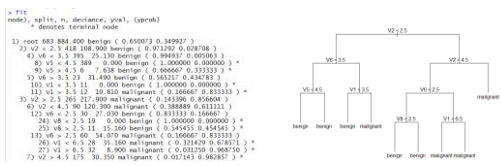
Example: breast cancer as a function av biological measurements

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


215

Decision trees in R

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




```
> 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
```


216

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
```


217

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)
```

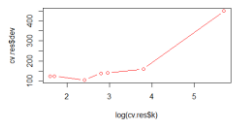
```
> table(valid$class,Yfit)
           Yfit
            benign malignant
  benign       222         8
  malignant      6       114
```

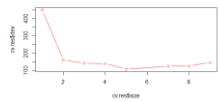

220

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?

218

Generalized Linear Models. Uncertainty estimation

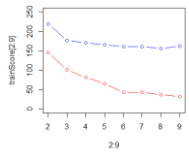
Lecture 2c

221

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?

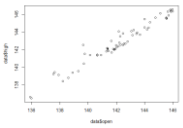
219

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)

222

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 $Y \sim EF(\mu, \ldots),\ f(\mu) = \boldsymbol{w}^T\boldsymbol{x}$
  - 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

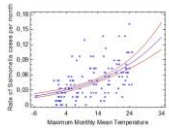
$$p(y|w,s) = h(y,s)g(\boldsymbol{w},\boldsymbol{x})e^{\frac{(\boldsymbol{w}^T\boldsymbol{x})y}{s}}$$

Generalized linear models

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

Example: Poisson regression

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

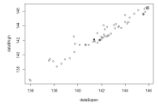


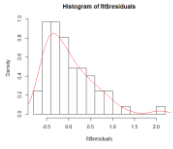
Generalized linear model: software

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

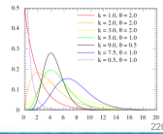
Example: Daily Stock prices NASDAQ
- Open
- High (within day)

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



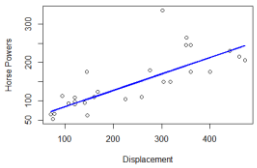


Gamma distribution: Wikipedia



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:

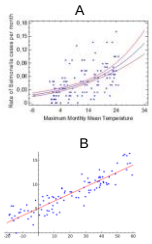
$$y = f(x, w)$$

- 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 …" 😮



229

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

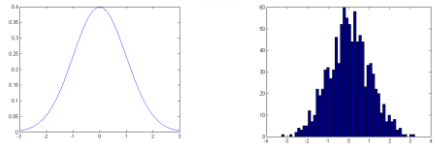
230

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**

231

The bootstrap: general principle



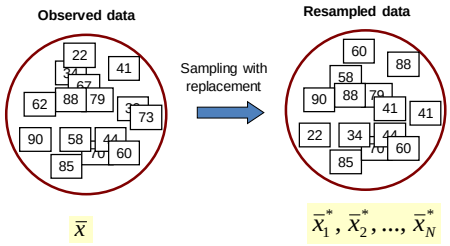
We want to determine uncertainty of $\hat{f}(D,X)$

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)

232

Nonparametric bootstrap



233

Nonparametric bootstrap

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

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

1. Estimate $\hat{w}$ from data **D**=(X_1,…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

234

Parametric bootstrap

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

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

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

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

Uncertainty estimation

1. Get $D_1 \ldots D_B$ by bootstrap
2. Use $\hat{f}(D_1) \ldots \hat{f}(D_B)$ 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

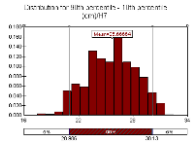
- To estimate 100(1-α) confidence interval for w

Bootstrap percentile method

1. Using bootstrap, compute $\hat{f}(D_1) \ldots \hat{f}(D_B)$, 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

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

Look at the plot…



Bootstrap: regression context

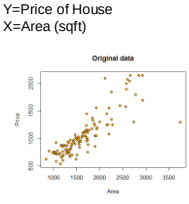
- Model $Y \sim F(X, w)$
- Data $D = \{(Y_i, X_i), i = 1, \ldots, n\}$
- Idea: produce several bootstrap sets that are similar to D

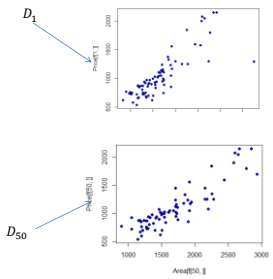
Nonparametric bootstrap:

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

Uncertainty estimation

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





We sample data index, from {1…N}

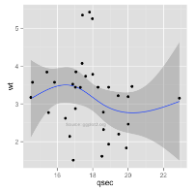
Bootstrap: regression context

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. $D_i = \{(X_i^*, Y_i^*), i = 1, \ldots, n\}$
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?
  - $\mu|x = f(x,w)$



241

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

242

Bootstrap: R

- Package **boot**
  - **Functions:**
    - boot()
    - boot.ci() – 1 parameter
    - envelope() – many parameters
- Random random generation for parametic bootstrap:
  - Rnorm()
  - Runif()
  - …

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

243

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
```


244

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

245

Bootstrap

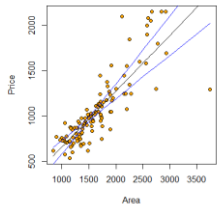
```
mle=lm(Price~Area, data=data2)

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")
```


246

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")
```



247

Prediction bands

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

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

Prediction band for parametric bootstrap

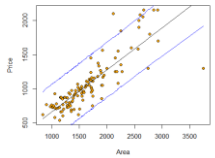
1. Run parametric bootstrap and get $D_1, \dots D_B$
2. Fit the model to the data and get $\hat{w}(D_1), \dots \hat{w}(D_B)$
3. For each X, generate from $Distribution(X, \hat{w}(D_1)), \dots Distribution(X, \hat{w}(D_B))$ and apply percentile method
4. Connect the intervals→get the band

248

Estimation of the model quality

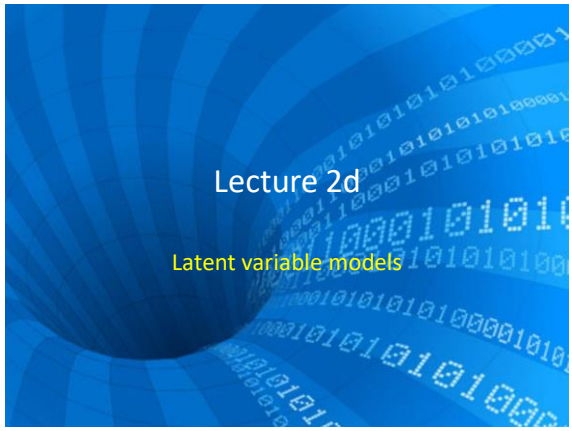
Example: parametric bootstrap

```
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?

249



250

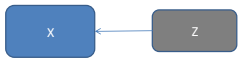
Overview

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

251

Latent variables

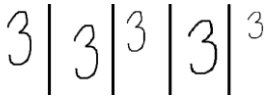
- Sometimes data depends on the variables we can not measure (hard to measure)
  - Answers on the test depend on Intelligence
  - Brain activity in the brain is measured by sensors
  - Stock prices depend on market confidence



252

Latent variables

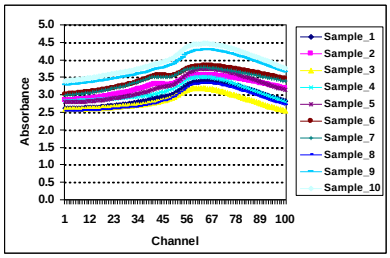
- Latent factor discovered→ data storage may decrease a lot
- Latent factors
  - Center
  - Scaling
- Original vs compressed
  - 100x100x5=50000
  - 100x100+2*5+2*5=10020



253

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

256

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
- …

254

Principal components analysis

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

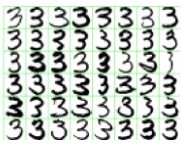
- The first principal component (PC1) is the direction that maximizes the variance of the projected data
- The second principal component (PC2) is the direction that maximizes the variance of the projected data after the variation along PC1 has been removed
- The third principal component (PC3) is the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

257

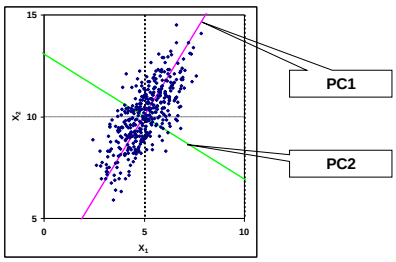
Principal Component Analysis (PCA)

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



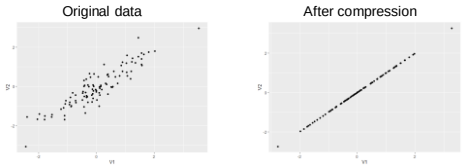
255

Principal Component Analysis - two inputs



258

PCA- after reducing dimensionality



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

259

PCA: computations

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

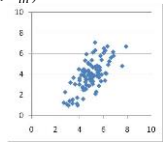
1. Centred data

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

2. Covariance matrix

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

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

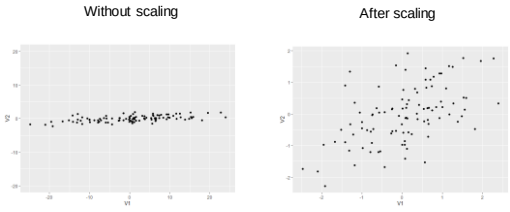


	Column 1	Column 2
Column 1	0.951	0.905
Column 2	0.905	1.883

262

PCA and scaling

- Do we need to scale features?



260

PCA: computations

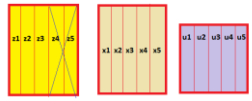
4. Coordinates of any data point x=(x_1…x_p) in the new coordinate system:
$z = (z_1, \dots z_n), \; z_i = x^T u_i$

Matrix form: $Z = X \, U$

5. Discard principle components after some *M:*
$Z = X \, U_M$

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

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



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

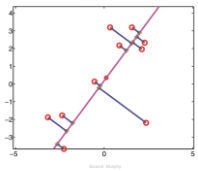
100*50 vs
100*4+50*4

263

PCA: another view

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

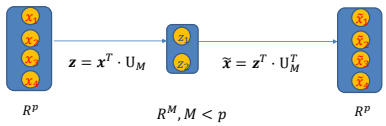
$$\min_{U_M} \sum_{i=1}^{N} \|x_n - \tilde{x}_n\|^2$$



261

PCA: computations

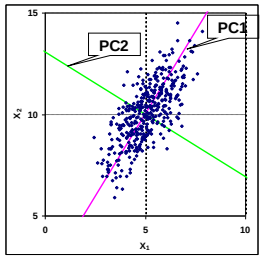
- PCA makes a **linear** compression of features



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

264

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)

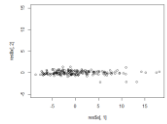
265

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
```



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

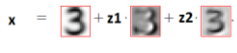
Data in (PC1, PC2) – scores (Z)

Do we need second dimension?

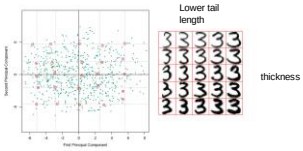
268

Principal Component Analysis

- Digits: two eigenvectors extracted



- Interptretation of eigenvectiors

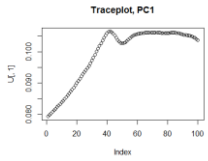


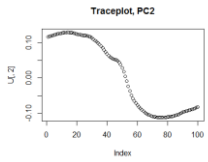
266

PCA in R

- Trace plots

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





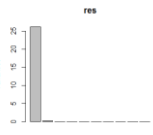
Which components contribute to PC1-2?

269

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!

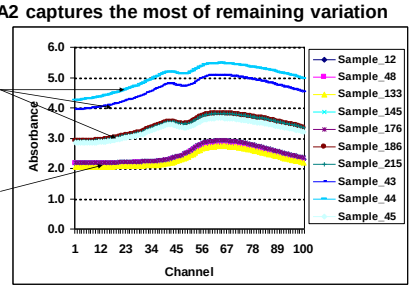
267

Absorbance records for ten samples of chopped meat

PCA2 captures the most of remaining variation

High fat samples

Low fat samples



270

Probabilistic PCA

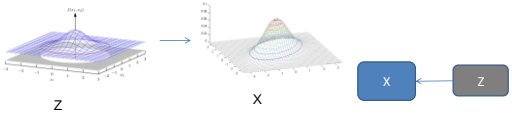
- z_i-latent variables, x_i- observed variables

$$\boldsymbol{z} \sim \boldsymbol{N}(0, \boldsymbol{I})$$
$$\boldsymbol{x}|\boldsymbol{z} \sim \boldsymbol{N}(\boldsymbol{x}|\boldsymbol{Wz} + \boldsymbol{\mu}, \sigma^2\boldsymbol{I})$$

- Alternatively

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

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



271

Probabilistic PCA

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

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

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

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

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

272

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

273

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

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

274

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

275

Probabilistic PCA in R

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

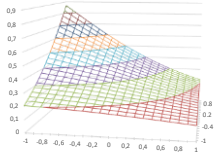
Ppca(data, nPcs,…)

Results: scores, loadings…

276

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**



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

277

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 $||v_i|| = 1$

278

ICA: estimation algorithm

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

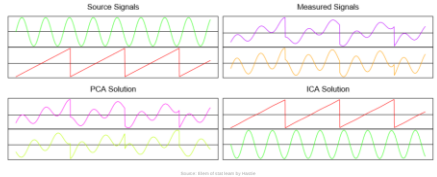
- **With prewhitening**
  1. Convert X into PCA coordinate system (do not remove dimensions): $X' = XU$
  2. Estimate V by maximum likelihood in ICA
  3. Estimate final scores $Z = X'V$

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

279

ICA

- Example

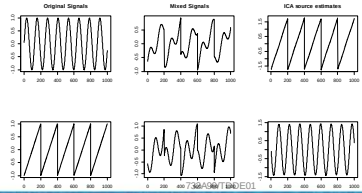


280

Independent component analysis: R

R package: fastICA

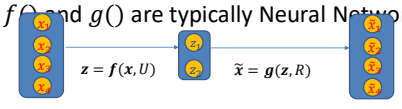
```
S <- cbind(sin((1:1000)/20), rep((((1:200)-100)/100), 5))
A <- matrix(c(0.291, 0.6557, -0.5439, 0.5572), 2, 2)
X <- S %*% A #mixing signals
a <- fastICA(X, 2) #now separate them
```



281

Autoencoders (nonlinear PCA)

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



R^p $\qquad R^M, M < p \qquad R^p$

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

…or some other loss function

282

732A99/TDDE01 Machine Learning

Lecture 3a Block 1: Kernel Methods

Jose M. Peña
IDA, Linköping University, Sweden

283

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 $\boldsymbol{x}$.

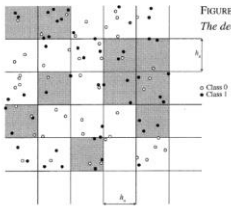


FIGURE 6.1. *A cubic histogram rule: The decision is 1 in the shaded area.*

- In other words,

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

284

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.

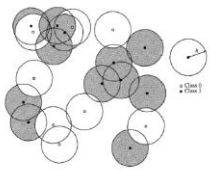


FIGURE 10.1. *The moving window rule in $\mathcal{R}^2$. The decision is 1 in the shaded area.*

- 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 $\boldsymbol{x}$.

285

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.

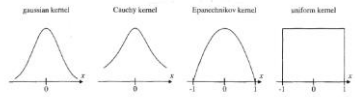


FIGURE 10.3. *Various kernels on $\mathcal{R}$.*

- 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: $k(u) = \mathbf{1}_{\{u \in S(0,1)\}}$

286

Kernel Classification

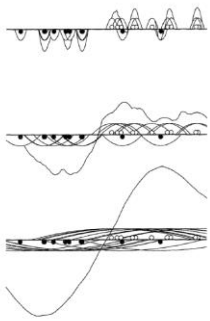


FIGURE 10.2. *Kernel rule on the real line. The figure shows $\sum_{i=1}^n (2Y_i - 1)K((x - X_i)/h)$ for $n = 20$, $K(u) = (1 - u^2)I_{\{|u| \leq 1\}}$ (the Epanechnikov kernel), and three smoothing factors h. One definitely undersmooths and one oversmooths. We took $p = 1/2$, and the class-conditional densities are $f_0(x) = 2(1 - x)$ and $f_1(x) = 2x$ on $[0, 1]$.*

287

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

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

288

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 $\int k(u) du = 1$.

289

Histogram, Moving Window, and Kernel Density Estimation

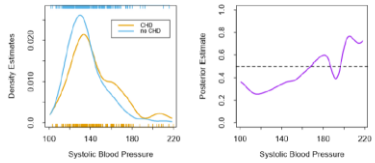


FIGURE 6.14. *The left panel shows the two separate density estimates for systolic blood pressure in the CHD versus no-CHD groups, using a Gaussian kernel density estimate in each. The right panel shows the estimated posterior probabilities for CHD, using (6.25).*

292

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.

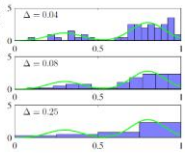
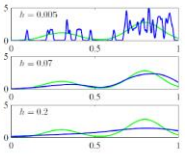


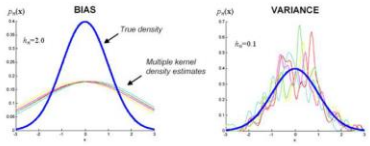
Figure 2.25 Illustration of the kernel density model (2.250) applied to the same data set used to demonstrate the histogram approach in Figure 2.24. We see that h acts as a smoothing parameter and that if it is set too small (top panel), the result is a very noisy density model, whereas if it is set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



290

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.

293

Histogram, Moving Window, and Kernel Density Estimation

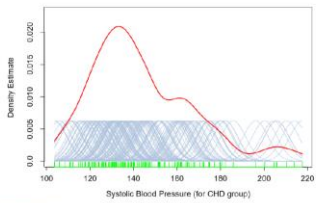


FIGURE 6.13. *A kernel density estimate for systolic blood pressure (for the CHD group). The density estimate at each point is the average contribution from each of the kernels at that point. We have scaled the kernels down by a factor of 10 to make the graph readable.*

- 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.

291

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.

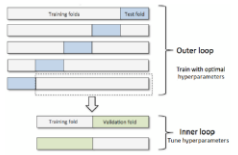
294

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 ?

295

Kernel Trick

- Two alternatives for building $k(\boldsymbol{x}, \boldsymbol{x}')$:
  - Choose a convenient $\phi(\boldsymbol{x})$ and let $k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}')$.
  - Build it from existing kernel functions as follows.

Techniques for Constructing New Kernels.

Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the following new kernels will also be valid:

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= c k_1(\mathbf{x}, \mathbf{x}') && (6.13)\\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') && (6.14)\\
k(\mathbf{x}, \mathbf{x}') &= q(k_1(\mathbf{x}, \mathbf{x}')) && (6.15)\\
k(\mathbf{x}, \mathbf{x}') &= \exp(k_1(\mathbf{x}, \mathbf{x}')) && (6.16)\\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') && (6.17)\\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') && (6.18)\\
k(\mathbf{x}, \mathbf{x}') &= k_3(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')) && (6.19)\\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}' && (6.20)\\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) && (6.21)\\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) k_b(\mathbf{x}_b, \mathbf{x}'_b) && (6.22)
\end{aligned}
$$

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\boldsymbol{\phi}(\mathbf{x})$ is a function from $\mathbf{x}$ to $\mathbb{R}^M$, $k_3(\cdot, \cdot)$ is a valid kernel in $\mathbb{R}^M$, $\mathbf{A}$ is a symmetric positive semidefinite matrix, $\mathbf{x}_a$ and $\mathbf{x}_b$ are variables (not necessarily disjoint) with $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, and k_a and k_b are valid kernel functions over their respective spaces.

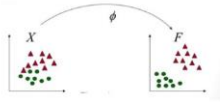
298

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 $M = D = 2$.

296

732A99/TDDE01 Machine Learning

Lecture 3b Block 1: Support Vector Machines

Jose M. Peña
IDA, Linköping University, Sweden

299

Kernel Trick

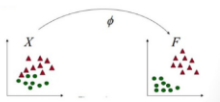
- Consider again moving window classification, regression, and density estimation.
- Note that $\boldsymbol{x}_n \in S(\boldsymbol{x}, h)$ if and only if $\|\boldsymbol{x} - \boldsymbol{x}_n\| \leq h$.
- Note that

$$\|\boldsymbol{x} - \boldsymbol{x}_n\| = \sqrt{(\boldsymbol{x} - \boldsymbol{x}_n)^T (\boldsymbol{x} - \boldsymbol{x}_n)} = \sqrt{\boldsymbol{x}^T \boldsymbol{x} + \boldsymbol{x}_n^T \boldsymbol{x}_n - 2\boldsymbol{x}^T \boldsymbol{x}_n}$$

- 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 $\phi(\boldsymbol{x}_n)$.

297

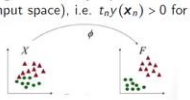
Support Vector Machines for Classification

- 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

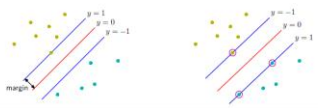
$$y(\boldsymbol{x}) = \boldsymbol{w}^T \phi(\boldsymbol{x}) + b$$

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.

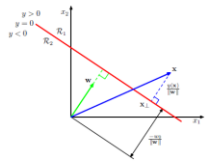


- Aim for the separating hyperplane that maximizes the margin (i.e. the smallest perpendicular distance from any point to the hyperplane) so as to minimize the generalization error.



300

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

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

- 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.

5/18

301

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

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

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

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

$$0=\sum_n a_n t_n$$

6/18

302

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.

7/18

303

Support Vector Machines for Classification

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

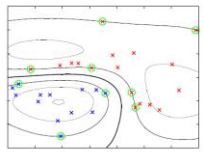
$$a_n(t_n y(\boldsymbol{x}_n)-1)=0$$

- 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!



8/18

304

Support Vector Machines for Classification

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

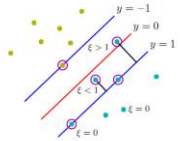
$$1=t_n y(\boldsymbol{x}_n)=t_n\Big(\sum_{m\in\mathcal{S}} a_m t_m k(\boldsymbol{x}_n,\boldsymbol{x}_m)+b\Big)$$

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

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



9/18

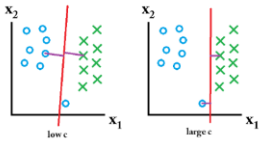
305

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

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

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

10/18

306

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:

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

$$\mu_n \xi_n = 0$$

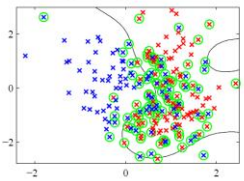
- Then, $a_n > 0$ if and only if $t_n y(\boldsymbol{x}_n) = 1 - \xi_n$ for all n. The points with $a_n > 0$ are called support vectors and they lie
  - on the margin if $a_n < C$, because then $\mu_n > 0$ and thus $\xi_n = 0$, or
  - inside the margin (even on the wrong side of the decision boundary) if $a_n = C$, because then $\mu_n = 0$ and thus ξ_n is unconstrained.

11/18

307

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.

12/18

308

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

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

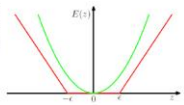
consider minimizing the ϵ-insensitive regularized error function

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

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).



13/18

309

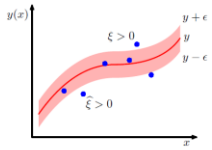
Support Vector Machines for Regression

- 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

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



14/18

310

Support Vector Machines for Regression

- The optimal regression curve is given by

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

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

$$\boldsymbol{w} = \sum_n (a_n - \widehat{a}_n)\phi(\boldsymbol{x}_n)$$

$$0 = \sum_n (a_n - \widehat{a}_n)$$

$$C = a_n + \mu_n$$

$$C = \widehat{a}_n + \widehat{\mu}_n$$

15/18

311

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 $\widehat{a}_n > 0$.

16/18

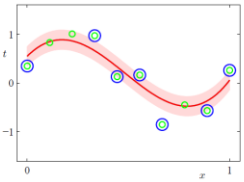
312

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!

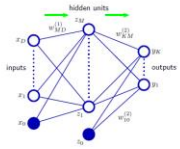


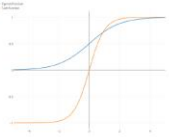
- To find b, consider any support vector $\boldsymbol{x}_n$ with $0 < a_n < C$. Then, $\mu_n > 0$ and thus $\xi_n = 0$ and thus $0 = t_n - \epsilon - y(\boldsymbol{x}_n)$. Then,

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

313

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.

316

732A99/TDDE01 Machine Learning

Lecture 3c Block 1: Neural Networks

Jose M. Peña
IDA, Linköping University, Sweden

314

Neural Networks

- For a large variety of activation functions, the two-layer NN can uniformly approximate any continuous function to arbitrary accuracy provided enough hidden units. Easy to fit the parameters ? Overfitting ?!

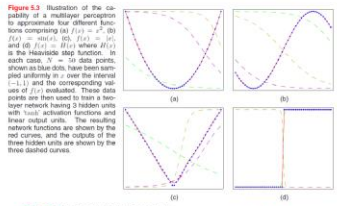


Figure 5.3 Illustration of the capability of a multilayer perceptron to approximate four different functions comprising (a) $f(x) = x^2$, (b) $f(x) = \sin(x)$, (c) $f(x) = |x|$, and (d) $f(x) = H(x)$ where $H(x)$ is the Heaviside step function. In each case, $N = 50$ data points, shown as blue dots, have been sampled uniformly in x over the interval $(-1, 1)$ and the corresponding values of $f(x)$ evaluated. These data points are then used to train a two-layer network having 3 hidden units with 'tanh' activation functions and linear output units. The resulting network functions are shown by the red curves, and the outputs of the three hidden units are shown by the three dashed curves.

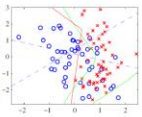


Figure 5.4 Example of the solution of a simple two-class classification problem involving synthetic data using a neural network having two inputs, two hidden units with 'tanh' activation functions, and a single output having a logistic sigmoid activation function. The dashed blue lines show the $z = 0.5$ contours for each of the hidden units, and the red line shows the $y = 0.5$ decision surface for the network. For comparison, the green line denotes the optimal decision boundary computed from the distributions used to generate the data.

317

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

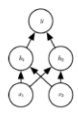
$$y(\boldsymbol{x}) = \sum_{m \in \mathcal{S}} (a_n - \widehat{a}_n) k(\boldsymbol{x}, \boldsymbol{x}_m) + b$$

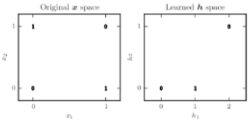
- SVMs imply data-selected user-defined basis functions.
- NNs imply a user-defined number of data-selected basis functions.

315

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**.





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

318

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.

8/16

319

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

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

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

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

where n is chosen randomly or sequentially.

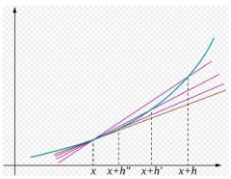
- Sequential gradient descent is less affected by the multimodality problem, as a local minimum of the whole data will not be generally a local minimum of each individual point.

9/16

320

Backpropagation Algorithm

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



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

10/16

321

Backpropagation Algorithm

- Since E_n depends on w_{ji} only via a_j, and $a_j = \sum_i w_{ji}x_i$, then

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}x_i = \delta_j x_i$$

- Since E_n depends on a_j only via a_k, then

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial a_k}{\partial a_j}$$

- Since $a_k = \sum_j w_{kj}z_j$ and $z_j = h(a_j)$, then

$$\frac{\partial a_k}{\partial a_j} = h'(a_j)w_{kj}$$

- Putting all together, we have that

$$\delta_j = h'(a_j)\sum_k \delta_k w_{kj}$$

- Since $y_k = a_k$ for regression and $a_k = \sum_j w_{kj}z_j$, then

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

- Backpropagation algorithm:
  1. Forward propagate to compute activations, and hidden and output units.
  2. Compute 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.

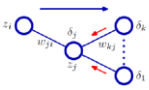
11/16

322

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

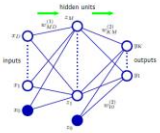
$$\frac{\partial E_n}{\partial w_{kj}} = \delta_k z_j \text{ and } \delta_k = \frac{\partial E_n}{\partial a_k} = y_k - t_k$$

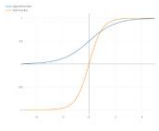
- This is an example of embarrassingly parallel algorithm.

12/16

323

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
  $$\frac{\partial E_n}{\partial w_{kj}} = \delta_k z_j \text{ and } \frac{\partial E_n}{\partial w_{ji}} = \delta_j x_i$$

13/16

324

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.

325

Regularization

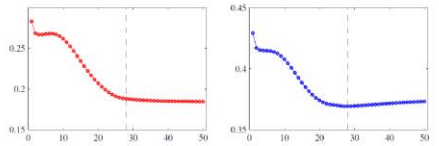


Figure 5.12 An illustration of the behaviour of training set error (left) and validation set error (right) during a typical training session, as a function of the iteration step, for the sinusoidal data set. The goal of achieving the best generalization performance suggests that training should be stopped at the point shown by the vertical dashed lines, corresponding to the minimum of the validation set error.

- Regularization when learning the parameters: Early stopping the backpropagation algorithm according to the error on some validation data.
- Regularization when learning the structure:
  - Cross-validation.
  - Penalizing complexity according to

$$E(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|^2 \text{ or } E(\mathbf{w}) + \frac{\lambda_1}{2}\|\mathbf{w}^{(1)}\|^2 + \frac{\lambda_2}{2}\|\mathbf{w}^{(2)}\|^2$$

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.

326

732A99/TDDE01 Machine Learning

Lecture 3d Block 1: Deep Learning

Jose M. Peña
IDA, Linköping University, Sweden

327

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 $p(\mathbf{x}, t)$,

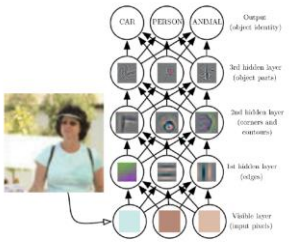
$$\lim_{k \to \infty} \inf_{y \in \mathcal{C}^{(k)}} L(y(\mathbf{x})) - L(p(t|\mathbf{x})) = 0$$

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.

328

Deep Neural Networks



- A deep NN is a function that maps input to output.
- The mapping is formed by composing many simpler functions.
- Each layer provides a new representation of the input, i.e. complex concepts are built from simpler ones.
- The representation is learned automatically from data.

329

Deep Neural Networks

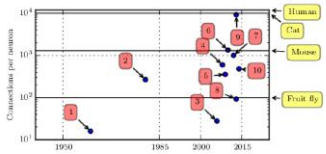


Figure 1.10: Initially, the number of connections between neurons in artificial neural networks was limited by hardware capabilities. Today, the number of connections between neurons is mostly a design consideration. Some artificial neural networks have nearly as many connections per neuron as a cat, and it is quite common for other neural networks to have as many connections per neuron as smaller mammals like mice. Even the human brain does not have an exorbitant amount of connections per neuron. Biological neural network sizes from Wikipedia (2015).

1. Adaptive linear element (Widrow and Hoff, 1960)
2. Neocognitron (Fukushima, 1980)
3. GPU-accelerated convolutional network (Chellapilla et al., 2006)
4. Deep Boltzmann machine (Salakhutdinov and Hinton, 2009a)
5. Unsupervised convolutional network (Jarrett et al., 2009)
6. GPU-accelerated multilayer perceptron (Ciresan et al., 2010)
7. Distributed autoencoder (Le et al., 2012)
8. Multi-GPU convolutional network (Krizhevsky et al., 2012)
9. COTS HPC unsupervised convolutional network (Coates et al., 2013)
10. GoogLeNet (Szegedy et al., 2014a)

330

Deep Neural Networks

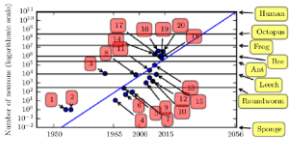


Figure 1.11: Since the introduction of hidden units, artificial neural networks have doubled in size roughly every 2.4 years. Biological neural network sizes from Wikipedia (2015).

1. Perceptron (Rosenblatt, 1958, 1962)
2. Adaptive linear element (Widrow and Hoff, 1960)
3. Neocognitron (Fukushima, 1980)
4. Early back-propagation network (Rumelhart et al., 1986b)
5. Recurrent neural network for speech recognition (Robinson and Fallside, 1991)
6. Multilayer perceptron for speech recognition (Bengio et al., 1991)
7. Mean field sigmoid belief network (Saul et al., 1996)
8. LeNet-5 (LeCun et al., 1998b)
9. Echo state network (Jaeger and Haas, 2004)
10. Deep belief network (Hinton et al., 2006)
11. GPU-accelerated convolutional network (Chellapilla et al., 2006)
12. Deep Boltzmann machine (Salakhutdinov and Hinton, 2009a)
13. GPU-accelerated deep belief network (Raina et al., 2009)
14. Unsupervised convolutional network (Jarrett et al., 2009)
15. GPU-accelerated multilayer perceptron (Ciresan et al., 2010)
16. OMP-1 network (Coates and Ng, 2011)
17. Distributed autoencoder (Le et al., 2012)
18. Multi-GPU convolutional network (Krizhevsky et al., 2012)
19. COTS HPC unsupervised convolutional network (Coates et al., 2013)
20. GoogLeNet (Szegedy et al., 2014a)

22 layers DNN, but 12 times fewer weights than DNN 19

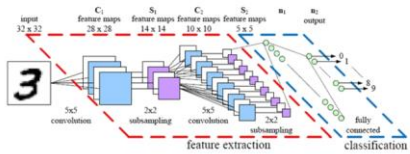
331

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".

332

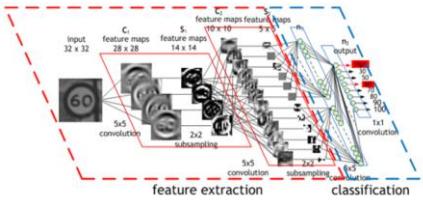
Convolutional Networks



- 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.

333

Convolutional Networks



334

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.

335

Rectifier Activation Function

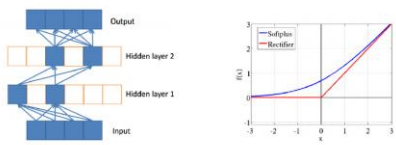


Figure 2: *Left:* **Sparse propagation of activations and gradients in a network of rectifier units.** The input selects a subset of active neurons and computation is linear in this subset. *Right:* **Rectifier and softplus activation functions.** The second one is a smooth version of the first.

- *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.

336

Rectifier Activation Function

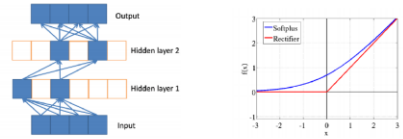


Figure 2: *Left:* **Sparse propagation of activations and gradients in a network of rectifier units.** The input selects a subset of active neurons and computation is linear in this subset. *Right:* **Rectifier and softplus activation functions.** The second one is a smooth version of the first.

- 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.

14/16

337

Layer-Wise Pre-Training

- The pre-training aims to find a good starting point for the subsequent run of the backpropagation algorithm.
- Supervised version:
  1. Train each layer of the DNN as if it was the hidden layer in a depth-two NN. As input, use the output of the last of the previously trained layers. As output, use the original classification or regression function.
  2. Run the backpropagation algorithm to fine-tune the weights.
- Unsupervised version: Similar to the supervised one but the hidden layers (except the last one) are trained to learn an encoding of the output of the previous layer, instead of the original classification or regression function.

15/16

338

Bra funktioner:

```
missclass_rate = function(title ,v1, v2) {
    t_table = table(v1, v2)
    missclass = 1-sum(diag(t_table))/sum(t_table)
     print(paste0("missclassification rate for ", title, ": ", missclass))
}
conf_matrix = function(title, true, predicted) {
        t_table = table(true, predicted)
        print(title)
        print(t_table)
}
mean_square_error = function(v1, v2) {
        SE = (v1-v2)
        SE = SE^2
        SE = mean(SE)
        return(SE)
}
kernel_gauss = function(diff, h_val) {
    U = diff / h_val U = U^2 return(exp(-U))
}
kernal_plot = function(diff, h_val) {
        kernal = kernel_gauss(diff, h_val)
        plot(kernal, type="l")
}
kernal_plot(seq(0,250000,1), h_distance)
mod = function(Lside, Rside) {
      returnValue = Lside - Rside * floor(Lside/Rside)
      return(returnValue)
}

# Filter out all dates after the requested date
sts = sts[as.Date(sts$date) < as.Date(date_in),] # Converts the time column from string to time, makes comparison easier later
sts$time = strptime(sts$time, format = "%H:%M:%S")
# Remove all saved variables and plots
dev.off() rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)
# Vector av nollor v1 = rep(0, antal)
#Random
Var <- runif(50, 0, 10) # Fördelat
rand_obs = rexp(50, rate = 1.13) #??

#Lab 1 - Assignment 1
#Should reset variables at beginning of run (eases tracability and rerun when solving the problems with code)
rm(list=ls())

#Import the necessary spambase data
#data <- read_excel("E:/Machinelearning/spambase.xlsx")
data=read.csv2("spambase.csv")

#Step 1 create sampledata devided into sets of training and test even split (provided in assignment)
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]

#------------------------------------------------------
#Step 2
#Use logistic regression (functions glm(), predict()) to classify the training and test data by the classification principle
#Logistic regression with glm() and predict() on the sampled data
#pp 1c s 26 result=lm(Y~X,  weights=W,data=mydata)

resulted_model = glm(Spam ~ .,  data=train)
# data=train uses the train=data[id,]
# Target = Column Spam from data and the . means everything BUT the Spam column
```

```
attempt_test=predict(resulted_model, newdata=test)
#resulted_model is the model resulted from previous glm() that will here be used to predict
#Use model to predict on new data. Therefore newdata=test which was created earlier when split available data.
attempt_train=predict(resulted_model, newdata=train)

#Classify training and test data with classification principle
classified_yhat_05_train = ifelse(attempt_train > 0.5, 1, 0)
classified_yhat_05_test = ifelse(attempt_test > 0.5, 1, 0)

#Create confusion matrices for training and test data
cf_matrix_05_train = table(factor(train$Spam, labels=c("The Actual Not Spam", "The Actual Spam")),
                           factor(classified_yhat_05_train, labels= c("The Predict Not Spam", "The Predict Spam")))

cf_matrix_05_test = table(factor(test$Spam, labels=c("The Actual Not Spam", "The Actual Spam")),
factor(classified_yhat_05_test, labels= c("The Predict Not Spam", "The Predict Spam")))

print(cf_matrix_05_train)
print(cf_matrix_05_test)

#Annans
train_conf_mat1 <- table(actual = train$Spam, predicted = classified_yhat_05_train)
test_conf_mat1      <-table(actual = test$Spam, predicted  = classified_yhat_05_test)
print(train_conf_mat1)
print(test_conf_mat1)

#Report on missclassification rates (using own function)
#Create missclassification function (provided in lecture 1c)

missclassification_rate=function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}

missclassified_05_train=missclassification_rate(train$Spam, classified_yhat_05_train)
missclassified_05_test=missclassification_rate(test$Spam, classified_yhat_05_test)

print(paste0("Missclassification rate for train:", missclassified_05_train))
print(paste0("Missclassification rate for test:", missclassified_05_test))

# The train provides slightly better result than test, due to better fit on train data since trained on it. This can be seen in the cf
matrixes and lower missclassification rates for train.

#-------------------------------------------------------
#Step 3

#Classify training and test data with classification principle based on information from STEP3
classified_yhat_08_train = ifelse(attempt_train > 0.8, 1, 0)
classified_yhat_08_test = ifelse(attempt_test > 0.8, 1, 0)

#Create confusion matrices for training and test data

cf_matrix_08_train = table(factor(train$Spam, labels=c("The Actual Not Spam", "The Actual Spam")),
factor(classified_yhat_08_train, labels= c("The Predict Not Spam", "The Predict Spam")))

cf_matrix_08_test = table(factor(test$Spam, labels=c("The Actual Not Spam", "The Actual Spam")),
factor(classified_yhat_08_test, labels= c("The Predict Not Spam", "The Predict Spam")))

print(cf_matrix_08_train)
print(cf_matrix_08_test)

#Report on missclassification rates (using own function)
missclassified_08_train=missclassification_rate(train$Spam, classified_yhat_08_train)
missclassified_08_test=missclassification_rate(test$Spam, classified_yhat_08_test)

print(paste0("Missclassification rate for train:", missclassified_08_train))
print(paste0("Missclassification rate for test:", missclassified_08_test))

# The test provides slightly better results than train as can be seen by lower missclassification rate.
# Due to the new rule more missclassifactions of spam is made
```

```
#-------------------------------------------------------

#Step 4
#Use standard classifier kknn() with K=30 from package kknn, report the the misclassification rates for the training and test data and compare the results with step 2.
#KKNN = Weighted k-Nearest Neighbor Classifier
library("kknn")
kknn_model_with_k_30_test = kknn(Spam~., train=train, test=test, k=30)
kknn_model_with_k_30_train = kknn(Spam~., train=train, test=train, k=30)

#Classify training and test data with classification principle based on information from STEP 2

classified_kknn30_test = ifelse(kknn_model_with_k_30_test$fitted.values > 0.5, 1, 0)
classified_kknn30_train = ifelse(kknn_model_with_k_30_train$fitted.values > 0.5, 1, 0)



#Report on missclassification rates (using own function)

missclassified_kknn30_test=missclassification_rate(test$Spam,classified_kknn30_test)
missclassified_kknn30_train=missclassification_rate(train$Spam,classified_kknn30_train)

print(paste0("Missclassification rate for train:", missclassified_kknn30_train))
print(paste0("Missclassification rate for test:", missclassified_kknn30_test))

#Compare with STEP 2
# Greater disparite between missclassification rates between test and train in kkn than in those in STEP2
# Train missclassifies less than test
# Train for KKNN has lower missclassification rate than train and test in step 2 while test in kknn has the highest missclassification rate in the comparison

#-------------------------------------------------------

#Step 5

#Repeat step 4 for K=1

kknn_model_with_k_1_test = kknn(Spam~., train=train, test=test, k=1)
kknn_model_with_k_1_train = kknn(Spam~., train=train, test=train, k=1)
classified_kknn1_test = ifelse(kknn_model_with_k_1_test$fitted.values > 0.5, 1, 0)
classified_kknn1_train = ifelse(kknn_model_with_k_1_train$fitted.values > 0.5, 1, 0)

missclassified_kknn1_test=missclassification_rate(test$Spam,classified_kknn1_test)
missclassified_kknn1_train=missclassification_rate(train$Spam,classified_kknn1_train)

print(paste0("Missclassification rate for train:", missclassified_kknn1_train))
print(paste0("Missclassification rate for test:", missclassified_kknn1_test))

# K= 1 leads to a 0 in missclassification rate for training data.
# Data is classified by closest neighbour which is of the same data set.# Missclassification rate increases slightly compared to that of STEP 4 likely due to using 1 neighbout achieving an effect similar to overfitting a model.

#Lab 1 - Assignment 2----------------------------------------------------------------------------------------------------

#Should reset variables at beginning of run (eases tracability and rerun when solving the problems with code)
rm(list=ls())
#Remove all plots(eases tracability and rerun when solving the problems with code)
dev.off()

#-------------------------------------------------------
#Step 1
#Import the necessary machines data
data=read.csv2("machines.csv")

#-------------------------------------------------------

#Step 2
#Probability model - Exponential distribution p(xIz)=z*e^-zx
```

```
#Create log-likelihood function of expression above.
#Function that computes loglikelihood

loglikelihood = function(theta_var, x){
  return(length(x)*log(theta_var)- (theta_var*sum(x)))
}

#Create a vector theta
sized_theta = seq(10^(-7), 4, by=0.0001)

#Calculate log values based on theta and x = data$Length
log_values = vector(mode="numeric", length=0)

for(i in sized_theta) {
  log_values = c(log_values, loglikelihood(i, data$Length))
}

#Plot the the dependence of loglikelihood on theta
plot(sized_theta, log_values, type="l", col="purple", xlim=c(0, 4), ylim = c(-200,0))

#theta providing maximum loglikelihood in plot, according to plot?
print(sized_theta[which.max(log_values)])
#Maximum loglikelihood of said theta
print(max(log_values))

#---------------------------------------------------------------------

#STEP 3
#Repeat STEP2 But with only first 6 data entries.

first_six_data = data[c(1:6),]
first_six_log_values = vector(mode="numeric", length=0)

for(i in sized_theta) {
  first_six_log_values = c(first_six_log_values, loglikelihood(i, data$Length[1:6]))

}

#Plot the the dependence of loglikelihood on theta (on same plot as STEP2)
lines(sized_theta, first_six_log_values, col="red")

#theta providing maximum loglikelihood in plot, according to plot?
print(sized_theta[which.max(first_six_log_values)])

#Maximum loglikelihood of said theta
print(max(first_six_log_values))

# More data gives more reliable log-likelihood function thus more reliable in step 2 than step 3

#---------------------------------------------------------------------
#STEP 4 theta = z, y = lamb (lambda)
# Assume now a Bayesian model with p(xIz)=z*e^-zx and a prior p(z) = y*e^-yz
# Write a function computing l(theta) = log(p(xIz)*p(z))

calc_l_of_theta = function (theta_var, x, lamb){
  exp_log=loglikelihood(theta_var,x)
  lamb_log= log(lamb)-lamb*theta_var
  return(exp_log+lamb_log)
}

# Create vector for l(theta)
all_theta_l_values = vector(mode="numeric", length=0)
for(i in sized_theta) {
  all_theta_l_values = c(all_theta_l_values, calc_l_of_theta(i,data$Length,10))

}

#Plot the curve showing the dependence of l(theta) on theta
lines (sized_theta, all_theta_l_values, col="turquoise")
```

```
#theta providing maximum loglikelihood in plot, according to plot?
print(sized_theta[which.max(all_theta_l_values)])
#Maximum loglikelihood of said theta
print(max(all_theta_l_values))

# Equeation is a simplified bayesian model that calculates probability of x given theta.
# Otimal theta = 0,91 in this step compared to step 2 which was higher at theta = 1.1262.
# Displays in the plot as a turquoise line.

#---------------------------------------------------------------------
#STEP 5
# Generate 50 new observations from p(xIz)=z*e^-zx with theta from STEP 2
set.seed(12345)
random_generated_exp=rexp(50, sized_theta[which.max(log_values)])

#Create histograms from orignial and new data
hist(random_generated_exp,
    col="yellow",
    xlim=c(0,6),
    ylim=c(0,30),
    main="Histogram showing overlapp")

hist(data$Length,
    col="blue",
    add=T)
#Create histograms from original and new data
hist(data$Length,
    col="blue",
    xlim=c(0,6),
    ylim=c(0,30),
   main="Histogram showing overlapp 2")

#The plot "histogram" displays frequency of data between the valuies on the x-axis. By using the seed recommended for the labs, the plots displays a higher frequency for lower values on the x-axis for both the orignial data an the generated predictions (though the randomly generated predictsions appear to have higher frequency as displayed by the overlapp)

#2.5 Try to use the same bins in the histogram and maybe you will come to different conclusion The left histogram contains the generated samples and the right one contains the original data. The generated samples seems to be fairly similar to the original ones, i.e., the found theta seems to work.

#The right histogram however, a randomly data set of 50 values is generated. The data is not completely random, but generated using an exponential distribution given the theta estimated from the 48 given data values. This of course would imply that the data sets look similar when plotted in a histogram. However this does not necessarily imply that they are representative of the entire population since the posteror iprobability was neglected for this measure.

#Lab 1 - Assignment 3 # Lab1Task3 Cross Validation  ----------
# Assignment 3: Feature selection by cross-validation in a linear model

#Should reset variables at beginning of run (eases tracability and rerun when solving the problems with code)
rm(list=ls())
#Remove all plots(eases tracability and rerun when solving the problems with code)
dev.off()

#Implement an R function that performs feature selection (best subset selection) in linear regression by using k-fold cross-validation without using any specialized function like lm() (use only basic R functions). Your function should depend on:
# X: matrix containing X measurements.
# Y: vector containing Y measurements
# Nfolds: number of folds in the cross-validation.
# You may assume in your code that matrix X has 5 columns.
# The function should plot the CV scores computed for various feature subsets against the number of features, and it should also return the optimal subset of features and the corresponding cross-validation (CV) score.  Before splitting into folds, the data should be permuted, and the seed 12345 should be used for that purpose.

# Import data and libraries
data = swiss
# Function to estimate the linear regression model. Predicts the parameters w for the features X, using the hat matrix
```

```
linrhat <- function(X, y) {
  return(solve(t(X)%*%X)%*%t(X)%*%y)
}

# Function that returns a matrix with indexes of the X data into k rows
kFoldMatrix = function(X, k) {
  rows = k
  columns = floor(NROW(X)/k)
  index = rows*columns  folds = matrix(data = NA, rows, columns)
  irandom = sample(1:index, replace = FALSE)
  for (i in 1:index) {
    folds[i] = irandom[i]
  }
  return(folds)
}

# Function to compute the model error (MSE)
mseError = function(x,y) {
  l = length(x)
  diff = sum((x-y)^2)
  return(diff/l)
}

# Function that computes the cross-validation error when performing k-fold cross-validation

cvError = function(X, y, k) {
  kFoldMatrix = kFoldMatrix(X, k)
  error = c()
  errorList = c()

  for(i in 1:k){
    # Remove one row as test set and the other as training sets
    test = kFoldMatrix[i,]
    train = kFoldMatrix[-i]
    # The training data
    Xi = X[train,]
    yi = y[train]
    # The parameters w
    w = linrhat(Xi, yi)

    #Compute the predicted values
    Yfit = Xi%*%w

    #Estimate the error
    error = mseError(Yfit, yi)
    errorList = c(errorList, error)

  }

  return(mean(errorList))

}

# Function that performs feature selection for X by minimizing MSE
# using k-fold cross-validation
featureSelection = function(X, y, k) {
  # Data frame to store the
  cvErrorList = c()
  featureList = list()
  count = 1
  # All possible combinations of features
  for (i in 1:ncol(X)) {
    fiComb = t(combn(1:ncol(X), i))
    for (row in 1:nrow(fiComb)){
      # Extract the selected features X
      Xi = as.matrix(X[,fiComb[row,]])
```

```
    featureList[[count]] = fiComb[row,]
    count = count + 1
    # Compute the cross-validation error for the selected features
    cvErrorList = c(cvErrorList, cvError(Xi, y, k))
   }
  }
# Create a data frame that contains the features and the cross-validation error
best = which.min(cvErrorList)
 print(best)
 featureAndError = data.frame(featureList = I(featureList), cvErrorList = cvErrorList)
 return(featureAndError)

}

#STEP 2
# Test your function on data set swiss available in the standard R repository:
```

- Fertility should be Y
- All other variables should be X
- Nfolds should be 5

```
# Report the resulting plot and interpret it. Report the optimal subset of features and comment whether it is reasonable that these specific features have largest impact on the target.

# Data
set.seed(12345)
X = data.matrix(data[,-1])
y = data.matrix(data[,1])
i = featureSelection(X, y, 5)

plot(lengths(i$featureList), i$cvErrorList, col="blue", pch=19,
    main="CV Error vs Number of Features", xlab = "Number of features", ylab = "CV Error")

print(i[31,]) # ger dock 5

#Selected Features = Agriculture, Education, Catholic, Infant.Mortality The optimal model has a subset of 4 features - Agriculture, Education, Catholic and Infant.Mortality. Itis reasonable that this subset of features has the largest impact on the target Fertility because this subsethas the lowest MSE. This can be explained because the fifth feature Examination which is dropped in thebest model is highly correlated with all the other features (predictors). High multicollinearity reduces theprecision of the estimated coefficients (increases standard error) as it becomes harder to determine the effect of individual predictors on the target. Hence, eliminating this highly correlated feature results in better estimates of the target for changes in each of the remaining individual predictors and lowers the MSE of the model predictions. It can also be observed from the graph that with an increase in the number of features/ predictors in the model (complexity), the MSE reduces.

#Lab 1 - Assignment 4----------

#Should reset variables at beginning of run (eases tracability and rerun when solving the problems with code)
rm(list=ls())

#Remove all plots(eases tracability and rerun when solving the problems with code)
dev.off()

#Import the necessary machines data
data=read.csv2("tecator.csv")

#-----------------------------------------------------------------

#STEP 1
#Import data to R and create a plot of Moisture versus Protein. Do you think that these data are described well by a linear model?

plot(data$Moisture, data$Protein,
    main="Moisture versus Protein",
    ylab="Moisture",
    xlab="Protein")

#Appears as if a linear model describes the relation between protein and moisture rather well
```

```
#-------------------------------------------------------

#Step 2
#Report a probabilistic model that describes Mi?
#  p(β) =N(B|my0,sumsymbol0) where assumption of parameters are Gaussian distributed. Where some some covariance sumzymbol0 and mean my0.

# Why is it appropriate to use MSE criterion when fitting this model to a training data?

#M = SUM(Ak*X^K), K= 0..n -> Mi = Ai*X^i + A(i-1)*X^(i-1) + ... + A1+X^1 + A0

# MSE can be used as an unbiased estimate of error variance. MSE is the average squared difference between the estimated values and the actual value. Measures of an estimator, the closer to zero values are, the better. MSE is useful here to show which models causes the least error variance of the models.

#-------------------------------------------------------
#Step 3

#Create sampledata divided into sets of training and test even split (use same as Assignment 1) Divide the data into training and validation sets( 50%/50%) and fit models  Mi =1…6.  For each model, record the training and the validation MSE and present a plot showing how training and validation MSE depend on I (write some R code to make this plot). Which model is best according to the plot? How do the MSE values change and why? Interpret this picture in terms of bias-variance trade off.

n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
val=data[-id,]

model_m1 = lm(Moisture ~ poly(Protein, 1), data = train)
model_m2 = lm(Moisture ~ poly(Protein, 2), data = train)
model_m3 = lm(Moisture ~ poly(Protein, 3), data = train)
model_m4 = lm(Moisture ~ poly(Protein, 4), data = train)
model_m5 = lm(Moisture ~ poly(Protein, 5), data = train)
model_m6 = lm(Moisture ~ poly(Protein, 6), data = train)

# M4 = glm(train$Moisture~train$Protein+I(train$Protein^2)+I(train$Protein^3)+I(train$Protein^4), data=train, family="gaussian")

#Create MSE function (Mean squared error) where MSE = 1/n * SUM,1-n,((Yi-^Yi)^2)
#Prediction vector = pv (containing Y)
#Correct vector (test data) = cd (Containing ^Y)
#n = length of vector
MSE_of=function(pv,cd){
  diff_total=0
    for (i in 1:length(pv)) { #Go through entire vectors
      diff_total = diff_total + (pv[i]-cd[i])^2 # The difference squared = (pv[i]-cd[i])^2

    }

    diff_total = diff_total/length(pv)
    return (diff_total)

}

#model_mi is the model resulted from previous lm() that will here be used to predict
#Use model to predict on new data.

attempt_predict_1 = predict(model_m1, newdata=val[1:107,])
attempt_predict_2 = predict(model_m2, newdata=val[1:107,])
attempt_predict_3 = predict(model_m3, newdata=val[1:107,])
attempt_predict_4 = predict(model_m4, newdata=val[1:107,])
attempt_predict_5 = predict(model_m5, newdata=val[1:107,])
attempt_predict_6 = predict(model_m6, newdata=val[1:107,])

attempt_predict_1t = predict(model_m1, newdata=train[1:107,])
attempt_predict_2t = predict(model_m2, newdata=train[1:107,])
attempt_predict_3t = predict(model_m3, newdata=train[1:107,])
attempt_predict_4t = predict(model_m4, newdata=train[1:107,])
```

```
attempt_predict_5t = predict(model_m5, newdata=train[1:107,])
attempt_predict_6t = predict(model_m6, newdata=train[1:107,])

#Calculate mean squared error for each of the predictions compared to the Moisture in the val data

mse1 = MSE_of(attempt_predict_1,val[1:107,]$Moisture)
mse2 = MSE_of(attempt_predict_2,val[1:107,]$Moisture)
mse3 = MSE_of(attempt_predict_3,val[1:107,]$Moisture)
mse4 = MSE_of(attempt_predict_4,val[1:107,]$Moisture)
mse5 = MSE_of(attempt_predict_5,val[1:107,]$Moisture)
mse6 = MSE_of(attempt_predict_6,val[1:107,]$Moisture)

#Calculate mean squared error for each of the predictions compared to the Moisture in the train data

mse1t = MSE_of(attempt_predict_1t,train[1:107,]$Moisture)
mse2t = MSE_of(attempt_predict_2t,train[1:107,]$Moisture)
mse3t = MSE_of(attempt_predict_3t,train[1:107,]$Moisture)
mse4t = MSE_of(attempt_predict_4t,train[1:107,]$Moisture)
mse5t = MSE_of(attempt_predict_5t,train[1:107,]$Moisture)
mse6t = MSE_of(attempt_predict_6t,train[1:107,]$Moisture)

#Plot how training and validation MSE depend on i
plot(1:6, c(mse1,mse2,mse3,mse4,mse5,mse6),
    type="o",
    col="purple",
    main="MSE dependent on i",
    ylab="MSE",
    xlab="i",
    ylim = c(20,45))

lines(1:6,c(mse1t,mse2t,mse3t,mse4t,mse5t,mse6t), col="turquoise")

#According to plot below, model_m3 on the validation data appears to be the best suitable model as MSE is the lowest for
model_m3 (i=3) of the 6 models shown. However, for the train data the best model appears to be i = 6. While values over i = 6
does lower the MSE for the train data it appears to increase it for the validation data. This is likely due to the model becoming
overfitted. Bias and variance are linked to the extent that a high bias correlates with low variance and vice versa.

# High bias also implies underfitting while a high variance instead would lead to a overfitting. Looking at the plot I = 1 leads to
high bias and low variance. Meanwhile I = 6 has a high variance and low bias. After I = 3, the plot displays an increase in variance,
rather significantly as the model is overfitted.

#-------------------------------------------------------
#Step 4

#Perform variable selection of a linear model in which Fat is response and Channel1-Channel100 are predictors by using stepAIC.
Comment on how many variables were selected.'

#Fat is response
#Channel 1-100 are predictors
# In accordance with ??stepAIC only k = 2 gives the genuine AIC.

library(MASS)

data_channels = data[, 2:102]
model_channels = lm(data$Fat ~., data=data_channels)
step = stepAIC(model_channels, K=2)
stepaicCoef = coef(step)
print(length(stepaicCoef))

# Prints 64 variables selected (looking at steps taken by search)
# By looking at the number of coefficients used in the result we can thusly determine that 64 variables were selected.

#-------------------------------------------------------
#Step 5
# Fit a Ridge regression model with the same predictor and response variables. Present a plot showing how model coefficients
depend on the log of the penalty factor lambda and report how the coefficients change with lambda.

#Ridge regression
```

```
library("glmnet")

r_model = glmnet(as.matrix(data_channels[,1:100]), data_channels[,101], alpha=0, family="gaussian")
plot(r_model, xvar="lambda", label=TRUE, ylab="Ridge model coefficients", xlab ="Log Lambda (log of the penalty factor)")

#Lambda is a penalty factor. According to plot, an increase in the value of lambda leads to smaller values of the coefficients.This larger lambda mean that each individual X affects the prediction to a lesser extent.

#-------------------------------------------------------
#Step 6
#Repeat STEP 5 but with Lasso. Compare ridge and lasso.

l_model = glmnet(as.matrix(data_channels[,1:100]), data_channels[,101], alpha=1, family="gaussian")

plot(l_model, xvar="lambda", label=TRUE, ylab="Lasso model coefficients")

# Similar to step 5 the coefficients go towards 0 along with the increase in value of lambda. However the pattern of the lines of the plots differ between step 5 and 6. The ridge model follows a more similar pattern between the different lines while LASSO cause erratic spikes in the different curves plotted along the different log lambda values. Thus, LASSO does not follow a pattern as even as Ridge.

#Compared to ridge, lasso have the regularization term as an absolute value. This impacts trade-off. Lasso does not only punish high values like ridge but also set coefficient as 0 if they are essentially irrelevant. Due to this we end up with fewer features that are included in the lasso model than what we started with.

#-------------------------------------------------------
#Step 7

#Use cross-validation to find the optimal LASSO model (make sure that case lambda = 0 is also considered by the procedure), report the optimal lambda and how many variables were chosen by the model and make conclusions. Present also a plot showing the dependence of the CV score and comment how the CV score changes with lambda.

#Cross validation for optimal lasso-model
#Create sequence of lambda values

lambda_seq = seq(from=0, to=5, by=0.001)
cross_val_lasso = cv.glmnet(as.matrix(data_channels[,1:100]), data_channels[,101], alpha=1, lambda=lambda_seq,
family="gaussian")

#coef(lasso.model, s= "lambda.min" )
cvCoef = coef(cross_val_lasso, s="lambda.1se")

#Optimal lambda is 0 which means that all variables should be selected.

#By looking at the result of the cross-validated lasso model, it shows that the optimal lambda as = 0.007. By using the coef() function we can then find the amount of variables selected, which is 30 int this case. The lambda that gives the minimum mean cross-validated error is lambda = 0.  However, the lambda that in this case provides the most regularized model and also ensures that the error is within one standard error is lambda = 0.007.

#-------------------------------------------------------
#Step 8
#Compare resultre from 4 o 7

print(cross_val_lasso$lambda.min)
print(which.min(cross_val_lasso$lambda))

#By comparing step 4 and 7 we can see that the stepAIC model ended up selecting 64 variables, compared to the cross-validated Lasso model of step 7 which selected 30 variables. Which is reasonable provided that lasso turn coefficient into 0 if they are essentially irrelevant

# lab2 Assignment 1#----------------------------------------------------#----------------------------------------------

#Should reset variables at beginning of run (eases tracability and rerun when solving the problems with code)
rm(list=ls())
#Remove all plots(eases tracability and rerun when solving the problems with code)
dev.off()
RNGversion("3.5.1")
#-------------------------------------------------------
#Step 1
```

Use australian-crabs.csv and make a scatterplot of carapace length (CL) versus rear width (RW) where observations are colored by Sex. Do you think that this data is easy to classify by linear discriminant analysis? Motivate your answer.

```
#Import the necessary crab data
data= read.csv("australian-crabs.csv", header = TRUE, sep = ",")

x <- data$CL
y <- data$RW
gender = data$sex

plot(x, y,
    main="Carapace Length versus Rear Width",
    ylab="Carapace Length (CL)",
    xlab="Rear Width (RW)",
    pch = 19,
    col = gender)

# LDA is used to find a linear combination of different features that can characterizes or separate 2 and/or more classes of objects. CL and RW following 2 linear patterns appear to be suitable for this.

#-----------------------------------------------------
#Step 2
```

Make LDA analysis with target Sex and features CL and RW and proportional prior by using lda() function in package MASS. Make a scatter plot of CL versus RW colored by the predicted Sex and compare it with the plot in step 1. Compute the misclassification error and comment on the quality of fit.

```
library(MASS)
set.seed(12345)
result_LDA = lda(data$sex~data$CL+data$RW, data=data)
print(result_LDA)

prediction = predict(result_LDA,data)

plot(x, y,
    main="LDA - Carapace Length versus Rear Width",
    ylab="Carapace Length (CL)",
    xlab="Rear Width (RW)",
    pch = 19,
    col = prediction$class)

missclassification_rate=function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}
miss = missclassification_rate(data$sex,prediction$class)
print (paste("Missclassification rate: ", miss))


# A misclassification rate of 3.5 % is incredibly low and indicates a good fit which can be seen when comparing the plot of step 1 and 2

#-----------------------------------------------------
#Step 3
```

Repeat step 2 but use priors $p(\text{Male})=0.9$, $p(\text{Female})=0.1$ instead. How did the classification result change and why?

```
set.seed(12345)
result_LDA_prior = lda(data$sex~data$CL+data$RW, data=data, prior=c(0.1,0.9))
print(result_LDA_prior)
prediction_prior = predict(result_LDA_prior,data)
plot(x, y,
    main="LDA - Prior - Carapace Length versus Rear Width",
    ylab="Carapace Length (CL)",
    xlab="Rear Width (RW)",
    pch = 19,
    col = prediction_prior$class)
```

```
miss_prior = missclassification_rate(data$sex,prediction_prior$class)
print (paste("Missclassification rate: ", miss_prior))

# The missclassification rate increased to 8 % from previous 3,5 %. This is due to prior indicating a 90 % ratio of Males and 10% of females while the data is divided 50/50 an thus the prediction leads to predicting much more males than previously

#-------------------------------------------------------
#Step 4
Make a similar kind of classification by logistic regression (use function glm()), plot the classified data and compute the misclassification error. Compare these results with the LDA results. Finally, report the equation of the decision boundary and draw the decision boundary in the plot of the classified data.
result_glm = glm(sex~CL+RW, data=data, family="binomial")
print(result_glm)
prediction_glm = predict(result_glm,data)  > .5
predicted.classes <- ifelse(prediction_glm > 0.5, 1, 0)

plot(data$CL, data$RW,
    col=as.numeric(predicted.classes)+1, pch=19,
    bg=as.numeric(predicted.classes),
    main="GLM - Prediction")

slope <- coef(result_glm)[2]/(-coef(result_glm)[3])
intercept <- coef(result_glm)[1]/(-coef(result_glm)[3])
abline(intercept , slope)
miss_glm = missclassification_rate(data$sex, prediction_glm>0.5)
print (paste("Missclassification rate: ", miss_glm))

# lab2 Assignment 2#----------------------------------------------------#----------------------------------------------
#-----------------------------------------------------
#Step 1

Import the data to R and divide into training/validation/test as 50/25/25: use data partitioning code specified in Lecture 1e.
data=read.csv2("creditscoring.csv")
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.25))
val=data[id2,]
id3=setdiff(id1,id2)
test=data[id3,]
#-----------------------------------------------------
#Step 2

Fit a decision tree to the training data by using the following measures of impurity a. Deviance b. Gini index and report the misclassification rates for the training and test data. Choose the measure providing the better results for the following steps.
library(tree)

missclassification_rate=function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}

tree_missclassification = function(tree_input,X) {
  prediction = predict(tree_input, newdata = X, type="class")
  miss_class = missclassification_rate(prediction, X$good_bad)
  return(miss_class)

}

fit_dev = tree(good_bad ~., data=train, split = c("deviance"))
fit_gini = tree(good_bad ~., data=train, split = c("gini"))
```

```
print (paste("Missclassification rate Deviance - Train: ", tree_missclassification(fit_dev, train)))
print (paste("Missclassification rate Deviance - Test: ", tree_missclassification(fit_dev, test)))

print (paste("Missclassification rate Gini Index - Train: ", tree_missclassification(fit_gini, train)))
print (paste("Missclassification rate Gini Index - Test: ", tree_missclassification(fit_gini, test)))

plot(fit_dev)
text(fit_dev, pretty=0)
 #For this assignment, deviance is chosen due to the lower classification rates on train and test.

#-----------------------------------------------------
#Step 3
```

Use training and validation sets to choose the optimal tree depth. Present the graphs of the dependence of deviances for the training and the validation data on the number of leaves. Report the optimal tree, report it's depth and the variables used by the tree. Interpret the information provided by the tree structure. Estimate the misclassification rate for the test data

```
# Uses fit_dev
trainScore=rep(0,15)
testScore=rep(0,15)

for(i in 2:15) {
  prunedTree=prune.tree(fit_dev,best=i)
  pred=predict(prunedTree, newdata=val, type="tree")
  trainScore[i]=deviance(prunedTree)
  testScore[i]=deviance(pred)

}

plot(2:15, trainScore[2:15], type="b", col="red", ylim=c(250,600), ylab="Score", xlab="Number of leaves")
points(2:15, testScore[2:15], type="b", col="blue")

# Optimal number of leaves 12 as displayed by the graph is actualy 4 had 12
final_Tree=prune.tree(fit_dev, best=4) #4 before 12
plot(final_Tree)
text(final_Tree, pretty=0)
#Depth = 7
print(summary(final_Tree))
#Variables used by the tree: "savings" "duration" "history" "age" "purpose"  "amount" "other" "resident"
#Misclass of test data
final_pred_test = predict(final_Tree, newdata=test, type="class")
miss_pruned_test = missclassification_rate(final_pred_test, test$good_bad)
print (paste("Missclassification rate on test data: ", miss_pruned_test))
table(test$good_bad,final_pred_test)
```

#The tree classifies first on savings and thereafter on the other variables. As such, the savings variable is the most significant variable. Variables used in the tree are: `"savings" "duration" "history"  "age" "purpose" "amount" "other" "resident".`

We have a misclassification rate: 26.8%. The optimal tree looks like the plot. By pruning the tree, it is possible to find the optimal number of leaves, which is 12 in this case as can be seen in the plot below. Therefore, the tree has a depth of 7

```
#-----------------------------------------------------
#Step 4
```

Use training data to perform classification using Naïve Bayes and report the confusion matrices and misclassification rates for the training and for the test data. Compare the results with those from step 3.

```
library(MASS)
library(e1071)

bayes_fit = naiveBayes(good_bad~., data=train)
bayes_fit
fit_fit_train = predict(bayes_fit, newdata=train)
table(fit_fit_train,train$good_bad)
miss_bayes_train = missclassification_rate(fit_fit_train, train$good_bad)
print(paste0("Missclassification rate: ",miss_bayes_train))

fit_fit_test = predict(bayes_fit, newdata=test)
table(fit_fit_test,test$good_bad)
```

```
miss_bayes_test = missclassification_rate(fit_fit_test, test$good_bad)
print(paste0("Missclassification rate: ",miss_bayes_test))
# Confusion matrices above
# These are is higher misclassification rates than that of step 3. Therefore, it can be concluded that Naïve Bayes gives worse classification than tree in this case.

#-------------------------------------------------------
#Step 5
```

Use the optimal tree and the Naïve Bayes model to classify the test data by using the following principle: $Y\_hat = p(Y='good'|X) > \pi$, otherwise y_hat $= 0$ where $\pi = 0.05, 0.1, 0.15, \ldots 0.9, 0.95$. Compute the TPR and FPR values for the two models and plot the corresponding ROC curves.

```
pi = seq(0.05, 0.95, by=0.05)

#Predict data
fit_naive_pred_test = predict(bayes_fit, newdata=test, type="raw")
fit_tree_pred_test =predict(final_Tree, newdata=test, type="vector")

#Probabilities for "good"
fit_naive_pred_test_good = fit_naive_pred_test[,"good"]
fit_tree_pred_test_good = fit_tree_pred_test[,"good"]

#Create vectors
TPR_final_tree = vector("numeric", length=length(pi))
FPR_final_tree = vector("numeric", length=length(pi))
TPR_bayes_fit = vector("numeric", length=length(pi))
FPR_bayes_fit = vector("numeric", length=length(pi))

#Calculate TPR
calculate_TPR = function(input) {
  create_table = table(test$good_bad, input)
  return(create_table[1,1]/sum(create_table[1,]))
}

#Calculate FPR
calculate_FPR = function(input) {
  create_table = table(test$good_bad, input)
  return(create_table[2,1]/sum(create_table[2,]))
}

#Loop through
for(i in seq(1,length(pi))) {
  #Classify with principle, tree
  classify_principle_tree = ifelse(fit_tree_pred_test_good > pi[i], "good", "bad")
  TPR_final_tree[i] = calculate_TPR(classify_principle_tree)
  FPR_final_tree[i] = calculate_FPR(classify_principle_tree)

  #Classify with principle, bayes
  classify_principle_bay = ifelse(fit_naive_pred_test_good > pi[i], "good", "bad")
  TPR_bayes_fit[i] = calculate_TPR(classify_principle_bay)
  FPR_bayes_fit[i] = calculate_FPR(classify_principle_bay)
}

plot(FPR_bayes_fit, TPR_bayes_fit,
     pch=16,
     col="purple",
     main="ROC curves",
     xlim=c(0,1),
     ylim=c(0,1),
     type="b",
     xlab="FPR",
     ylab="TPR")

#Order the tree data to show lines between points
order = data.frame(cbind(FPR_final_tree, TPR_final_tree))
make_order = order[order(FPR_final_tree),]
with(make_order, lines(FPR_final_tree, TPR_final_tree, pch=16, col="orange"))
```

#Conclusion: You find the best classifier as the one with the greatest area beneath its curve. In the plot below we can see that this I the case of the purple line which is the Naïve Bayesian.

#----------

#Step 6

Repeat Naïve Bayes classification as it was in step 4 but use the following loss matrix:

$$L = \begin{matrix} \\ Observed \\ \\ \end{matrix} \begin{matrix} Predicted \\ good \\ bad \end{matrix} \begin{pmatrix} 0 & 1 \\ 10 & 0 \end{pmatrix}$$

and report the confusion matrix for the training and test data. Compare the results with the results from step 4 and discuss how the rates has changed and why.

```
bayes_fit_l = naiveBayes(good_bad~., data=train)
fit_fit_l_train = predict(bayes_fit_l, newdata=train, type="raw")
fit_fit_l_test = predict(bayes_fit_l, newdata=test, type="raw")
loss_matrix = c(10,1)

classify_train = ifelse(loss_matrix[1]*fit_fit_l_train[,2] < loss_matrix[2]*fit_fit_l_train[,1], "good", "bad")
classify_test = ifelse(loss_matrix[1]*fit_fit_l_test[,2] < loss_matrix[2]*fit_fit_l_test[,1], "good", "bad")

cf_matrix_train = table(train$good_bad,classify_train)
cf_matrix_test = table(test$good_bad,classify_test)

print(cf_matrix_train)
print(cf_matrix_test)
```

Comparing with step 4. The loss matrix punishment forces more of the data to be predicted as bad. The predictions become overcautious. Due to this, very little of the data is predicted as something good. Below are the confusion matrices of the training and test data:

 # lab2 Assignment 3#----------#----------

The data file **State.csv** contains per capita state and local public expenditures and associated state demographic and economic characteristics, 1960, and there are variables

• MET: Percentage of population living in standard metropolitan areas

• EX: Per capita state and local public expenditures ($)

#----------

#Step 1

Reorder your data with respect to the increase of MET and plot EX versus MET. Discuss what kind of model can be appropriate here. Use the reordered data in steps 2-5.

```
library(tree)
library(knitr)
RNGversion('3.5.1')
set.seed(12345)
data2 <-read.csv2("State.csv")
data2 <- data2[order(data2$MET),]
plot(data2$MET,data2$EX, xlab = "MET",ylab = "EX", main = "MET vs. EX")
```

From the scatterplot of MET and EX, we can not find obvious distribution with the data. So, we can try linear regression model or some regression tree model with different parameters. Then decide the optimal model by comparing the training errors and test errors.

#----------

#Step 2

Use package **tree** and fit a regression tree model with target EX and feature MET in which the number of the leaves is selected by cross-validation, use the entire data set and set minimum number of observations in a leaf equal to 8 (setting *minsize* in *tree.control*). Report the selected tree. Plot the original and the fitted data and histogram of residuals. Comment on the distribution of the residuals and the quality of the fit.

```
reg_tree <- tree(EX~MET,data = data2,
                 control =tree.control
                 (nrow(data2), minsize = 8))
# use cv.tree to select optimal tree depth
cvreg_tree <- cv.tree(reg_tree)
plot(cvreg_tree$size,cvreg_tree$dev,type = "b",
    xlab = "tree size",ylab = "dev of cvtree",
    main = "tree size vs. deviation")

# pruned tree with 3 leaves is the best one
pru_regtree <- prune.tree(reg_tree,best = 3)
plot(pru_regtree)
text(pru_regtree, pretty = 0)
```

```
# predict y by pruned tree #y_hat <- predict(reg_tree,newdata = data2) # predict by original tree

y_hat <- predict(pru_regtree,newdata = data2)
ex_y_residual <- data.frame(original_value=data2$EX
                            fitted_value=y_hat,
                            residual=data2$EX-y_hat)

plot(c(1:nrow(data2)),data2$EX,col="red",
    xlab = "observations",ylab = "values of EX",
    main = "origianl value vs. fitted value")
points(c(1:nrow(data2)),y_hat,col="blue")
legend("top",legend = c("original","fitted"),
      col = c("red","blue"),lty = 1:2,cex = 0.4)

ex_residual <- ex_y_residual$residual

hist(ex_residual, prob=TRUE,xlab = "residual",
    main = "Histogram of residual ")
lines(density(ex_y_residual$residual),lwd=3, col="red")
```

From the plot of tree-size against deviations, we choose 3 as the tree size since it has the lowest deciations in all of the tree sizes.

The distribution of the residuals is not normal distribution from the histogram. It shows some skewed tail on right. From the picture of original data and the fitted data, we find the quality of the fit is not very good. Since in the regression tree, we use the mean of data of leaves in the same branch as the prediction value, it may exist significant difference between the original data and fitted data.

```
#-----------------------------------------------------
```

#Step 3

Compute and plot the 95% confidence bands for the regression tree model from step 2 (fit a regression tree with the same settings and the same number of leaves as in step 2 to the resampled data) by using a non-parametric bootstrap. Comment whether the band is smooth or bumpy and try to explain why. Consider the width of the confidence band and comment whether results of the regression model in step 2 seem to be reliable.

```
library(boot)
f <- function(data,index){
  data3 <- data[index,]
  reg_tree <- tree(EX~MET,data = data3,
                   control =tree.control(nrow(data3),
                                         minsize = 8))
  pru_regtree <- prune.tree(reg_tree,best = 3)
  pre_boot <- predict(pru_regtree,newdata = data2)
  return(pre_boot)

}

res <- boot(data2,f,R=1000)
e <- envelope(res) #compute confidence bands,level=0.95(default)
plot(c(1:nrow(data2)),data2$EX,col="black",
    xlab = "observations",ylab = "values of EX",
    main = "95% confidence band with non-parametric bootstrap")
points(c(1:nrow(data2)),y_hat,col="blue")

# plot 95% confidence bands of regression tree model
# the upper confidence band is stored in e$point[1,]
# the lower confidence band is stored in e$point[2,]

points(c(1:nrow(data2)),e$point[2,],col="pink")
points(c(1:nrow(data2)),e$point[1,],col="green")
lines(c(1:nrow(data2)),e$point[2,],col="pink")
lines(c(1:nrow(data2)),e$point[1,],col="green")
```

In the picture, the original values of EX are shown by black points, and predicted values are blue points. Then green points (and lines) are used for the upper confidence band and pink points (and lines) for the lower confidence band. Obviously, the confidence bands are bumpy. That is because the distribution of residual is not normal distribution. Even the confidence bands are not smooth, the predicted values by regression tree model still locate between the upper and lower bands. That means the results of the regression model in step 2 seem to be reliable.

```
#-----------------------------------------------------
```

#Step 4

Compute and plot the 95% confidence and prediction bands the regression tree model from step 2 (fit a regression tree with the same settings and the same number of leaves as in step 2 to the resampled data) by using a parametric bootstrap, assume $Y \sim N(\mu_i, \sigma^2)$ where μ_i are labels in the tree leaves and σ^2 is the residual variance. Consider the width of the confidence band and comment whether results of the regression model in step 2 seem to be reliable. Does it look
like only 5% of data are outside the prediction band? Should it be?

```
mle <- prune.tree(tree(EX~MET,data = data2, control = tree.control(nrow(data2), minsize = 8)),best = 3)

rng <- function(data,mle){
  data3 <- data.frame(EX=data$EX,MET=data$MET)
  n <- length(data3$EX)
  data3$EX <- rnorm(n, mean=predict(mle,newdata=data3),
                           sd(data3$EX-predict(mle,newdata=data3)))
  return(data3)
}

f1 <- function(data3){
  reg_tree <- tree(EX~MET,data = data3,
              control =tree.control(nrow(data3), minsize = 8))

  pru_regtree <- prune.tree(reg_tree,best = 3)
  pre_boot <- predict(pru_regtree,newdata = data2)

  return(pre_boot)

}

res2 <- boot(data2,statistic =f1,R=1000,
          mle = mle, ran.gen = rng, sim = "parametric")

e2 <- envelope(res2)

plot(c(1:nrow(data2)),data2$EX,col="black",
    xlab = "observations",ylab = "values of EX ",
    main = "95% confidence band with parametric bootstrap")

points(c(1:nrow(data2)),y_hat,col="blue")
points(c(1:nrow(data2)),e2$point[2,],col="pink")
points(c(1:nrow(data2)),e2$point[1,],col="green")
lines(c(1:nrow(data2)),e2$point[2,],col="pink")
lines(c(1:nrow(data2)),e2$point[1,],col="green")

# prediction interval band

f2 <- function(data3){
  reg_tree <- tree(EX~MET,data = data3,
              control =tree.control(nrow(data3),
                                    minsize = 8))
  pru_regtree <- prune.tree(reg_tree,best = 3)
  pre_boot <- predict(pru_regtree,newdata = data2)
  n <- length(data2$EX)

  predictEX <- rnorm(n,pre_boot,
                sd(data2$EX-pre_boot))

  return(predictEX)

}

res3 <- boot(data2, statistic = f2, R=1000 ,mle = mle, ran.gen = rng, sim = "parametric")

e3 <- envelope(res3)

#In envelope(res3) : unable to achieve requested overall error rate

plot(c(1:nrow(data2)),data2$EX,col="black",
    xlab = "observations",ylab = "values of EX",
    ylim = c(130,470), # the range(e3$point[1,]) and [2,]
    main = "95% prediction band with parametric bootstrap")
```

```
points(c(1:nrow(data2)),y_hat,col="blue")
points(c(1:nrow(data2)),e3$point[2,],col="pink")
points(c(1:nrow(data2)),e3$point[1,],col="green")
lines(c(1:nrow(data2)),e3$point[2,],col="pink")
lines(c(1:nrow(data2)),e3$point[1,],col="green")
```

The width of the confidence band in a parametric bootstrap is narrower than width in a non-parametric bootstra. Many predicted values are outside of this condfidence band. From this, the results of the regression model in step 2 seem to be unreliable. The prediction band looks great. It looks like only 5% of data are outside the prediction band. It should happend because we get the prediction band from the normal distribution with mean of original value and standard deviation of the residual.

Consider the histogram of residuals from step 2 and suggest what kind of bootstrap is actually more appropriate here.

Because the histogram of residuals is not normally distributed, we suggest that non-parametric bootstrap is more appropriate here.

 # lab2 Assignment 4#----------#----------

The data file **NIRspectra.csv** contains near-infrared spectra and viscosity levels for a collection of diesel fuels. Your task is to investigate how the measured spectra can be used to predict the viscosity.

#----------

#Step 1

Conduct a standard PCA by using the feature space and provide a plot explaining how much variation is explained by each feature. Does the plot show how many PC should be extracted? Select the minimal number of components explaining at least 99% of the total variance. Provide also a plot of the scores in the coordinates (PC1, PC2). Are there unusual diesel fuels according to this plot?

```
data=read.csv2("NIRspectra.csv")

mydata=data
mydata$Viscosity=c()
result = prcomp(mydata)
lambda=result$sdev^2

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

In this case the plot shows mainly 2 variances and thus indicates that 2 PCs should likely be extracted. First 2 components cover 99% of the total variance, "94.635" "5.008". See plot below:

```
screeplot(result)

#In this case the plot shows mainly 2 variances and tus indicates that 2 PCs should likely be extracted

U=result$rotation
head(U)
plot(result$x[,1], result$x[,2], ylim=c(-0.25,0.25), xlim=c(-0.25,2))
#There are two unusal diesel plots as can be seen by the two outliers in the plot to the right
```

#----------

#Step 2

Make trace plots of the loadings of the components selected in step 1. Is there any principle component that is explained by mainly a few original features?

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

#The variance of PC2 appears to be explainable by a few original features as can be seen in the plot where the index aroun 120+ provides much higher values.

#first pca first eigenvector if begin high it thats in the first features direction

As can be seen in the variance of PC2, it seems to be explainable by a few original features as in the plot **Traceplot, PC2** below, indexes 0-100 display relatively low values while indexes approaching 120+ suddenly increases variance and provides much higher values.

#----------

#Step 3

Perform Independent Component Analysis with the number of components selected in step 1 (set seed 12345). Check the documentation for the fastICA method in R and do the following: a. Compute $W' = K \cdot W$ and present the columns of W' in form of the trace plots. Compare with the trace plots in step 2 and make conclusions. What kind of measure is represented by the matrix W'? b. Make a plot of the scores of the first two latent features and compare it with the score plot from step 1.

```
library("fastICA")
set.seed(12345)

fika <- fastICA(mydata,2)
W_fnutt = fika$K%*%fika$W

plot(W_fnutt[,1], main="ICA - Traceplot W', Column1")
plot(W_fnutt[,2],main="ICA - Traceplot W', Column2")

# W' represents the inverted PCs, and is The measure represented by W' is a kind of loadings.
# "ICA - Traceplot W', Column1" is the inversion of "Traceplot, PC1"
# "ICA - Traceplot W', Column2" is the inversion of "Traceplot, PC2"

#b
plot(fika$S[,1], fika$S[,2])
```

Comparing the plot below with that of step 1. This one is inverted from the one in step1. However, it also has a wider spread. It is an estimated source matrix.

```
# lab3 Assignment 1#----------------------------------------------------#----------------------------
```

```
# ------------------ Assignment 1 ---------------------------------
```

Implement a kernel method to predict the hourly temperatures for a date and place in Sweden. To do so, you are provided with the files stations.csv and temps50k.csv. These files contain information about weather stations and temperature measurements in the stations at different days and times. The data have been kindly provided by the Swedish Meteorological and Hydrological Institute (SMHI). You are asked to provide a temperature forecast for a date and place in Sweden. The forecast should consist of the predicted temperatures from 4 am to 24 pm in an interval of 2 hours. Use a kernel that is the sum of three Gaussian kernels: Y The first to account for the distance from a station to the point of interest. Y The second to account for the distance between the day a temperature measurement was made and the day of interest. Y The third to account for the distance between the hour of the day a temperature measurement was made and the hour of interest. Choose an appropriate smoothing coefficient or width for each of the three kernels above. Answer to the following questions: Y Show that your choice for the kernels' width is sensible, i.e. that it gives more weight to closer points. Discuss why your of definition of closeness is reasonable. Y Instead of combining the three kernels into one by summing them up, multiply them. Compare the results obtained in both cases and elaborate on why they may differ. Note that the file temps50k.csv may contain temperature measurements that are posterior. to the day and hour of your forecast. You must filter such measurements out, i.e. they cannot be used to compute the forecast. Feel free to use the template below to solve the assignment.

```
library(geosphere)
stations <- read.csv("stations.csv")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")

# Function to calculate the kernel value
kernel_gauss = function(diff, h_val) {
  U = diff / h_val
  U = U^2
  return(exp(-U))

}

kernal_plot = function(diff, h_val) {
  kernal = kernel_gauss(diff, h_val)
  plot(kernal, type="l")

}

# These values are decided by looking at the plots
h_distance <- 75000
h_date <-14
h_time <-4
kernal_plot(distance_diff <- seq(0,250000,1), h_distance)
kernal_plot(seq(0,30,1), h_date)
kernal_plot(seq(0,24,0.5), h_time)

# Function to calculate mod, required for days
mod = function(Lside, Rside) {
```

```
  returnValue = Lside - Rside * floor(Lside/Rside)
  return(returnValue)

}

# Functions to calculate the differences
calc_distance = function(input, sts) {
  dist = distHaversine(data.frame(sts$longitude,sts$latitude), input)
  return(dist)

}

# Take into consideration the difference in days  between years
# Two days ahead five years ago should give differnce 2 days
# Also that it allows days backwards from 25/02 to 24/02 is 1 day, not 364

calc_day = function(input, sts) {
  days = difftime(sts$date, input)
  days = mod(days, 365)
  days = ifelse(days > 365/2, (365 - days), days)
  return(days)
}

calc_time = function(input, sts) {
  time = as.numeric(difftime(input, sts$time, units = "secs"))
  time = time /3600
  time = ifelse(abs(time) > 12, (24 -abs(time)), abs(time))
  return(time)
}

# Main function
predict_temp = function(lat_in, long_in, date_in, times_in,sts) {
  # Filter out all dates after the requested date
  sts = sts[as.Date(sts$date) < as.Date(date_in),]
  # Converts the time column from string to time, makes comparison easier later
  sts$time = strptime(sts$time, format = "%H:%M:%S")
  times_in = strptime(times_in, format = "%H:%M:%S")
  date_in = as.Date(date_in)
  # Calculating the difference and kernel values for distance and date
  distance = calc_distance(c(long_in, lat_in),sts)
  day = calc_day(date_in, sts)
  kernel_distance = kernel_gauss(distance, h_distance)
  kernel_day = kernel_gauss(day, h_date)
  temps_sum = rep(0,length(times_in))
  temps_mul = rep(0,length(times_in))

  for(i in 1:length(times_in)) {
    # Calculating the difference and kernel values for time
    time = calc_time(times_in[i], sts)
    kernel_time = kernel_gauss(time, h_time)

    # Summing/ multiplying all kernel value and calculate the predicted temperature
    k_sum = kernel_distance+kernel_day+kernel_time
    k_mul = kernel_distance*kernel_day*kernel_time
    temps_sum[i] = sts$air_temperature%*% k_sum / sum(k_sum)
    temps_mul[i] = sts$air_temperature%*% k_mul /sum(k_mul)

  }

  #Plot and print result
  plot(temps_sum, type="o"
  print("With summerizing:")
  print(temps_sum)
  plot(temps_mul, type="o")
  print("With multiplying:")
  print(temps_mul)
```

```
}
# Inputs from user
lat <- 58.4274
long <- 14.826
date <- "2013-11-04"
times <- c("04:00:00", "06:00:00", "08:00:00", "10:00:00", "12:00:00", "14:00:00", "16:00:00", "18:00:00", "20:00:00",
"22:00:00", "24:00:00")

# Give predictions
predict_temp(lat, long, date, times, st)
# Time to plot the results
plot(temperature_with_sum, type="o", xlab="Time Index", ylab="Temp", xaxt = "n", main = "The sum of the kernels (temp of
selected date)")
axis(1, at=1:length(times), labels=times)
plot(temperature_with_multiplication, type="o", xlab="Time Index", ylab="Temp", xaxt = "n", main = "The product of the
kernels (temp of selected date)")
axis(1, at=1:length(times), labels=times)
```

Initially decide smoothing coefficient / width for all of the three kernel functions we are to use. I choose the width by trying different values and plotting these. For this assignment I used the width 75000 for distance, 14 days and 4 hours. As such we only take measurements taken within a 75 km distance from out point of interest, 14 days from the desired date and 4 hours within the time of interest. Which seems rather sensible, stray to far away from the point/date/time of interest and the prediction loses some relevance. Below we can see the plot of the kernel functions as sum and multiplication to represent the temperature for a specific time, date and place. Specific date 2013-11-04. The plots differ between summing up and multiplying the kernels. This could be due to the sum adding up the independent kernels thus providing a worse estimation/result along with a lower estimated temperature compared to when multiplying the kernels. Multiplying the kernels makes them dependent on each other for the result which also ensures that a 0 becomes a 0. Strong values then enforce each other while low lowers each other more compared to using addition.

 # lab3 Assignment 2#----------#----------

Use the function ksvm from the R package kernlab to learn a SVM for classifying the spam dataset that is included with the package. Consider the radial basis function kernel (also known as Gaussian) with a width of 0.05. For the C parameter, consider values 0.5, 1 and 5. This implies that you have to consider three models.

- Perform model selection, i.e. select the most promising of the three models (use any method of your choice except cross-validation or nested cross-validation).
- Estimate the generalization error of the SVM selected above (use any method of your choice except cross-validation or nested cross-validation).
- Produce the SVM that will be returned to the user, i.e. show the code.
  What is the purpose of the parameter C ?

```
#install.packages("kernlab")
library(kernlab)
data(spam)
data <- spam
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.70))
train=data[id,]
test=data[-id,]

ksvm.model <- function(x) {
  model <- ksvm(type~., data = train, kernel = "rbfdot", kpar=list(sigma=0.05), C = x)
  pred.model <- predict(model, test)
  confmat <- table(test$type, pred.model)
  misclass <- 1 - sum(diag(confmat)/sum(confmat))
  cat("\nModel misclassification rate for C = ", x, "is: ", misclass)
  confmat

}

ksvm.model(0.5)
ksvm.model(1)
ksvm.model(5)
```

```
model.new <- ksvm(type~., data = train, kernel = "rbfdot", kpar=list(sigma=0.05), C = 1)
model.new
```

According to the results from the 3 models predicted on the test data, the model with *C* = 1 yields the smallest classification error of 0.07458364. Thus, we are going to choose this model for the next step. This is worth noting that the classification error for this model is very close to the model with *C* = 5.

#The purpose of C?

For Support Vector Machines method using Kernel classification, C is a parameter termed as “cost of constraints violation”. ksvm function solves a sequence of sub problems using SMO Algorithm. For n samples, there aree n(n-1)/2 number of possible sub problems. Every sub problem include pair of variables. Usually, there is no need to solve all the sub problems as in some of them have variables that violate the constraints. We define the cost of constraints violation value to filter out the sub problems that have the variables with violations and an optimum model is obtained based on accuracy. In terms of bias-variance trade off, C acts as an adjustment parameter. Larger values of C, results in more variance and consecutively less bias. C penalizes larger residuals so when its value is increased, less residuals are allowed. Thus, variance is increased, allowing less misclassifications.

 # lab3 Assignment 3#----------#----------

Train a neural network to learn the trigonometric sine function. To do so, sample 50 points uniformly at random in the interval [0; 10]. Apply the sine function to each point. The resulting pairs are the data available to you. Use 25 of the 50 points for training and the rest for validation. The validation set is used for early stop of the gradient descent. That is, you should use the validation set to detect when to stop the gradient descent and so avoid overfitting. Stop the gradient descent when the partial derivatives of the error function are below a given threshold value. Check the argument threshold in the documentation. Consider threshold values i/1000 with i = 1,…,10. Initialize the weights of the neural network to random values in the interval [−1, 1]. Use a neural network with a single hidden layer of 10 units. **Use the default values for the arguments not mentioned here**. Choose the most appropriate value for the threshold. Motivate your choice. Provide the final neural network learned with the chosen threshold. Feel free to use the following template.

#----------

As part of the assignment we must choose fitting starting weights. I did this by counting the synaptic constants/connections present in the neural network. See picture below for a picture of the neural network. By counting to 31 connections I created a random vector with 31 entries between -1 and 1.

I choose the best threshold by testing different values of i/1000, i = 1…10. Calculate mean square error to see which i provides the lowest MSE. Use validation set to detect when to stop the gradient descent and avoid overfitting. See 2 figures below. Validation data set provides an mse where I = 4 and the threshold is 0.004. If using the train data set it would be I = 1 and threshold = 0.001 however this would become overfitting as the training data simply wants as low threshold as possible. So chosen threshold is 4/1000. Below is a plot of the result. The sin function created by the neural network as compared to the data. As seen by the closeness of the data and the prediction, it seems to follow rather well.

```
#Setup Sin dataset
library(neuralnet)
set.seed(1234567890)
Var <- runif(50, 0, 10) # 50 according to lab instruction
trva <- data.frame(Var, Sin=sin(Var))
tr <- trva[1:25,] # Training data set
va <- trva[26:50,] # Validation data set

# Random initialization of the weights in the interval [-1, 1]
winit <- runif(31, -1, 1) # see plot of nn, 31 lines between the nodes added together becomes 31.

#Create Mean squared error function
MSE_of=function(pv,cd){ #Used in previous assignments
  diff_total=0
  for (i in 1:length(pv)) {
    diff_total = diff_total + (pv[i]-cd[i])^2
  }
  diff_total = diff_total/length(pv)
  return (diff_total)
}

mse_for_train_dataset <- numeric()
mse_for_validation_dataset <- numeric()
index <- numeric()

for(i in 1:10) { # i = 1..10 in lab instruction
```

```
  # Lab instruction to use a single hidden layer = 10 units
  # Lab instruction to use a i/1000 as threshold value
  nn <- neuralnet(Sin~Var, data=tr, threshold = i/1000, startweights = winit,  hidden = 10)

  #nn <- neuralnet(Sin ~ Var, data= tr, hidden = c(10), startweights = winit, threshold = threshold[i]/ 1000)
  #nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = c(3,3), startweights = winit, threshold = i/1000, lifesign = "full")
  #Solution
  #Predict
  prediction_train = compute(nn, tr)$net.result
  prediction_validation = compute(nn, va)$net.result
  #aux <- compute(nn, va[,1])$net.result
  #resva[i] <- sum((va[,2] - aux)**2)/2
  #Calculate MSE
  mse_for_train_dataset[i] = MSE_of(prediction_train, tr$Sin)
  mse_for_validation_dataset[i] = MSE_of(prediction_validation, va$Sin)
  #Create index later used in MSE plots
  index[i] <- i/1000
}
# Plot MSEs # back propogation train data gives overfitting and search for low threshold
plot(index, mse_for_train_dataset, type="o",
     xlab="Threshold with index i/1000",
     ylab="Mean Square Error of [i]",
     main = "Train data")
plot(index, mse_for_validation_dataset, type="o",  # for better fit with data and validation choose i= 4
     xlab="Threshold with index i/1000",
     ylab="Mean Square Error of [i]",
     main = "Validation data")

# Which i provides smallest MSE?
print(which.min(mse_for_train_dataset))
print(which.min(mse_for_validation_dataset))
#Plot the neural network with i= 4 -> treshhold = 4/1000
plot(nn <- neuralnet(Sin~Var, data=tr, threshold = which.min(mse_for_validation_dataset)/1000, startweights = winit,  hidden =
10)) # TODO which.min(mse_for_train_dataset)
# Plot of the predictions (black dots) and the data (red dots)
plot(prediction(nn)$rep1, main = "Plot of the predictions (black dots) and the data (red dots)")
points(trva, col = "red")

#Exam
# estimate generalization error for the best run above (one layer with threshold 4/1000)
Var <- runif(50, 0, 10)
te <- data.frame(Var, Sin=sin(Var))
winit <- runif(31, -1, 1)
nn <- neuralnet(formula = Sin ~ Var, data = trva, hidden = 10, startweights = winit,
            threshold = 4/1000, lifesign = "full")
sum((te[,2] - compute(nn, te)$net.result)**2)/2
#sum((te[,2] - compute(nn, te[,1])$net.result)**2)/2 ???
#sum((te - compute(nn, te)$net.result)**2)/2 ????

# ----------EXAM 2018----------#----------
The data file video.csv contains characteristics of a sample of Youtube videos. Import data to R and divide it randomly (50/50)
into training and test sets. #Should reset variables at beginning of run (eases tracability and rerun when solving the problems
with code)
rm(list=ls())
#Remove all plots(eases tracability and rerun when solving the problems with code)
dev.off()
RNGversion("3.5.1")
#----------Assignment 1
Perform principal component analysis using the numeric variables in the training data except of “utime” variable. Do this
analysis with and without scaling of the features. How many components are necessary to explain more than 95% variation of
the data in both cases? Explain why so few components are needed when scaling is not done. (2p)
#----------Part1
data0=read.csv("video.csv")
```

```
data1=data0
data1$codec=c()

n=dim(data1)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data1[id,]
test=data1[-id,]

data11=data1
data11$utime=c()
res=prcomp(data11)
lambda=res$sdev^2
sprintf("%2.3f",cumsum(lambda)/sum(lambda)*100)

res=prcomp(scale(data11))
lambda=res$sdev^2
sprintf("%2.3f",cumsum(lambda)/sum(lambda)*100)
```

Variation without scaling
```
[1] "99.723" "99.935" "99.989" "100.000" "100.000" "100.000" "100.000" "100.000" "10
[10] "100.000" "100.000" "100.000" "100.000" "100.000" "100.000" "100.000" "100.000"
```
Variation witht scaling
```
[1] "35.076" "50.934" "63.495" "70.112" "76.285" "81.969" "87.425" "92.320"
"95.623"
[10] "97.555" "99.139" "99.653" "99.932" "99.968" "100.000" "100.000" "100.000"
```

1 component in unscaled data, 9 components in the scaled data are needed to explain 95% variation. The reason in that the original data is on a very different scale → variation in one feature dominates variation in the other features.

#----------Part2

Write a code that fits a principle component regression ("utime" as response and all scaled numerical variables as features) with MM components to the training data and estimates the training and test errors, do this for all feasible MM values. Plot dependence of the training and test errors on MM and explain this plot in terms of bias-variance tradeoff. (**Hint**: prediction function for principal component regression has some peculiarities, see predict.mvr) **(2p)**

```
library(pls)
trE=numeric(17)
testE=numeric(17)

for (i in 1:17){
  pcrN=pcr(utime~., 17, data=train,  scale=T)
  Yf=predict(pcrN, ncomp=i)
  Yt=predict(pcrN, newdata=test, ncomp=i)
  trE[i]=mean((train$utime-Yf)^2)
  testE[i]=mean((test$utime-Yt)^2)
}
plot(testE, type="l", col="red", ylim=c(100,300), ylab="Error")
points(trE, type="l", col="blue")
```

When the number of components increases, the model becomes more complex and the bias goes down while variance goes up. The optimal model should have the lowest test error, in this case M=14. However, there are much simpler models with similar test errors, for ex. M=8.

#----------Part3

Use PCR model with MM=8 and report a fitted probabilistic model that shows the connection between the target and the principal components. **(1p)**

```
pcrF=pcr(utime~., 8, data=train, validation="none", scale=T)
mean(residuals(pcrF)^2)
Yloadings(pcrF)
```

```
pcrF=pcr(utime~., 8, data=train, validation="none", scale=T)
Yloadings(pcrF)
Loadings:
Comp 1 Comp 2 Comp 3 Comp 4 Comp 5 Comp 6 Comp 7 Comp 8
utime -1.196 1.966 6.048 0.156 0.195 -3.136 4.988
> mean(residuals(pcrF)^2)
```

Equation: $Utime\_scaled s \sim N(-1.196 \cdot Comp1+1.966 \cdot Comp2+6.048 \cdot Comp3+0.156 \cdot Comp4+0.195 \cdot Comp5-3.136 \cdot Comp6+4.988 \cdot Comp8, \sigma 2=156)$

#----------Part4

Use original data to create variable “class” that shows “mpeg” if variable “codec” is equal to “mpeg4”, and “other” for all other values of “codec”. Create a plot of “duration” versus “frames” where cases are colored by “class”. Do you think that the classes are easily separable by a linear decision boundary? **(1p)**

```
data2=data0
data2$class=ifelse(data2$codec=="mpeg4", "mpeg4", "other")
data2$codec=c()
plot(data2$frames,data2$duration, col=as.factor(data2$class), cex=0.5, xlab="frames", ylab="duration")
```

Classes seen to be rather clearly linearly separable (with exception of a few cases near to the origin).

```
#----------------------------------------------------------------Part5
```

Fit a Linear Discriminant Analysis model with “class” as target and “frames” and “duration” as features to the entire dataset (scale features first). Produce the plot showing the classified data and report the training error. Explain why LDA was unable to achieve perfect (or nearly perfect) classification in this case. **(2p)**

```
data2$frames=scale(data2$frames)
data2$duration=scale(data2$duration)
library(MASS)

m3=lda(as.factor(class)~frames+duration, data=data2)
plot(data2$frames,data2$duration, col=predict(m3)$class,  cex=0.5, xlab="frames", ylab="duration")
missclass=function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}
missclass(data2$class, predict(m3, type="class")$class)
```

Misclassification error [1] 0.172

The result of classification is rather bad. It is clear that covariance matrices per class are very different. In addition, class-conditional distributions do not look like multivariate normal

```
#----------------------------------------------------------------Part6
```

Fit a decision tree model with “class” as target and “frames” and “duration” as features to the entire dataset, choose an appropriate tree size by cross-validation. Report the training error. How many leaves are there in the final tree? Explain why such a complicated tree is needed to describe such a simple decision boundary. **(2p)**

```
library(tree)
m4=tree(as.factor(class)~frames+duration, data=data2)
set.seed(12345)
cv.res=cv.tree(m4)
plot(cv.res$size, cv.res$dev, type="b",
    col="red")
print(m4)
plot(m4)
missclass(data2$class, predict(m4, type="class"))
```

Misclassification error

[1] 0.001

According to the cross-validation plot, the optimal tree is the largest one among those that are grown with default settings. The optimal tree among these has 11 leaves.

Such a complicated tree is needed because the optimal decision boundary is linear but not parallel to any of the coordinate axes. Accordingly, decision tree would need to make many splits and produce a stair-kind of decision boundary that would approximate this linear decision boundary.

```
#----------------------------------------------------------------Assignment 2
```

SUPPORT VECTOR MACHINES

You are asked to use the function ksvm from the R package kernlab to learn a support vector machine (SVM) for classifying the *spam* dataset that is included with the package. Consider the radial basis function kernel (also known as Gaussian) with a width of 0.05. For the C parameter, consider values 0.5, 1 and 5. This implies that you have to consider three models.

(2p) Perform model selection, i.e. select the most promising of the three models (use any method of your choice except cross-validation or nested cross-validation).

(1p) Estimate the generalization error of the SVM selected above (use any method of your choice except cross-validation or nested cross-validation).

(1p) Produce the SVM that will be returned to the user, i.e. show the code. **(1p)** What is the purpose of the parameter C ?

```
#----------------------------------------------------------------Part1
# JMP

library(kernlab)
```

```
set.seed(1234567890)
data(spam)
# Model selection
index <- sample(1:4601)
tr <- spam[index[1:2500], ]
va <- spam[index[2501:3501], ]
te <- spam[index[3502:4601], ]

filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=0.5)
mailtype <- predict(filter,va[,-58])
t <- table(mailtype,va[,58])
(t[1,2]+t[2,1])/sum(t)

filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=1)
mailtype <- predict(filter,va[,-58])
t <- table(mailtype,va[,58])
(t[1,2]+t[2,1])/sum(t)

filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=5)
mailtype <- predict(filter,va[,-58])
t <- table(mailtype,va[,58])
(t[1,2]+t[2,1])/sum(t)

# Error estimation
filter <- ksvm(type~.,data=spam[index[1:3501], ],kernel="rbfdot",kpar=list(sigma=0.05),C=1)
mailtype <- predict(filter,te[,-58])
t <- table(mailtype,te[,58])
(t[1,2]+t[2,1])/sum(t)

# Final model
filter <- ksvm(type~.,data=spam,kernel="rbfdot",kpar=list(sigma=0.05),C=1)
# See the attached file for the code. The data is split into three folds: Training, validation, and
testing. The validation data is used to select the best model (C = 0:05 in this case), whose
generalization error is estimated with the help of the test data (note that we use all the data for
learning except the test data). The model returned to the user is the result of using all the data
for learning and C = 0:05. The estimated error of the model returned to the user is precisely
the error we computed on the test data.
#----------------------------------------------------------------------Assignment 3
(3p) Train a neural network (NN) to learn the trigonometric sine function. To do so, sample 50 points uniformly at
random in the interval [0, 10]. Apply the sine function to each point. The resulting pairs are the data available to you. Use
25 of the 50 points for training and the rest for validation. The validation set is used for early stop of the gradient descent.
Consider threshold values i/1000 with i = 1, . . . , 10. Initialize the weights of the neural network to random values in the
interval [−1, 1]. Consider two NN architectures: A single hidden layer of 10 units, and two hidden layers with 3 units each.
Choose the most appropriate NN architecture and threshold value. Motivate your choice. Feel free to reuse the code of the
corresponding lab.
(1p) Estimate the generalization error of the NN selected above (use any method of your choice).
(1p) In the light of the results above, would you say that the more layers the better ? Motivate your answer.
#----------------------------------------------------------------Part1
# JMP
library(neuralnet)

# two layers
set.seed(1234567890)

Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation

# plot(trva)
# plot(tr)
# plot(va)
```

```
restr <- vector(length = 10)
resva <- vector(length = 10)
winit <- runif(22, -1, 1) # Random initializaiton of the weights in the interval [-1, 1]
for(i in 1:10) {
  nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = c(3,3), startweights = winit,
                  threshold = i/1000, lifesign = "full")
  # nn$result.matrix
  aux <- compute(nn, tr[,1])$net.result # Compute predictions for the trainig set and their squared error
  restr[i] <- sum((tr[,2] - aux)**2)/2

  aux <- compute(nn, va[,1])$net.result # The same for the validation set
  resva[i] <- sum((va[,2] - aux)**2)/2
}
plot(restr, type = "o")
plot(resva, type = "o")
restr
resva

# one layer
set.seed(1234567890)
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation

# plot(trva)
# plot(tr)
# plot(va)

restr <- vector(length = 10)
resva <- vector(length = 10)
winit <- runif(41, -1, 1) # Random initializaiton of the weights in the interval [-1, 1]
for(i in 1:10) {
  nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = c(10), startweights = winit,
                  threshold = i/1000, lifesign = "full")
  # nn$result.matrix
  aux <- compute(nn, tr[,1])$net.result # Compute predictions for the trainig set and their squared error
  restr[i] <- sum((tr[,2] - aux)**2)/2
  aux <- compute(nn, va[,1])$net.result # The same for the validation set
  resva[i] <- sum((va[,2] - aux)**2)/2
}
plot(restr, type = "o")
plot(resva, type = "o")
restr
resva

# estimate generalization error for the best run above (one layer with threshold 4/1000)
Var <- runif(50, 0, 10)
te <- data.frame(Var, Sin=sin(Var))
winit <- runif(31, -1, 1)
nn <- neuralnet(formula = Sin ~ Var, data = trva, hidden = 10, startweights = winit,
                threshold = 4/1000, lifesign = "full")
sum((te[,2] - compute(nn, te[,1])$net.result)**2)/2
```

See the file for the code. The best model is that with one layer of 10 neurons and threshold for early stopping equal to 4/1000. The generalization error is estimated by sampling additional test data, since we have access to the true function. Therefore, we conclude that more layers is not always better.

 # ----------EXAM 2017----------#----------

The data file **australian-crabs.csv** contains measurements of various crabs, such as Frontal lobe (FL), Rear width (RW), Carapace Length (CL), Carapace Width (CW), Body depth (BD) as well as indication of which kind of crab it is (Species).

1. Plot the dependence of CW versus BD where the points are colored by Species. Are CW and BD good predictors of the Species? **(1p)**

2. Create a Naïve Bayes classifier model with Species as target and CW and BD as predictors. Present the confusion matrix and comment on the quality of the classification. Based on the assumptions of the Naïve Bayes, explain why this model is not appropriate for these data **(2p)**

3. Fit the logistic regression now with Species as target and CW and BD as predictors and present the equation of the decision boundary. Plot the classified data and the decision boundary and comment on the quality of the classification **(2p)**

4. Scale variables CW and BD and perform principal component analysis with these two variables. Present the proportion of variation explained by PC1 and PC2 and based on results from step 1 explain why the first principal component contains so much variation. Present the equations expressing principal component coordinates through the original coordinates. **(2p)**

5. Create a Naïve Bayes classifier model with Species as target and PC1 and PC2 as predictors. Compute the confusion matrix and explain how much the classification quality has changed and why. **(2p)**

```
#-----------------------------------------------------------------Assignment 1
#-----------------------------------------------------------------Part1

library(e1071)
data = read.csv("australian-crabs.csv")
blue = subset(data, species == "Blue")
orange = subset(data, species == "Orange")
plot(blue$CW, blue$BD, col="blue", xlab = "Caparace Width", ylab = "Body Depth")
points(orange$CW, orange$BD, col="orange")

#-----------------------------------------------------------------Part2

# naive bayes classifier
fit = naiveBayes(species ~ CW+BD, data = data)
pred = predict(fit, newdata = data)
tab = table(data$species, pred)
print(tab) # confusion matrix
print(1-sum(diag(2)*tab)/sum(tab)) # misclassification rate
print(mean(data$species !=pred)) # simpler misclassification rate

#-----------------------------------------------------------------Part3

# logistic regression
lg = glm(species ~ CW+BD, family = binomial, data = data)
lg.pred = predict(lg, newdata = data)
plot(lg.pred)
summary(lg.pred)

#-----------------------------------------------------------------Part4

# principal components
res = prcomp(~CW+BD, data = data, scale = TRUE)
pcdata = data.frame(species = data$species, PC1 = res$x[,1], PC2 = res$x[,2])

#-----------------------------------------------------------------Part5

# improved naive bayes classifier
fit2 = naiveBayes(species ~ .,  data = pcdata)
pred2 = predict(fit2, newdata = pcdata)
tab2 = table(pcdata$species, pred2)
print(tab2) # confusion matrix
print(1-sum(diag(2)*tab2)/sum(tab2)) # misclassification rate
```

#---Assignment 3

In this assignment, you are asked to use the R package kernlab to learn SVMs for classifying the spam dataset that is included with the package. Consider the radial basis function kernel (also known as Gaussian) with a width of 0.05. For the *C* parameter, consider values 1, 10 and 100.

(2p) Estimate the error for the three values of *C*. Use cross-validation with 2 folds. Hint: Use the argument cross=2 when calling the function ksvm. Use the function cross() to print out the error estimate. Use set.seed(1234567890).

#---Part1

```
data(spam)

index <- sample(1:dim(spam)[1])
spamtrain <- spam[index[1:floor(dim(spam)[1]/2)], ]
spamtest <- spam[index[((ceiling(dim(spam)[1]/2)) + 1):dim(spam)[1]], ]

## train a support vector machine
values = c(1,10,100)
for(v in values) {
  set.seed(1234567890)
  filter <- ksvm(type~.,data=spam,kernel="rbfdot", kpar=list(sigma=0.05),C=v,cross=2)
  print(v)
  print(cross(filter))
}
```

 # ----------EXAM 2016----------#----------

```
require(readxl)
library(readxl)
crx = read.csv("crx.csv")

#----------Part1

# Divade data into 3 sets.

n = dim(crx)[1] # n = number of rows
set.seed(12345)
id = sample(1:n, floor(n * 0.8)) # Procent of a set.
train = crx[id, ]
test = crx[-id, ]
tree_fit = tree(Class ~., data = data.frame(train))
predict_train = predict(tree_fit, newdata = train)#, type = "class")
plot(tree_fit)
text(tree_fit)
test_new = test[-2,]
train_new = train[-2,]
new_tree_fit = tree(Class ~., data = train_new)
new_predict_train = predict(tree_fit, newdata = train_new)
plot(new_tree_fit)
text(new_tree_fit)

#----------Part2

cross_validated_fit_tree = cv.tree(tree_fit)
summary(cross_validated_fit_tree)
plot(cross_validated_fit_tree$size,cross_validated_fit_tree$dev, col = "blue", type = "b")
points(log(cross_validated_fit_tree$k),cross_validated_fit_tree$dev, col =" green", type = "b")
plot(y = cross_validated_fit_tree$dev, x = 1:7, type = "b", col = "green", main = "Find best (lowest) score")
prune_tree = prune.tree(tree_fit, best = 5)
plot(prune_tree)
text(prune_tree)
summary(prune_tree)

#----------Part3 ----- Fungerar Ej

library(glmnet)
x_train = model.matrix(~ .-1, train[,-16])
model=c()
model=cv.glmnet(x_train,train[16], alpha=1,family="gaussian")
model$lambda.min
plot(model)
```

```
coef(model, s="lambda.min")
summary(model)
covariates = scale ((crx) [ ,c(2,3, 8,14,15)])
response = scale ( crx [ , 16])
lambdas = seq (-6.5, 0, 0.1)
lambdas = exp(lambdas)
MLasso = glmnet(as.matrix(covariates), response, alpha = 1, lambda = lambdas , family = "gaussian")
plot(MLasso , xvar = "lambda" , label = TRUE)
plot(CV_MLasso , xvar = "lambda" , label = TRUE)
summary(MLasso)
CV_MLasso = cv.glmnet(as.matrix(covariates), response, alpha = 1, lambda = lambdas , family = "gaussian")
plot(log(CV_MLasso$lambda) , CV_MLasso$cvm,  xlab = "lambda")
coef(CV_MLasso, s ="lambda.min")
min_lamda = MLasso$lambda[which.min(MLasso$lambda)]
```

#----------Assignment 2
#---------- Part 2

In the following steps, you are asked to use the R package *kernlab* to learn a SVM for classifying the *spam* dataset that is included with the package. For the *C* parameter, consider values 1 and 5. Consider the radial basis function kernel (also known as Gaussian) and the linear kernel. For the former, consider a width of 0.01 and 0.05. This implies that you have to select among six models.

```
# SVM
data(spam)
index <- sample(1:dim(spam)[1])
spamtrain <- spam[index[1:floor(dim(spam)[1]/2)], ]
spamtest <- spam[index[((ceiling(dim(spam)[1]/2)) + 1):dim(spam)[1]], ]

m1 <- ksvm(type~.,data=spamtrain,kernel="vanilladot",C=1,cross=2)
m2 <- ksvm(type~.,data=spamtrain,kernel="rbfdot",kpar=list(sigma=0.01),C=1,cross=2)
m3 <- ksvm(type~.,data=spamtrain,kernel="rbfdot",kpar=list(sigma=0.05),C=1,cross=2)
m4 <- ksvm(type~.,data=spamtrain,kernel="vanilladot",C=5,cross=2)
m5 <- ksvm(type~.,data=spamtrain,kernel="rbfdot",kpar=list(sigma=0.01),C=5,cross=2)
m6 <- ksvm(type~.,data=spamtrain,kernel="rbfdot",kpar=list(sigma=0.05),C=5,cross=2)

# takes a function, predicts classes and outputs misclassification rate
test <- function(m) {
  p <- predict(m,spamtest[,-58])
  mean(p != spamtest[,58]) # misclassification rate

}
# comparing misclass-rates of the models
missclass_rates = c(test(m1), test(m2), test(m3), test(m4), test(m5), test(m6))
which.min(missclass_rates) # m5 won
```

#---------- Neural network

Implement the backpropagation algorithm for fitting the parameters of a NN for regression. The NN has one input unit, 10 hidden units, and one output unit. Use the *tanh* activation function.
Recall that you have an example on Bishop's book as well as on the course slides. Feel free to use stochastic or batch gradient descent. Please use only basic R functions in your solution, e.g. *sum*, *tanh*. **(4p)**

5. Run your code for 5000 iterations (if time permits) on the training data *tr* below. A learning rate in the interval [1252,125] should work fine. Plot the error on *tr* as well as on the validation data *va*, as a function of the number of iterations. **(1p)**

```
# neural network
set.seed(1234567890)
Var = runif(50,0,10)
trva = data.frame(Var, Sin = sin(Var))
tr = trva[1:25,] # training data
va = trva[26:50,] # validation data
```

```
w_j = runif(10, -1, 1)
b_j = runif(10, -1, 1)
w_k = runif(10, -1, 1)
b_k = runif(1, -1, 1)

l_rate = 1/nrow(tr)^2
n_ite = 5000
error = rep(0, n_ite)
error_va = rep(0, n_ite)

for(i in 1:n_ite) {
  for(n in 1:nrow(tr)) {
    z_j = tanh(w_j * tr[n,]$Var + b_j)
    y_k = sum(w_k * z_j) + b_k
    error[i] = error[i] + (y_k + tr[n,]$Sin)^2
  }

  for(n in 1:nrow(va)) {
    z_j = tanh(w_j * va[n,]$Var + b_j)
    y_k = sum(w_k * z_j) + b_k
    error_va[i] = error_va[i] + (y_k + va[n,]$Sin)^2
  }

    cat("i: ", i , ", error: ", error[i]/2, ", error_va: ", error_va[i]/2, "\n")
  flush.console()

    for(n in 1:nrow(tr)) {
    z_j = tanh(w_j * tr[n,]$Var + b_j)
    y_k = sum(w_k * z_j) + b_k
    d_k = y_k - tr[n,]$Sin
    d_j = (1 - z_j^2) * w_k * d_k
    partial_w_k = d_k * z_j
    partial_b_k = d_k
    partial_w_j = d_j * tr[n,]$Var
    partial_b_j = d_j
    w_k = w_k - l_rate * partial_w_k
    b_k = b_k - l_rate * partial_b_k
    w_j = w_j - l_rate * partial_w_j
    b_j = b_j - l_rate * partial_b_j
  }
}

w_j
b_j
w_k
b_k

plot(error/2, ylim = c(0, 5))
points(error_va/2, col = "red")

# prediction on training data
pred = matrix(nrow = nrow(tr), ncol = 2)

for(n in 1:nrow(tr)) {
  z_j = tanh(w_j * tr[n,]$Var + b_j)
  y_k = sum(w_k * z_j) + b_k
  pred[n,] = c(tr[n,]$Var, y_k)

}

plot(pred)
points(tr, col = "red")
```

```
# prediction on validation data
pred = matrix(nrow = nrow(va), ncol = 2)
for(n in 1:nrow(va)) {
  z_j = tanh(w_j * va[n,]$Var + b_j)
  y_k = sum(w_k * z_j) + b_k
  pred[n,] = c(va[n,]$Var, y_k)
}
plot(pred)
points(va, col = "red")
```

----------NOTES----------#

Generalization error

The concepts of generalization error and overfitting are closely related. Overfitting occurs when the learned function fs becomes sensitive to the noise in the sample. As a result, the function will perform well on the training set but not perform well on other data from the joint probability distribution of x and y. Thus, the more overfitting occurs, the larger the generalization error.

The amount of overfitting can be tested using cross-validation methods, that split the sample into simulated training samples and testing samples. The model is then trained on a training sample and evaluated on the testing sample. The testing sample is previously unseen by the algorithm and so represents a random sample from the joint probability distribution of x and y. This test sample allows us to approximate the expected error and as a result approximate a particular form of the generalization error. Many algorithms exist to prevent overfitting. The minimization algorithm can penalize more complex functions (known as Tikhonov regularization), or the hypothesis space can be constrained, either explicitly in the form of the functions or by adding constraints to the minimization function (Ivanov regularization).

The approach to finding a function that does not overfit is at odds with the goal of finding a function that is sufficiently complex to capture the particular characteristics of the data. This is known as the bias–variance tradeoff. Keeping a function simple to avoid overfitting may introduce a bias in the resulting predictions, while allowing it to be more complex leads to overfitting and a higher variance in the predictions. It is impossible to minimize both simultaneously.

Definition

Firstly, let's define "generalization error". *In supervised learning applications in machine learning and statistical learning theory, generalization error (also known as the out-of-sample error) is a measure of how accurately an algorithm is able to predict outcome values for previously unseen data. wikipedia* Notice that the gap between predictions and observed data is induced by **model inaccuracy**, **sampling error**, and **noise**. Some of the errors are reducible but some are not. Choosing the right algorithm and tuning parameters could improve model accuracy, but we will never be able to make our predictions 100% accurate.

Mathematical Notations

D: the training dataset
x: our sample
y: the **real** values of the outcome variable
y_D: the **observed** values of the outcome variable **in dataset D**
$f(x; D)$: the **fitted** values of the outcome variable, i.e. the output of our model when input=x, and the model is learned from dataset D
E_D: the expectation on dataset D
Error(f;D): the generalization error of model f trained on dataset D

Using regression as an example, we have

Generalization error

$$Error(f; D) = E[(f(x; D) - y_D)^2]$$

where

$$\overline{f(x)} = E_D[f(x; D)]$$

bias could be denoted as

$$bias^2(x) = (\overline{f(x)} - y_D)^2$$

Variance be

$$var(x) = E_D[(f(x; D) - \overline{f(x)})^2]$$

And **noise**

$$\epsilon^2 = E_D[(y_D - y)^2]$$

For noise, we have "zero assumption": the expectation of noise is zero.

$$E_D[y_D - y] = 0$$

Bias-variance decomposition

An important way to understand generalization error is **bias-variance decomposition**.

Intuitively speaking, **bias** is the **error rate** in the world of big data. A model has a high bias when, for example, it fails to capture meaningful patterns in the data. Bias is measured by the differences between the *expected predicted values* and the *observed values*, in the dataset D when the prediction variables are at the level of x (X=x). In contrast with bias, **variance** is an algorithm's **flexibility** to learn patterns in the observed data. **Variance** is the amount that an algorithm will change if a **different dataset** is used. A model is of high variance when, for instance, it tries too hard that it not only captures the pattern of meaningful features but also that the meaningless error (**overfitting**).

Interpretation

Bias measures the deviation between the expected output of our model and the real values, so it indicates the **fit of our model**. Variance measures the amount that the outputs of our model will change if a different dataset is used. It is the impacts of using different datasets. Noise is the irreducible error, the **lowest bound of generalization error** for the current task that any model will not be able to get rid of, indicating the difficulty of this task. These 3 components above determine the model's ability to react to new unseen data rather than just the data that it was trained on.

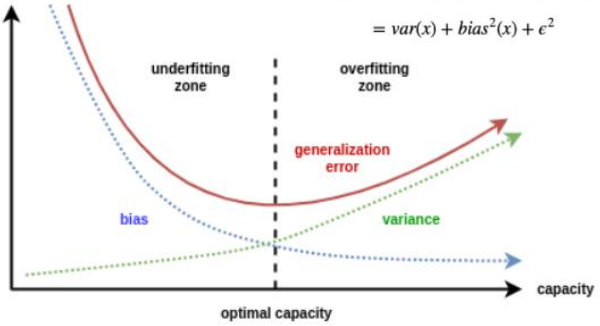
Now we could decompose generalization error

$$E_D[(f(x;D) - y_D)^2] = E_D[(f(x;D) - \overline{f(x)} + \overline{f(x)} - y_D)^2]$$
$$= E_D[(f(x;D) - \overline{f(x)})^2] + E_D[(\overline{f(x)} - y_D)^2] + 2E_D[(f(x;D) - \overline{f(x)})(\overline{f(x)} - y_D)]$$
$$= E_D[(f(x;D) - \overline{f(x)})^2] + E_D[(\overline{f(x)} - y_D)^2]$$
$$= var(x) + E_D[(\overline{f(x)} - y + y - y_D)^2]$$
$$= var(x) + E_D[(\overline{f(x)} - y)^2] + E_D[(y - y_D)^2] + 2E_D[(\overline{f(x)} - y)(y - y_D)]$$
$$= var(x) + E_D[(\overline{f(x)} - y)^2] + E_D[(y - y_D)^2]$$
$$= var(x) + (\overline{f(x)} - y)^2 + E_D[(y - y_D)^2]$$
$$= var(x) + bias^2(x) + \epsilon^2$$

Bias-Variance Tradeoff

Bias-Variance Tradeoff as a Function of Model Capacity

Generalization error could be measured by MSE. As the model capacity increases, the bias decreases as the model fits the training datasets better. However, the variance increases, as your model become sophisticated to fit patterns of the current dataset, changing datasets (even come from the same distribution) would be impactful. As scientist, our challenge lies in finding the optimal capacity where both bias and variance are low.



SVM

In machine learning, **support-vector machines** (**SVMs**, also **support-vector networks**[1]) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. When data are unlabelled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups. The **support-vector clustering**[2] algorithm, created by Hava Siegelmann and Vladimir Vapnik, applies the statistics of support vectors, developed in the support vector machines algorithm, to categorize unlabeled data, and is one of the most widely used clustering algorithms in industrial applications

The effectiveness of SVM depends on the selection of kernel, the kernel's parameters, and soft margin parameter C. A common choice is a Gaussian kernel, which has a single parameter. Potential drawbacks of the SVM include the following aspects:

- Requires full labeling of input data
  Uncalibrated class membership probabilities—SVM stems from Vapnik's theory which avoids estimating probabilities on finite data
  The SVM is only directly applicable for two-class tasks. Therefore, algorithms that reduce the multi-class task to several binary problems have to be applied; see the multi-class SVM section.
  Parameters of a solved model are difficult to interpret.

The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, you should get misclassified examples, often even if your training data is linearly separable.

In a SVM you are searching for two things: a hyperplane with the largest minimum margin, and a hyperplane that correctly separates as many instances as possible. The problem is that you will not always be able to get both things. The c parameter determines how great your desire is for the latter

??Ksvm

the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes the inner product in feature space between two vector arguments (see `kernels`).
kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings:

`rbfdot` Radial Basis kernel "Gaussian"
`polydot` Polynomial kernel
`vanilladot` Linear kernel
`tanhdot` Hyperbolic tangent kernel
`laplacedot` Laplacian kernel
`besseldot` Bessel kernel
`anovadot` ANOVA RBF kernel

`splinedot` Spline kernel
`stringdot` String kernel

k-Fold Cross-Validation

The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.
The general procedure is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
    1. Take the group as a hold out or test data set
    2. Take the remaining groups as a training data set
    3. Fit a model on the training set and evaluate it on the test set
    4. Retain the evaluation score and discard the model
4. Summarize the skill of the model using the sample of model evaluation scores

Nested cross- validation

A chief confusion about CV is not understanding the need for multiple uses of it, within layers. For instance, one can use cross validation within the model selection process and a different cross validation loop to actually select the winning model. The CV equivalent to the training/validation/test split – that is, for model selection — requires that we run an inner CV that's equivalent to the train/validation partition within an outer CV that's equivalent to the validation/test partition. This is called Nested CV.

Poisson

Density, distribution function, quantile function and random generation for the Poisson distribution with parameter `lambda`.

Usage

```
dpois(x, lambda, log = FALSE)
ppois(q, lambda, lower.tail = TRUE, log.p = FALSE)
qpois(p, lambda, lower.tail = TRUE, log.p = FALSE)
rpois(n, lambda)
```

`x` vector of (non-negative integer) quantiles.
`q` vector of quantiles.
`p` vector of probabilities.
`n` number of random values to return.

Examples

```
require(graphics)
-log(dpois(0:7, lambda = 1) * gamma(1+ 0:7)) # == 1
Ni <- rpois(50, lambda = 4); table(factor(Ni, 0:max(Ni)))
1 - ppois(10*(15:25), lambda = 100)  # becomes 0 (cancellation)
    ppois(10*(15:25), lambda = 100, lower.tail = FALSE)  # no cancellation

par(mfrow = c(2, 1))
x <- seq(-0.01, 5, 0.01)
plot(x, ppois(x, 1), type = "s", ylab = "F(x)", main = "Poisson(1) CDF")
plot(x, pbinom(x, 100, 0.01), type = "s", ylab = "F(x)",
     main = "Binomial(100, 0.01) CDF")
## The (limit) case  lambda = 0 :
stopifnot(identical(dpois(0,0), 1), identical(ppois(0,0), 1),identical(qpois(1,0), 0))
```

```
predicted <- predict(fit, newdata = testing, type = "vector")
mean_square_error <- mean((predicted - testing$Al)^2)

# Part 3: PLS Regression Model

full_training_indices <- c(training_indices, validation_indices)
full_training <- glass[full_training_indices,] # Using C-V here.
fit <- plsr(Al ~ ., data = full_training, validation = "CV")

summary(fit) # Gives us most of the relevant information.

png("validation_plot.png")
validationplot(fit)
dev.off()

png("loading_plot.png")
loadingplot(loadings(fit), label = "names")
dev.off()

predicted <- predict(fit, newdata = testing)
mean_square_error <- mean((predicted - testing$Al)^2)
```

```
library(MASS)
library(tree)
library(pls)
```

Assignment 1

```
data = read.csv2("glass.csv",stringsAsFactors = TRUE)
n = nrow(data)
set.seed(12345)
data = data[sample(1:n,n),]
training = data[0:floor(n*0.5),]
validation = data[floor(n*0.5):floor(n*0.75),]
test = data[floor(n*0.75):n,]
```

Generate tree models for the testing and validation data and show the error plots

```
n = nrow(training)
control = tree.control(n, minsize=1)
fit = tree(Al ~ ., data = training, control = control)
```

```
fit.cv = cv.tree(fit)
size = summary(fit)$size[1]
print(size)
results = matrix(0,size,2)
variances = matrix(0,size,2)

for(i in 2:size){
  results[i,1] = mean((predict(prune.tree(fit,best = i), newdata=validation) -
validation$Al)^2)
  results[i,2] = mean((predict(prune.tree(fit,best = i), newdata=training) -
training$Al)^2)
  variances[i,1] = var(predict(prune.tree(fit,best = i), newdata=validation))
  variances[i,2] = var(predict(prune.tree(fit,best = i), newdata=training))
}

plot(1:size, results[,1], type = "l")
plot(1:size,variances[,1],col="Red")
plot(1:size, results[,2], type = "l")
plot(1:size,variances[,2],col="Red")
```

Best tree size

```
best_size = fit.cv$size[which.min(fit.cv$dev)]
optimal_tree = prune.tree(fit,best=best_size)
print(best_size)
summary(optimal_tree)
```

test error

```
result = mean((predict(optimal_tree, newdata = test) - test$Al)^2)
print(result)
```

PLS regression model

```
fit = plsr(Al ~ ., data = training, validation = "CV")
summary(fit)
```

We can observe how 3 components are enough to explain 90% of the variance of the data and 6 for the target. According to the CV 7 is the optimal number of varaibles to consider.

```
print(names(data)[which.max((fit$coefficients)^2)])
validationplot(fit,val.type = "MS")
```

shows that the most significant component was the "Channel29" component. The function from the components to the result can be seen as the following coefficients (Al.6comps)

```
fit$coefficients
```

and the predicted error of the PLS model is

```
print( mean((predict(fit,newdata=test) - test$Al)^2))
```

Comparing the reggresion tree model and the PLS we can see how the PLS model had a lower MSE.

Assignment 2

Plot the data in the coordinates hp vs qsec

```
data = mtcars
plot(data$hp, data$qsec, col=data$am+1)
```

The assumption of LDA is that the covaranance of all the calses should be equal (in practice simmilar) to each other. Looking at the plot above we can see how this asusmption seems to be fullfiled. The data will not be classified perfeclty, even if class priors are choosen perfeclt because no matter where you put the class sepperation, som elements will be missclassified.

LDA with equal priors (0.5,0.5)

```
fit_equal = lda(am ~ qsec + hp, data = data, prior = c(0.5, 0.5))
plot(data$hp, data$qsec, col=predict(fit_equal)$class)
```

LDA with proportial priiors

```
fit_prop = lda(am ~ qsec + hp, data = data)
plot(data$hp, data$qsec, col=predict(fit_prop)$class)
```

looking at the missclassfication rate of both proportial priors and equal priors

```
table(predict(fit_equal)$class, data$am)
sum(predict(fit_equal)$class != data$am)/nrow(data)

table(predict(fit_prop)$class, data$am)
sum(predict(fit_prop)$class != data$am)/nrow(data)
```

we can observe how equal priors had a lower missclassfication rate compared to proportional priors.

Looking at the models we can se that

```
fit_equal
fit_prop
```

the slope hasn't changed anything between the two models but the intercept has.

IMplement kernel density estmiation with Epanechnikov kernel that uses matricies X, XTest and a sclara (\lambda) to estimate thedensity from X and predict it as XTest.

```
epanechnikov = function(x) {
  x_nor = norm(x,"F")
  if(x_nor^2 >= 0){
    return(1 - x_nor^2)
  }
  return(0)
}

kernel = function(X,XTest,lambda){
 n = nrow(X)
 return(apply(XTest,1, function(x){
   value
   for(i in 1:n){
```

```
    value = 1/(n*lambda) * sum(epanechnikov(X[i,1] - x))
  }
  return(value)
 }))
}
```

Assignemnt 3

```
data = read.csv2("wine.csv", sep = ",")
data$class[which(data$class == 2)] = -1
for(i in 1:ncol(data)){data[,i] = as.numeric(data[,i])}

set.seed(12345)
samples = sample(1:nrow(data),floor(nrow(data)*0.7))
training = data[samples,]
test = data[-samples,]


library(neuralnet)


set.seed(12345)
f <- as.formula(paste("class ~", paste(training[!training %in% "class"], collapse
= " + ")))
nn <- neuralnet(f, training, hidden = 0, act.fct = "tanh")
print(colMeans(nn$generalized.weights[[1]]))
```

We can observe how the feature "proline" has a significant higher weight compared to all other features, this would indicate that it's the most important while the feaure "alcohol" is the least important.

```
pred_train = sign(compute(nn, training)$net.result)
pred_test =  sign(compute(nn, test)$net.result)
miss_rate_train = sum(pred_train != training$class) / nrow(training)
miss_rate_test = sum(pred_test != test$class) / nrow(test)
print(miss_rate_test)
```

```
print(miss_rate_train)


set.seed(12345)
nn <- neuralnet(f, training, hidden = 1, act.fct = "tanh")
plot(nn)
print(colMeans(nn$generalized.weights[[1]]))


pred_train = sign(compute(nn, training)$net.result)
pred_test =  sign(compute(nn, test)$net.result)
miss_rate_train = sum(pred_train != training$class) / nrow(training)
miss_rate_test = sum(pred_test != test$class) / nrow(test)
print(miss_rate_test)
print(miss_rate_train)
```

Base R
Cheat Sheet

Getting Help

Accessing the help files

?mean
Get help of a particular function.

help.search('weighted mean')
Search the help files for a word or phrase.

help(package = 'dplyr')
Find help for a package.

More about an object

str(iris)
Get a summary of an object's structure.

class(iris)
Find the class an object belongs to.

Using Libraries

install.packages('dplyr')
Download and install a package from CRAN.

library(dplyr)
Load the package into the session, making all its functions available to use.

dplyr::select
Use a particular function from a package.

data(iris)
Load a built-in dataset into the environment.

Working Directory

getwd()
Find the current working directory (where inputs are found and outputs are sent).

setwd('C://file/path')
Change the current working directory.

Use projects in RStudio to set the working directory to the folder you are working in.

Vectors

Creating Vectors

`c(2, 4, 6)`	`2 4 6`	Join elements into a vector
`2:6`	`2 3 4 5 6`	An integer sequence
`seq(2, 3, by=0.5)`	`2.0 2.5 3.0`	A complex sequence
`rep(1:2, times=3)`	`1 2 1 2 1 2`	Repeat a vector
`rep(1:2, each=3)`	`1 1 1 2 2 2`	Repeat elements of a vector

Vector Functions

sort(x)
Return x sorted.

rev(x)
Return x reversed.

table(x)
See counts of values.

unique(x)
See unique values.

Selecting Vector Elements

By Position

`x[4]`	The fourth element.
`x[-4]`	All but the fourth.
`x[2:4]`	Elements two to four.
`x[-(2:4)]`	All elements except two to four.
`x[c(1, 5)]`	Elements one and five.

By Value

`x[x == 10]`	Elements which are equal to 10.
`x[x < 0]`	All elements less than zero.
`x[x %in% c(1, 2, 5)]`	Elements in the set 1, 2, 5.

Named Vectors

`x['apple']`	Element with name 'apple'.

Programming

For Loop

```
for (variable in sequence){
   Do something
}
```

Example

```
for (i in 1:4){
   j <- i + 10
   print(j)
}
```

While Loop

```
while (condition){
   Do something
}
```

Example

```
while (i < 5){
   print(i)
   i <- i + 1
}
```

If Statements

```
if (condition){
   Do something
} else {
   Do something different
}
```

Example

```
if (i > 3){
   print('Yes')
} else {
   print('No')
}
```

Functions

```
function_name <- function(var){
   Do something
   return(new_variable)
}
```

Example

```
square <- function(x){
   squared <- x*x
   return(squared)
}
```

Reading and Writing Data

Input	Ouput	Description
`df <- read.table('file.txt')`	`write.table(df, 'file.txt')`	Read and write a delimited text file.
`df <- read.csv('file.csv')`	`write.csv(df, 'file.csv')`	Read and write a comma separated value file. This is a special case of read.table/ write.table.
`load('file.RData')`	`save(df, file = 'file.Rdata')`	Read and write an R data file, a file type special for R.

Conditions

`a == b`	Are equal	`a > b`	Greater than	`a >= b`	Greater than or equal to	`is.na(a)`	Is missing
`a != b`	Not equal	`a < b`	Less than	`a <= b`	Less than or equal to	`is.null(a)`	Is null

Types

Converting between common data types in R. Can always go from a higher value in the table to a lower value.

`as.logical`	`TRUE, FALSE, TRUE`	Boolean values (TRUE or FALSE).
`as.numeric`	`1, 0, 1`	Integers or floating point numbers.
`as.character`	`'1', '0', '1'`	Character strings. Generally preferred to factors.
`as.factor`	`'1', '0', '1', levels: '1', '0'`	Character strings with preset levels. Needed for some statistical models.

Maths Functions

`log(x)`	Natural log.	`sum(x)`	Sum.
`exp(x)`	Exponential.	`mean(x)`	Mean.
`max(x)`	Largest element.	`median(x)`	Median.
`min(x)`	Smallest element.	`quantile(x)`	Percentage quantiles.
`round(x, n)`	Round to n decimal places.	`rank(x)`	Rank of elements.
`signif(x, n)`	Round to n significant figures.	`var(x)`	The variance.
`cor(x, y)`	Correlation.	`sd(x)`	The standard deviation.

Variable Assignment

```
> a <- 'apple'
> a
[1] 'apple'
```

The Environment

`ls()`	List all variables in the environment.
`rm(x)`	Remove x from the environment.
`rm(list = ls())`	Remove all variables from the environment.

You can use the environment panel in RStudio to browse variables in your environment.

Matrixes

`m <- matrix(x, nrow = 3, ncol = 3)`
Create a matrix from x.

`m[2, ]` - Select a row

`m[ , 1]` - Select a column

`m[2, 3]` - Select an element

`t(m)`
Transpose

`m %*% n`
Matrix Multiplication

`solve(m, n)`
Find x in: m * x = n

Lists

`l <- list(x = 1:5, y = c('a', 'b'))`
A list is collection of elements which can be of different types.

`l[[2]]`	`l[1]`	`l$x`	`l['y']`
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

Also see the **dplyr** library.

Data Frames

`df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))`
A special case of a list where all elements are the same length.

x	y
1	a
2	b
3	c

List subsetting

`df$x`

`df[[2]]`

Understanding a data frame

`View(df)`	See the full data frame.
`head(df)`	See the first 6 rows.

Matrix subsetting

`df[ , 2]`

`df[2, ]`

`df[2, 2]`

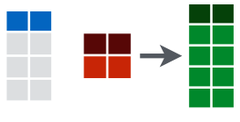
`nrow(df)`
Number of rows.

`ncol(df)`
Number of columns.

`dim(df)`
Number of columns and rows.

`cbind` - Bind columns.

`rbind` - Bind rows.



Strings

Also see the **stringr** library.

`paste(x, y, sep = ' ')`	Join multiple vectors together.
`paste(x, collapse = ' ')`	Join elements of a vector together.
`grep(pattern, x)`	Find regular expression matches in x.
`gsub(pattern, replace, x)`	Replace matches in x with a string.
`toupper(x)`	Convert to uppercase.
`tolower(x)`	Convert to lowercase.
`nchar(x)`	Number of characters in a string.

Factors

`factor(x)`
Turn a vector into a factor. Can set the levels of the factor and the order.

`cut(x, breaks = 4)`
Turn a numeric vector into a factor but 'cutting' into sections.

Statistics

`lm(x ~ y, data=df)`
Linear model.

`glm(x ~ y, data=df)`
Generalised linear model.

`summary`
Get more detailed information out a model.

`t.test(x, y)`
Preform a t-test for difference between means.

`pairwise.t.test`
Preform a t-test for paired data.

`prop.test`
Test for a difference between proportions.

`aov`
Analysis of variance.

Distributions

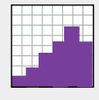
	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	`rnorm`	`dnorm`	`pnorm`	`qnorm`
Poison	`rpois`	`dpois`	`ppois`	`qpois`
Binomial	`rbinom`	`dbinom`	`pbinom`	`qbinom`
Uniform	`runif`	`dunif`	`punif`	`qunif`

Plotting

Also see the **ggplot2** library.

`plot(x)`
Values of x in order.

`plot(x, y)`
Values of x against y.



`hist(x)`
Histogram of x.

Dates

See the **lubridate** library.

The Ultimate R Cheat Sheet – Data Management (Version 4)

Google "R Cheat Sheet" for alternatives. The best cheat sheets are those that you make yourself! Arbitrary variable and table names that are not part of the R function itself are highlighted in bold.

Import, export, and quick checks

- **dat1**=read.csv(**"name.csv"**) to import a standard CSV file (first row are variable names).
- attach(**dat1**) to set a table as default to look for variables. Use detach() to release.
- **dat1**=read.delim(**"name.txt"**) to import a standard tab-delimited file.
- **dat1**=read.fwf(**"name.prn"**, widths=c(**8,8,8**)) fixed width (3 variables, 8 characters wide).
- ?read.table to find out more options for importing non-standard data files.
- **dat1**=read.dbf(**"name.dbf"**) requires installation of the foreign package to import DBF files.
- head(**dat1**) to check the first few rows and variable names of the data table you imported.
- names(**dat1**) to list variable names in quotation marks (handy for copy and paste to code).
- data.frame(names(**dat1**)) gives you a list of your variables with the column number indicated, which can be handy for sub-setting a data table (see next page)
- nrow(**dat1**) and ncol(**dat1**) returns the number of rows and columns of a data table.
- length(**dat1**$**VAR1**[!is.na(**dat1**$**VAR1**)] returns a count of non-missing values in a variable.
- str(**dat1**) to check variable types, which is useful to see if the import executed correctly.
- write.csv(**results, "myresults.csv",** na="", row.names=F) to export data. Without the option statements, missing values will be represented by NA and row numbers will be written out.

Data types and basic data table manipulations

- There are three important variable types: numeric, character and factor (a double variable with a numeric and character value). You can query or assign types: is.factor() or as.factor().
- If you import a data table, variables that contain one or more character entries will be set to factor. You can force them to numeric with this: as.numeric(as.character(**dat1**$**VAR1**))
- After subsetting or modification, you might want to refresh factor levels with droplevels(**dat1**)
- Data tables can be set as.data.frame(), as.matrix(), as.distance()
- names(**dat1**)=c(**"ID", "X", "Y", "Z"**) renames variables. Note that the length of the vector must match the number of variable you have (four in this case).
- row.names(**dat1**)=**dat1**$**ID**. assigns an ID field to row names. Note that the default row names are consecutive numbers. In order for this to work, each value in the ID field must be unique.
- To generate unique and descriptive row names that may serve as IDs, you can combine two or more variables: row.names(**dat1**)=paste(**dat1**$**SITE, dat1**$**PLOT,** sep="-")
- If you only have numerical values in your data table, you can transpose it (switch rows and columns): **dat1_t**=t(**dat1**). Row names become variables, so run the row.names() function above first.
- **dat1**[order(**X**),] orders rows by variable X. dat[order(**X,Y**),] orders rows by variable X, then variable Y. **dat1**[order(**X,-Y**),]. Orders rows by variable X, then descending by variable Y.
- fix(**dat1**) to open the entire data table as a spreadsheet and edit cells with a double-click.

Creating systematic data and data tables

- c(1:10) is a generic concatenate function to create a vector, here numbers from 1 to 10.
- seq(0, 100, 10) generates a sequence from 0 to 100 in steps of 10.
- rep(5,10) replicates 5, 10 times. rep(c(1,2,3),2) gives 1 2 3 1 2 3. rep(c(1,2,3), each=2) gives 1 1 2 2 3 3. This can be useful to create data entry sheets for experimental designs.
- data.frame(**VAR1**=c(1:10), **VAR2**=seq(10, 100, 10), **VAR3**=rep(c(**"this", "that"**),5)) creates a data frame from a number of vectors.
- expand.grid(**SITE**=c(**"A","B"**),**TREAT**=c(**"low","med","high"**), REP=c(1:5)) is an elegant method to create systematic data tables.

Creating random data and random sampling

- `rnorm(10)` takes 10 random samples from a normal distribution with a mean of zero and a standard deviation of 1
- `runif(10)` takes 10 random samples from a uniform distribution between zero and one.
- `round(rnorm(10)*3+15))` takes 10 random samples from a normal distribution with a mean of 15 and a standard deviation of 3, and with decimals removed by the rounding function.
- `round(runif(10)*5+15)` returns random integers between 15 and 20, uniformly distributed.
- `sample(c("A","B","C"), 10, replace=TRUE)` returns a random sample from any custom vector or variable with replacement.
- **sample1**=**dat1**`[sample(1:nrow(`**dat1**`), 50, replace=FALSE),]` takes 50 random rows from dat1 (without duplicate sampling). This can be handy for bootstrapping or to run quick test analyses on subsets of very large datasets.

Sub-setting data tables, conditional subsets

- **dat1**`[1:10, 1:5]` returns the first 10 rows and the first 5 columns of table dat1.
- **dat2**=**dat1**`[50:70,]` returns a subset of rows 50 to 70.
- **cleandata**=**dat1**`[-c(2,7,15),]` removes rows 2, 7 and 15.
- **selectvars**=**dat1**`[,c("ID","YIELD")]` sub-sets the variables ID and YIELD
- **selectrows**=**dat1**`[`**dat1**`$`**VAR1**`=="Site 1",]` sub-sets entries that were measured at Site 1. Possible conditional operators are == equal, != non-equal, > larger, < smaller, >= larger or equal, <= smaller or equal, & AND, | OR, ! NOT, () brackets to order complex conditional statements.
- **selecttreats**=**dat1**`[`**dat1**`$`**TREAT** `%in% c("CTRL", "N", "P", "K"),]` can replace multiple conditional == statements linked together by OR.

Transforming variables in data tables, conditional transformations

- **dat2**=`transform(`**dat1, VAR1**=**VAR1**`*0.4)`. Multiplies VAR1 by 0.4
- **dat2**=`transform(`**dat1, VAR2**=**VAR1**`*2)`. Creates variable VAR2 that is twice the value of VAR1
- **dat2**=`transform(`**dat1, VAR1**=`ifelse(`**VAR3**`=="Site 1", VAR1`*`0.4, VAR1`)) Multiplies VAR1 by 0.1 for entries measured at Site 1. For other sites the value stays the same. The general format is ifelse(condition, value if true, value if false).
- The `vegan` package offers many useful standard transformations for variables or an entire data table: **dat2**=`decostand(`**dat1,**`"standardize")` Check out `?decostand` to see all transformations.

Merging data tables

- **dat3**=`merge(`**dat1,dat2,**`by="ID")` merge two tables by ID field.
- **dat3**=`merge(`**dat1,dat2,**`by.x="ID",by.y="STN")` merge by an ID field that is differently named in the two datasets.
- **dat3**=`merge(`**dat1,dat2,**`by=c(`**"LAT","LONG"**`))` merge by multiple ID fields.
- **dat3**=`merge(`**dat1,dat2,**`by.x=`**"ID"**`,by.y=`**"ID"**`,all.x=T,all.y=F)` left merge; a`ll.x=F, all.y=T` right merge; `all.x=T,all.y=T` keep all rows; `all.x=F,all.y=F` keep matching rows.
- `cbind(`**dat1,dat2**`)` On very rare occasions, you merge data without a criteria (ID). This is generally dangerous, because the commands will slap the two tables together without checking the order!
- **dat3**=`rbind(`**dat1,dat2**`)` adding rows of two data tables. The variables have to match exactly and you will get error messages if they don't match. So, unlike cbind(), rbind() is generally safe to use.
- **dat3**=`rbind.fill(`**dat1,dat2**`)` will force non-matching datasets together, filling missing values and executing variable type conversions where appropriate. Requires the `reshape` package.

Summary statistics for variables and tables

- `mean() weighted.mean(,) median() max() min() range() which.max() which.min() var() sd() quantile() quantile(,c(0.025,0.05,0.95,0.975)) rank(x)` some descriptive statistical functions for variables or vectors. For all functions, and

important option is `na.rm=T`, which means that missing values are ignored in the calculations, e.g. `mean(VAR1, na.rm=T)`. Without that option, the function returns missing values as a result of missing values in the input.

- `rowSums()`, `colSums()`, `rowMeans()` or `colMeans()` applies functions to rows or columns of a table. For example, `rowsum(dat1[,10:15])` will return the row-sums of the variables in columns 10 to 15. Don't forget `na.rm=T`.
- `apply(dat,1,max)` apply any function (e.g. **max**), to either rows (**1**) or columns (**2**) of a table (**dat**).

Pivot table functionality

- The functions aggregate and ddply can be used to summarize data similarly to working with Excel pivot tables. Aggregate has simpler syntax if you have many variables that you want to summarize in the same way; ddply is better if you have few variables but want several custom summary statistics.
- `aggregate(dat1[,4:9], by=list(TREAT1=dat1$TREAT1, TREAT2=dat2$TREAT2), mean)` calculates the means of a number of numerical variables in columns 4 to 9 for two treatments.
- `ddply(dat1,.(TREAT), summarise, mVAR1=mean(VAR1))` returns means of VAR1 by a class variables TREAT. This requires installation of the library `plyr`.
- `ddply(dat1,.(TREAT1, TREAT2), summarise, cVAR1=length(VAR1[!is.na(VAR1)])` returns a count of non-missing values in variable VAR1 for each combination of two class variables.
- `ddply(dat1,.(TREAT1, TREAT2), summarise, mVAR1=mean(VAR1, na.rm=T), seVAR1=sd(VAR1, na.rm=T)/sqrt(length(VAR1[!is.na(VAR1)]))).` A clever piece of code to calculate means and standard errors, with missing values being properly handled.

Long-to-wide and wide-to-long data table conversions

- First we generate an artificial dataset to play with (copy and paste to R to see what it does):
  ```
  long=expand.grid(SITE=c("A","B"),TREAT=c("low","med","high"), REP=c(1:5))
  long$YIELD=round(rnorm(10)*5+15); long
  ```
- Long-to-wide conversion with the `reshape2` package, where SITE and REP remain columns, but the treatment levels of TREAT now become several new columns:
  ```
  wide=dcast(long, SITE+REP~TREAT, value.var="YIELD")
  ```
  Wide-to-long conversion back to what we had before. The variables that you want to maintain as columns are SITE and REP, all others will be gathered into a new variable where the remaining columns become treatment levels. You have to fix the variable names to get to the original long:
  ```
  long2=melt(wide, id.vars=c("SITE","REP"))
  names(long2)=c("SITE","REP","TREAT","YIELD")
  ```

Dealing with missing values

- `transform(dat1, VAR1=ifelse(is.na(VAR1),0,VAR1))` sets missing values in variable VAR1 to 0. You may do this if missing has a biological meaning of zero, e.g. zero productivity.
- `transform(dat1, VAR1=ifelse(VAR1==0,NA,VAR1))` … or vice versa if zero really means that the measurement was not taken.
- `dat1[is.na(dat1)]=0` sets missing values to 0 in entire dataset.
- `dat1[dat1==0]=NA` ... vice versa.
- `dat2=na.omit(dat1)` delete rows with missing values in any variable
- `dat2=dat1[!is.na(dat1$VAR1),]` delete rows with missing values in VAR1
- `dat2=transform(dat1, VAR2=ifelse(is.na(VAR1),NA,VAR2))` modify a second variable (here: set to missing) based on missing values in a first variable.

Dealing with duplicate data entries or IDs:

- `unique(dat1)` or `dat1[!duplicated(dat1),]` removes exact duplicate rows.
- `dat1[duplicated(dat1),]` returns the duplicate rows.
- `dat1[!duplicated(dat1[,c("ID")]),]` removes all rows with duplicate IDs (first is kept).
- `dat1[duplicated(dat1[,c("ID")]),]` returns the rows with duplicate IDs.

Loops and automation

- `v1=vector(length=20)` initializes an empty vector with 20 elements. This is often required as an initial statement to subsequently write results of a loop into an output vector or output table.
- `m1=matrix(nrow=20, ncol=10)` similarly initializes an empty matrix with 20 rows and 10 columns.
- `for (i in 1:10) {` one or more operations with `v1[i]` or `m1[,i]` `}`
- `for (i in 1:10) { for (j in 1:20) {` one or more operations with `m1[j,i]` `}}`
- Example for an application, where a for-loop is used to calculate cumulative values. Copy and paste the code below into R to see what it does.

```
dat=round(rnorm(10)+2)
cum=vector(length=10)
cum[1]=dat[1]
for (i in 2:length(dat)) { cum[i]=cum[i-1]+dat[i] }
cbind(dat,cum)
```

- Example for an application where a for-loop allows automatic data processing of multiple files in a directory. This batch-converts DBF format files into CSV format files. With similar code, you could merge a large number of files into one master file, or do manipulations or analysis on multiple files consecutively.

```
library(foreign)
setwd("C:/your path/")
a<-list.files(); a
for (name in a) { dat1=read.dbf(name)
  write.csv(dat1, paste(name,".csv"), row.names=F, quote=F) }
```

Handy built-in functions

- `paste("hello", "world")` joins vectors after converting them to characters. The `sep=""` option can place any character string or nothing between values (a single space is the default)
- `substr("Year 1998",6,9)` extracts characters from start to stop position from vector
- `tolower("Year 1998")` convert to lowercase - handy to correct inconsistencies in data entry.
- `toupper("Year 1998")` convert to uppercase
- `nchar("Year 1998")` number of characters in a string, allows you to substring the last four digits of a variable regardless of length, for example: `substr(VAR1,nchar(VAR1)-3,nchar(VAR1))`
- Plenty of math functions, of course: `log(VAR1), log10(VAR1), log(VAR1,2), exp(VAR1), sqrt(VAR1), abs(VAR1), round(VAR1,2)`

Programming custom functions

- You can program your own functions, if something is missing, or if you want to utilize a bunch of code over and over to make similar calculations. Here is a clever example for calculating the statistical mode of a variable, which is missing from the built-in R functions.

```
mode=function(input){ freq=table(as.vector(input))
                      descending_freq=sort(-freq)
                      mode=names(descending_freq)[1]
                      as.numeric(mode)
                      }
VAR1=c(1,3,3,2,3,2,2,3,5,3)
mode(VAR1)
```

More information, help, and on-line resources

- Adding a question mark before a command or functions brings up a help file. E.g. `?paste`. Be sure to check out the example code at the end of the help file, which often helps to understand the syntax.
- More information and R resources can be found with the search engine http://www.rseek.org