Regression and regularization

Lecture 1d

Overview

- Linear regression
- Ridge Regression
- Lasso
- Variable selection

Simple linear regression

Model:

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

or

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

or

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

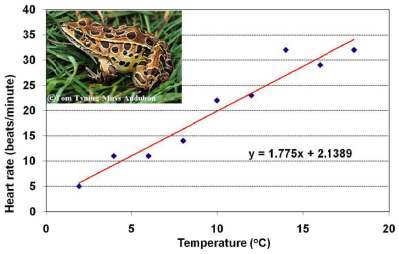
Terminology:

w_0: **intercept (or bias)**

w_1: **regression coefficient**

Response

The target responds directly and linearly to changes in the feature



Ordinary least squares regression (OLS)

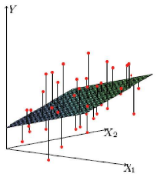
Model:

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

where

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

Why is "1" here?



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

Ordinary least squares regression

Given data set D

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

Estimation: maximizing the likelihood

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

Is equivalent to minimizing

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

Matrix formulation of OLS regression

Optimality condition:

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

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

Parameter estimates and predictions

- Least squares estimates of the parameters

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

- Predicted values

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

Hat matrix

Why is it called so?

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

Degrees of freedom

Definition:

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

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

$$df = trace(S)$$

- For linear regression, degrees of freedom is

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

Different types of features

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

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

Model becomes linear if to recompute:

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

Example of dummy coding:

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

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

.
.
.

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

Basis function expansion

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

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

Linear regression in R

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

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

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

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


An example of ordinary least squares regression

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

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

```
> 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 6.30e-13 ***
Year            39246       4226   9.288 5.25e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

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

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

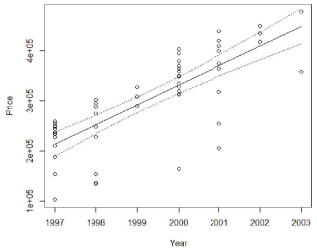
An example of ordinary least squares regression

```
> summary(fit2)

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

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

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

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


An example of ordinary least squares regression

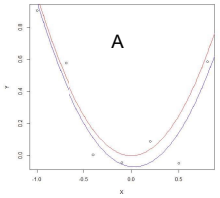
- Prediction

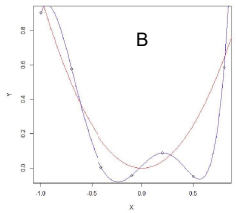
```
fitted <- predict(fit1, interval =
"confidence")

# plot the data and the fitted line
attach(mydata)
plot(Year, Price)
lines(Year, fitted[, "fit"])

# plot the confidence bands
lines(Year, fitted[, "lwr"], lty = "dotted",
col="blue")
lines(Year, fitted[, "upr"], lty = "dotted",
col="blue")
detach(mydata)
```



Ridge regression

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





Ridge regression

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

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

→ **l_2 regularization**

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

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

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

Ridge regression

Equivalent form

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

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

Solution

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

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

Hat matrix

How do we compute degrees of freedom here?

Ridge regression

Properties

- Extreme cases:
  - $\lambda = 0$ usual linear regression (no shrinkage)
  - $\lambda = +\infty$ fitting a constant ($w = 0$ except of w_0)
- When input variables are ortogonal (not realistic), $X^T X = I \rightarrow$

$$\hat{\boldsymbol{w}}^{\text{ridge}} = \frac{1}{1+\lambda}\mathbf{w}^{\mathbf{linreg}}$$ → coefficients are equally shrunk

- **Ridge regression is particularly useful if the explanatory variables are strongly correlated to each other.**
  - Correlated variables often correspond large w → shrunk
- Degrees of freedom decrease when λ increases
  - $\lambda = 0 \rightarrow d.f. = p$

Ridge regression

Properties

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

Ridge regression

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

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

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

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

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

Ridge regression

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

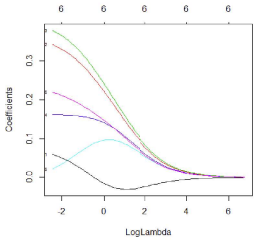
- Cycle time
- Memory
- Channels
- …

Build model predicting performance

Ridge regression

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

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

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

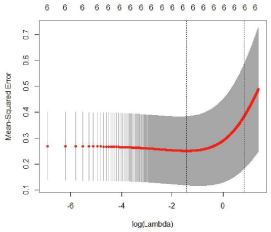


Ridge regression

- Choosing the best model by cross-validation:

```
model=cv.glmnet(as.matrix(covariates),
response, alpha=0,family="gaussian")
model$lambda.min
plot(model)
coef(model, s="lambda.min")
```

```
> coef(model, s="lambda.min")
7 x 1 sparse Matrix of class "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
>
```

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)**:**

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

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

LASSO

- Equivalently

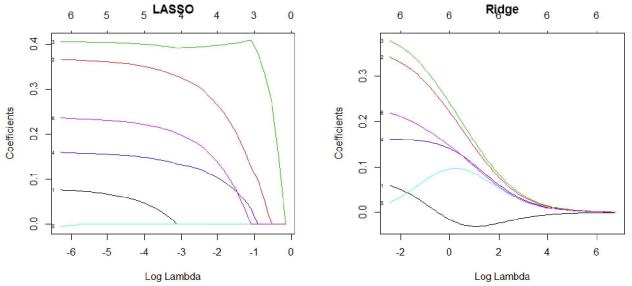
$$\hat{w}^{lasso} = \text{argmin}\sum_{i=1}^{N}(y_i - w_0 - w_1 x_{1,j} - \ldots - w_p x_{pj})^2$$

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

LASSO vs Ridge

- **LASSO yields sparse solutions!**

Example Computer hardware data



LASSO vs Ridge

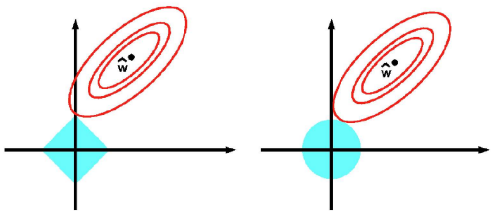
- Only 5 variables selected by LASSO

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

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


LASSO vs Ridge

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



LASSO properies

- **Lasso is widely used when $\boldsymbol{p \gg n}$**
  - 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

Variable selection

- .. Or “Feature selection”

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

Reasons:

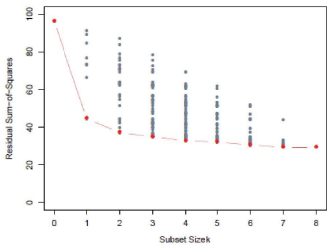
- Model can become overfitted (recall polynomial regression)
- Large number of predictors → model is difficult to use and interpret

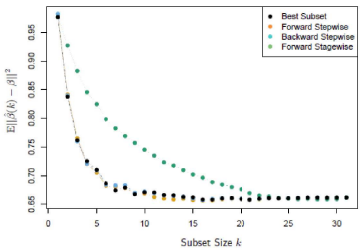
Variable selection

Alternative 1: Variable subset selection

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

RSS and MSE depend on k





Variable selection in R

- Use stepAIC() in MASS

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

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

Residuals:
      Min        1Q    Median        3Q       Max
-1.20212 -0.15512  0.03579  0.16567  2.42280

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

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

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

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

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


Model selection

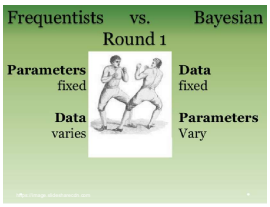
Lecture 1e

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

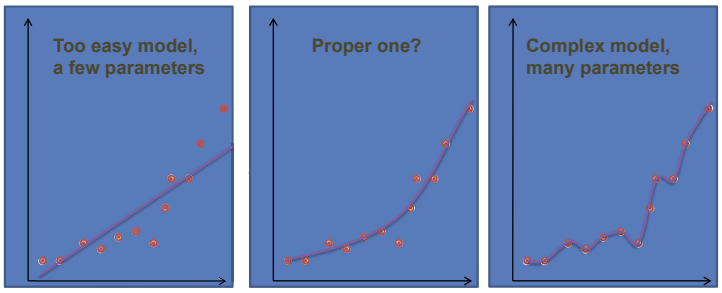
- $\widehat{\boldsymbol{w}} = \delta(D)$ (some function of your data) – an **estimator**
- Optimal parameter values?→there can be many ways to compute them (MLE, shrinkage…)
  - Compare Bayesian: given estimators $\boldsymbol{w^1}$ and $\boldsymbol{w^2}$, we **can** compare them! $p(\boldsymbol{w^1}|\boldsymbol{D}) > \boldsymbol{p}(\boldsymbol{w^2}|\boldsymbol{D})$
  - 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: $\boldsymbol{w} = (0, \ldots, 0, 1)$
- 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

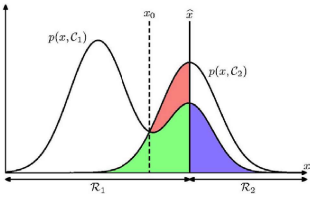
- Given a model, choose the optimal parameter values
  - Decision theory
- Define loss $L(Y, \hat{y})$
  - How much we loose in guessing true Y incorrectly
- If we know the true distribution $p(y, x|w)$ then we choose $\hat{y}$

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

Model selection

Example: Spam classification

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



Loss functions

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

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

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

Example: Compute loss function related to
  - Normal distribution

Guess why such loss function was chosen

Loss functions

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

Model selection

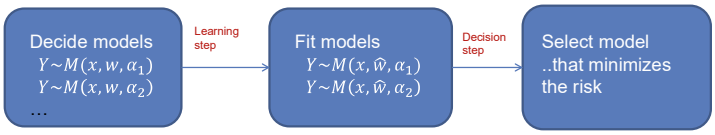
- **Problem**: true model and true $\boldsymbol{w}$ are unknown→can not compute expected loss!
- How to find an optimal model?
  - Consider what expected loss (**risk**) depends on
    $$R(Y,\hat{y}) = E[L(Y,\hat{y}(X,D))\,]$$
- Random factors:
  - D – **training set**
  - Y, X – data to be predicted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

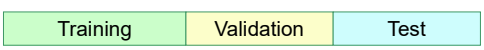
- Given data $D = \{\boldsymbol{X}_i, Y_i, i = 1 \ldots n\}$



- 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

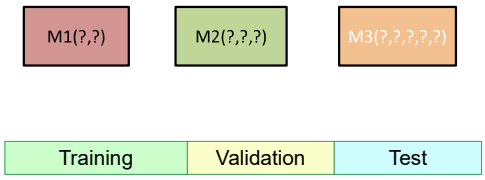
Divide into training, validation and test sets



- Choose proportions in some way

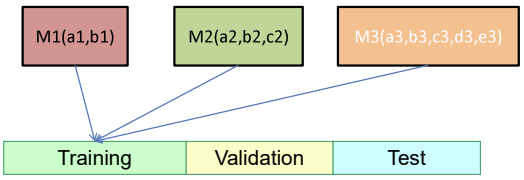
Holdout method

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



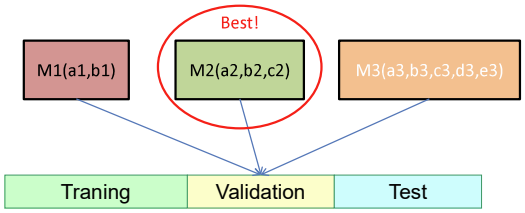
Holdout method

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



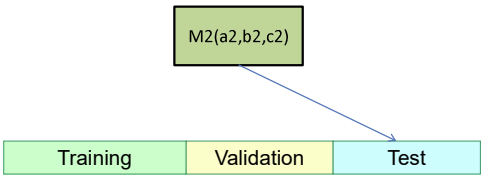
Holdout method

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

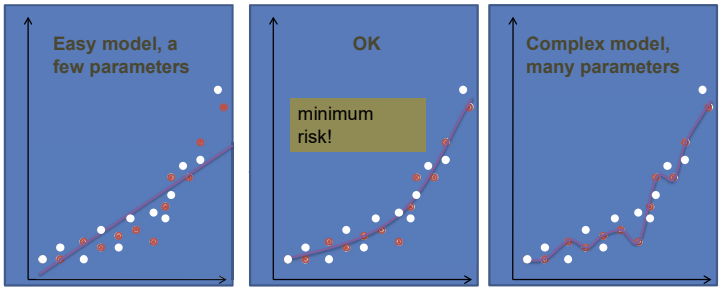


Holdout method

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



Holdout method



Holdout in R

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

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

- How to partition into train/valid/test?

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

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

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

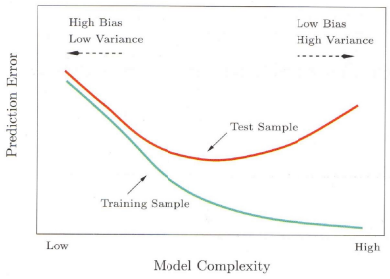
Bias-variance tradeoff

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

Bias-variance tradeoff

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

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



When loss is not quadratic, no such nice formula exist

Cross-validation

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

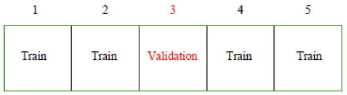
Cross-validation (Estimates Err)

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

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

Cross-validation

Cross-validation



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

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

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?

Cross-validation vs Holdout

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

Analytical methods

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

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

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

Analytical methods

- One can show that

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

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

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

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

Model selection

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

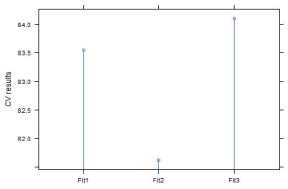
- Cycle time
- Memory
- Channels
- ...

Build model predicting performance

Cross-validatation

- Try models with different predictor sets

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

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



Linear classification methods

Lecture 2a

Overview

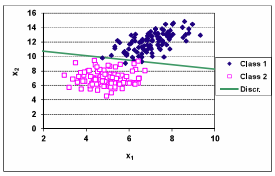
- Discriminant Analysis models
- Logistic regression
- Generalized Linear Model

Classification

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

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

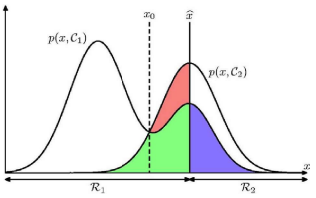
Disanvantages of deterministic classifiers:

- 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

Probabilities into decision

- Loss minimization

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



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

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

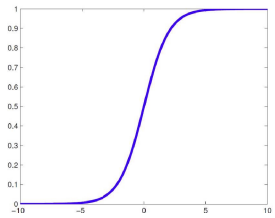
Logistic regression

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

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

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

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



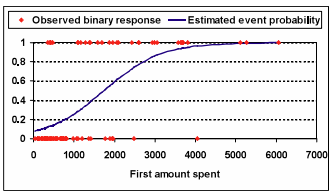
Logistic regression

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

The log of the odds is linear in *x*

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

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

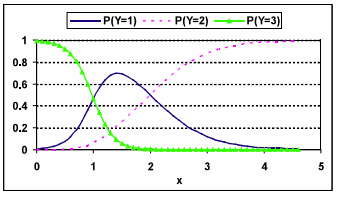


Logistic regression

- When Y is categorical,

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

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



Logistic regression

Fitting logistic regression

- In binary case,

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

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

Estimation:

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

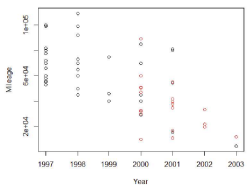
Decision boundaries of logistic regression are linear

Logistic regression

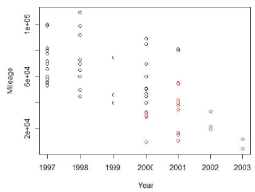
- In R, use glm() with family="binomial"
  - Predicted probabilities: predict(fit,newdata, type="response")

Example Equipment=f(Year, mileage)

Original data



Classified data

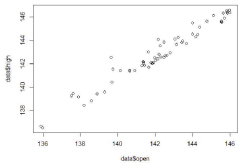


Moving beyond typical distributions

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

Example 1: Daily Stock prices NASDAQ
- Open
- High (within day)

Does it seem that the error is normal here?



Example 2: Number of calls to bank
- Y=Number of calls
- X= time

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

Exponential family

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

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

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

Example: Bernoulli

Generalized linear models

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

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

- If f is a canonical link, then

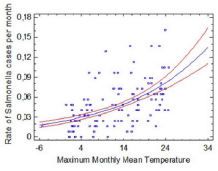
$$p(y|w,s) = h(y,s)g(\boldsymbol{w},\boldsymbol{x})e^{\frac{(\boldsymbol{w}^T\boldsymbol{x})y}{s}}$$

Generalized linear models

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

Example: Poisson regression

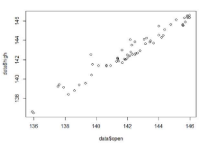
$f^{-1}(\mu) = e^{\mu}, Y \sim Poisson\left(e^{\boldsymbol{w}^T\boldsymbol{x}}\right)$



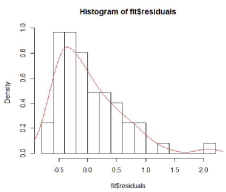
Generalized linear model: software

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

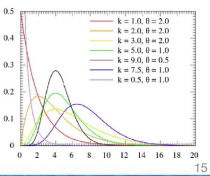
Example: Daily Stock prices NASDAQ
- Open
- High (within day)



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



Gamma distribution: Wikipedia



Generalized linear model

Assume

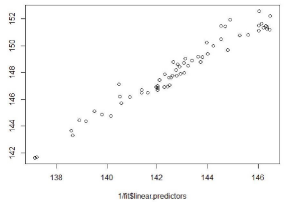
$$High \sim Gamma(1, \frac{1}{w_0 + w_1 Open})$$

What is link function here?

New generated data

- Has similar pattern as original data!

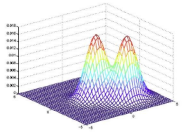
```
Call:
glm(formula = high ~ open, family = Gamma(link = "inverse"),
    data = data)

Deviance Residuals:
      Min         1Q     Median         3Q        Max
-0.0052879  -0.0028896  -0.0006678   0.0016598   0.0148083

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.355e-02  1.962e-04   69.06   <2e-16 ***
open        -4.604e-05  1.379e-06  -33.39   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```



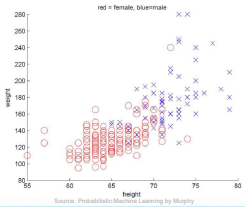
Quadratic discriminant analysis

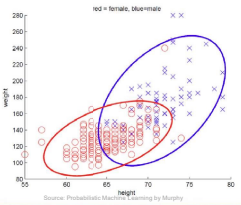
- Generative classifier
- Main assumptions:
  - $\boldsymbol{x}$ is now **random** as well as y

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



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

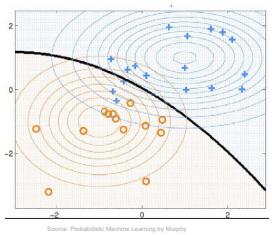




Quadratic discriminant analysis

- If parameters are estimated, classify:

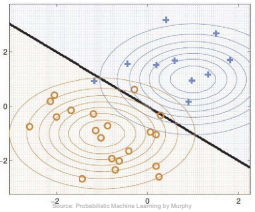
$$\hat{y}(\boldsymbol{x}) = \arg\max_c p(y = c|\boldsymbol{x}, \theta)$$



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**:

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



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**),

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

LDA and QDA: code

- Syntax in R, library **MASS**

lda(formula, data, ..., subset, na.action)
- Prior – class probabiliies
- Subset – indices, if training data should be used

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

predict(..)

LDA: output

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

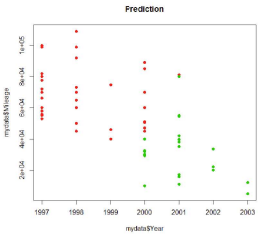
```
> print(resLDA)
Call:
lda(Equipment ~ Mileage + Year, data = mydata)

Prior probabilities of groups:
        0         1
0.6440678 0.3559322

Group means:
   Mileage      Year
0 63539.21 1998.447
1 36857.62 2000.762

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


LDA: output

- Misclassified items

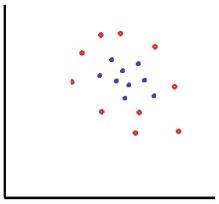
```
> table(Pred$class, mydata$Equipment)

    0  1
0  31  6
1   7 15
```

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



LDA versus Logistic regression

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



LDA versus Logistic regression

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

Naïve Bayes classifiers
Decision trees

Lecture 2b

Naive Bayes classifiers: motivation

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

Naive Bayes classifiers: motivation

Idea: use Bayes classifier

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

Chance of observing a given combination of words in science fiction

Proportion of science fiction documents

Naive Bayes classifiers: motivation

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

Naive Bayes classifiers

- Naive Bayes assumption: conditional independence

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

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

Naive Bayes classifiers - discrete inputs

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

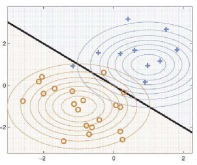
Naive Bayes classifiers - discrete inputs

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

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

Naive Bayes – continuous inputs

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

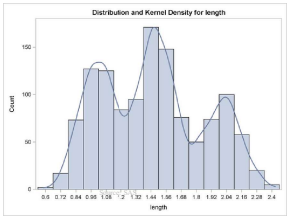


- Naive bayes is a special case of LDA (given A)
  - →MLE are means and variances (per class)

Naive Bayes – continuous inputs

- Assumption B: $p(x_j|y = C)$ are unknown functions of x_j that can be estimated from data
  - Nonparametric density estimation (kernel for ex.)

1. Estimate $p(X_i = x_j|Y = y_k)$ using nonparametric methods
2. Estimate $p(Y = y_k)$ as class proportions
3. Use Bayes rule and 0-1 loss to classify



Naive Bayes in R

- naiveBayes in package **e1071**

Example: Satisfaction of householders with their present housing circumstances

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

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

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

```
> table(Yfit,housing1$Sat)

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

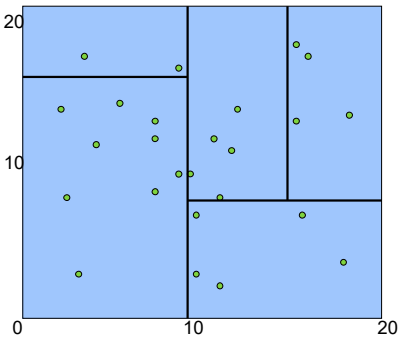

Decision trees

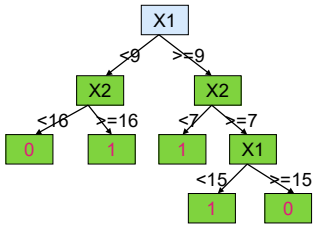
Idea

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

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

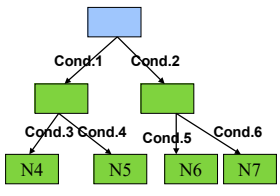
Classification tree toy example



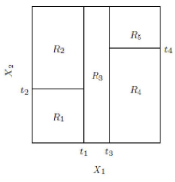


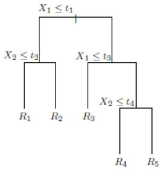
Definitions

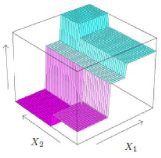
- Root node
- Nodes
- Leaves (terminal nodes)
- Parent node, child node
- Decision rules
- A value is assigned to the leaves



Regression tree toy example





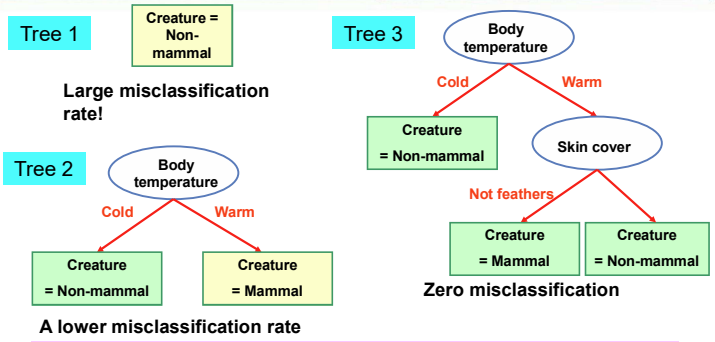


A classification problem

Create a classification tree that would describe the following patterns

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

Several solutions



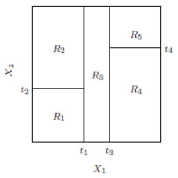
Green boxes represent pure nodes =nodes where observed values are the same

Decision trees

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



Decision trees

Example:

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

Classification trees

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

Class proportions

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

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

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

Classification trees

- Impurity measure $\mathrm{Q}(R_m)$
  - 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 $\mathrm{Q}(R_m)\ N(R_m)$

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

Fitting regression trees: CART

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

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

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

and solve

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

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

CART: comments

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

Optimal trees

- **Postpruning**

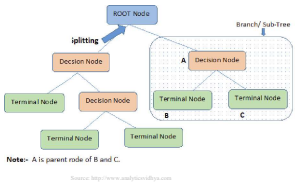
Weakest link pruning:

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

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

$|T|$ =#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!

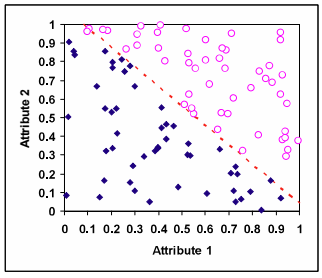
Decision trees: comments

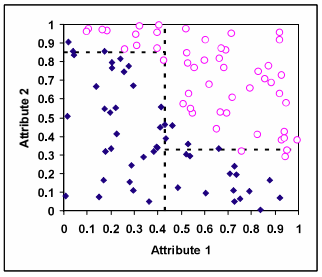
- Similar algorithms work for regression trees – replace $N \cdot Q(R)$ by $SSE(R)$
- Easy to interpret
- Easy to handle all types of predictors in one model
- **Automatic variable selection**
- Relatively robust to outliers
- Handle large datasets

- Trees have high variance: a small change in response→ totally different tree
- Greedy algorithms → fit may be not so good
- Lack of smoothness

Decision trees: issues

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





Decision trees in R

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

tree(formula, data, weights, control, split = c("deviance", "gini"), ...)

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

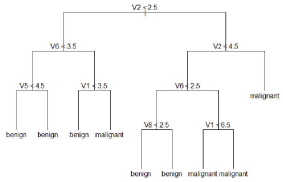
Example: breast cancer as a function av biological measurements

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

Decision trees in R

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

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

 1) root 683 884.400 benign ( 0.650073 0.349927 )
   2) V2 < 2.5 418 108.900 benign ( 0.971292 0.028708 )
     4) V6 < 3.5 395  25.330 benign ( 0.994937 0.005063 )
       8) V5 < 4.5 389   1.000 benign ( 1.000000 0.000000 ) *
       9) V5 > 4.5 6   7.638 benign ( 0.666667 0.333333 ) *
     5) V6 > 3.5 23  31.490 benign ( 0.565217 0.434783 )
      10) V1 < 3.5 11   0.000 benign ( 1.000000 0.000000 ) *
      11) V1 > 3.5 12  10.810 malignant ( 0.166667 0.833333 ) *
   3) V2 > 2.5 265 217.900 malignant ( 0.143396 0.856604 )
     6) V2 < 4.5 90 120.300 malignant ( 0.388889 0.611111 )
      12) V6 < 2.5 30  27.030 benign ( 0.833333 0.166667 )
        24) V8 < 2.5 19   0.000 benign ( 1.000000 0.000000 ) *
        25) V8 > 2.5 11   5.160 benign ( 0.545455 0.454545 ) *
      13) V6 > 2.5 60  54.070 malignant ( 0.166667 0.833333 )
        26) V1 < 6.5 28  35.160 malignant ( 0.321429 0.678571 ) *
        27) V1 > 6.5 32   8.900 malignant ( 0.031250 0.968750 ) *
     7) V2 > 4.5 175  30.150 malignant ( 0.017143 0.982857 ) *
```




```
> summary(fit)

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


Decision trees in R

- Misclassification results

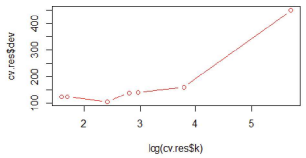
```
Yfit=predict(fit, newdata=biopsy, type="class")
table(biopsy$class,Yfit)
```

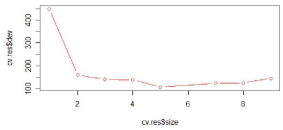
```
> table(biopsy$class,Yfit)
           Yfit
            benign malignant
  benign       440        18
  malignant      7       234
```


Decision trees in R

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

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

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



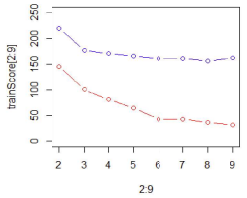


What is optimal number of leaves?

Decision trees in R

- Selecting optimal tree by train/validation

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

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

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



What is optimal number of leaves?

Decision trees in R

- Final tree: 5 leaves

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

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


Lecture 2c

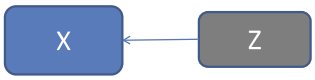
Latent variable models

Overview

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

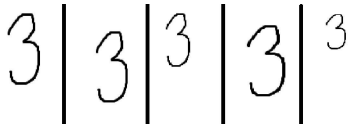
Latent variables

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



Latent variables

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



Principal Component Analysis (PCA)

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

Applications

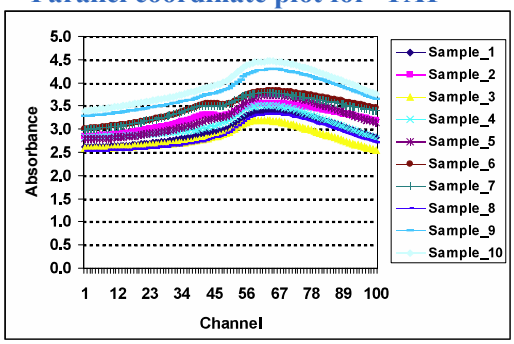
- Image recognition
- Information compression
- Subspace clustering
- …

Principal Component Analysis (PCA)

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

Absorbance records for ten samples of chopped meat

Parallel coordinate plot for "FAT"

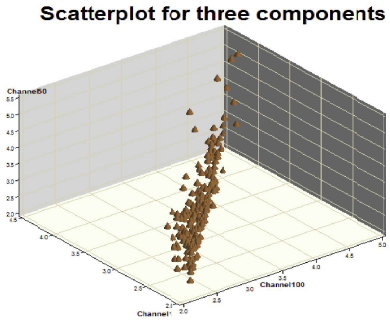


1 target (fat)

100 features (absorbance at 100 wavelengths or channels)

The features are strongly correlated to each other

3-D plots of absorbance records for samples of meat - channels 1, 50 and 100



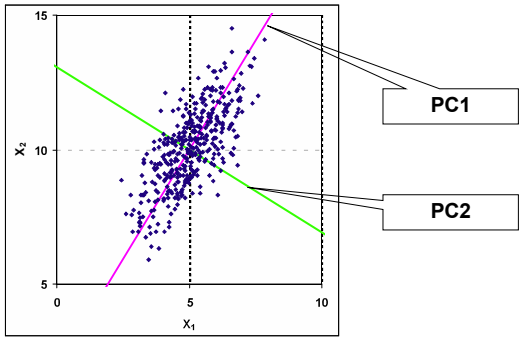
Principal components analysis

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

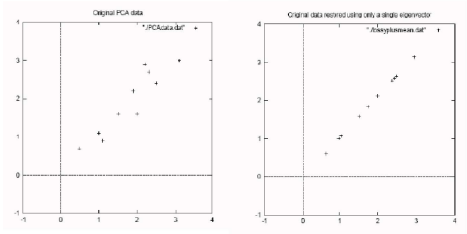
- The first principal component (PC1) is the direction that maximizes the variance of the projected data
- The second principal component (PC2) is the direction that maximizes the variance of the projected data after the variation along PC1 has been removed
- The third principal component (PC3) is the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

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

Principal Component Analysis - two inputs



PCA- after reducing dimensionality

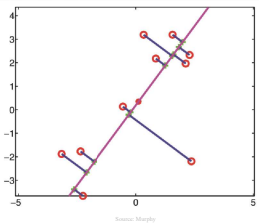


- Data became approximate (but less data to store)
- $PC_1, \ldots PC_M$ are actually eigenvections of **sample covariance** (first largest eigenvalue,…,Mth largest egenvalue)

PCA: another view

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

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



PCA: computations

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

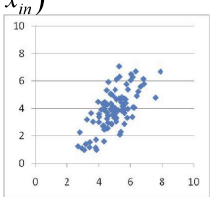
1. Centred data

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

2. Covariance matrix

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

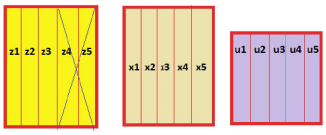
3. Search for eigenvectors and eigenvalues of **S**



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

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

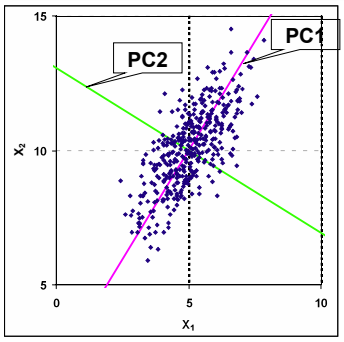
Getting approximate original data:

$$X' = ZU_M^T$$

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

100*50 vs
100*4+50*4

Principal Component Analysis



Eigenanalysis of the Covariance Matrix

Eigenvalue	2.8162	0.3835
Proportion	0.880	0.120
Cumulative	0.880	1.000

Variable	PC1	PC2
X1	0.523	0.852
X2	0.852	-0.523

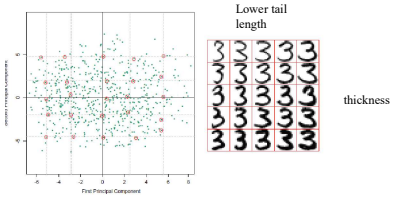
Loadings (U)

Principal Component Analysis

- Digits: two eigenvectors extracted

$$x = \bar{3} + z1 \cdot \bar{3} + z2 \cdot \bar{3} \dots$$

- Interptretation of eigenvectiors

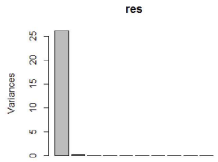


PCA in R

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

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

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

> sprintf("%2.3f",lambda/sum(lambda)*100)
  [1] "98.679" "0.901"  "0.296"  "0.114"  "0.006
  [9] "0.000"  "0.000"  "0.000"  "0.000"  "0.000
```



Only 1 component captures the 99% of variation!

PCA in R

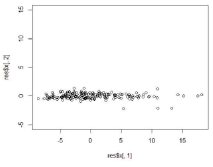
- Principal component **loadings (U)**

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

```
> head(U)
                PC1       PC2        PC3
Channel1 0.07938192 0.1156228 0.08073156 -0.0927
Channel2 0.07987445 0.1170972 0.07887873 -0.0981
Channel3 0.08036498 0.1185571 0.07702127 -0.1031
Channel4 0.08085611 0.1200006 0.07515015 -0.1077
Channel5 0.08135022 0.1214075 0.07323819 -0.1119
Channel6 0.08184896 0.1227491 0.07125048 -0.1156
```

- Data in (PC1, PC2) – **scores (Z)**

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

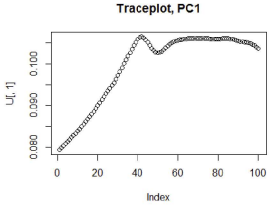


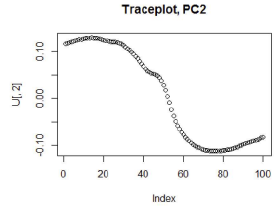
Do we need second dimension?

PCA in R

- Trace plots

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

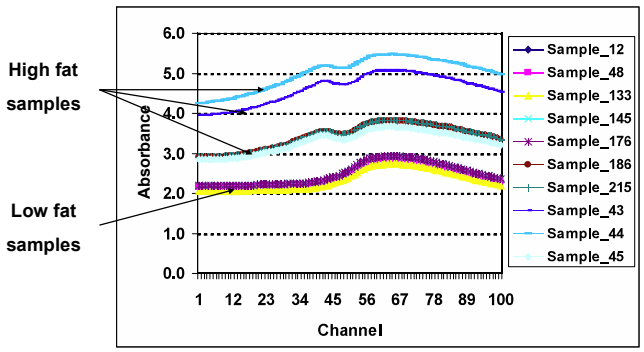




Which components contribute to PC1-2?

Absorbance records for ten samples of chopped meat

PCA2 captures the most of remaining variation



PCA for high-dimensional data

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

Probabilistic PCA

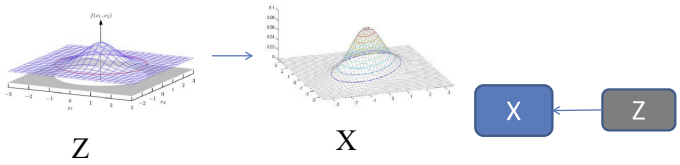
- z_i-latent variables, x_i- observed variables

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

- 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.



Probabilistic PCA

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

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

- 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'

Probabilistic PCA

- Estimation of parameters: ML

Theorem. ML estimates are given by

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

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

Probabilistic PCA

- Estimation of Z
  - Use mean of posterior

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

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

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

Advantages of probabilistic PCA

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

Probabilistic PCA in R

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

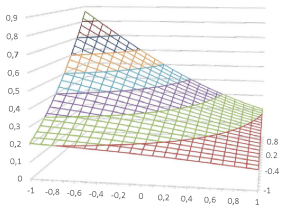
Ppca(data, nPcs,…)

Results: scores, loadings…

Independent component analysis (ICA)

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

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



ICA

- Model

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

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

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

Subject to $||v_i|| = 1$

ICA

- Setting $G_j(z) = -\log\left(p_j(z)\right), z_j = v_i^T \boldsymbol{x}$ and assuming large sample

$$\min_V \sum_{j=1}^{p} E\left(G_j(z_j)\right)$$

Subject to $||v_i|| = 1$

- **Prewhitening**
  - Use PCA: $X' = XU$
  - Computing z_is for given V: $Z = X'V$

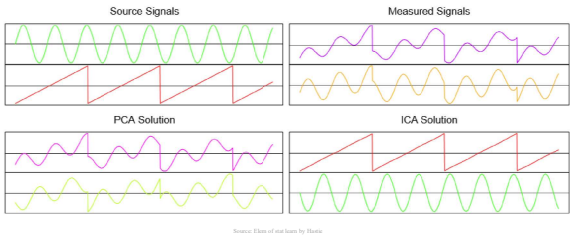
ICA

- **Estimation B**: maximize negentropy
  - ICA looks for model which is as much non-Gaussian as possible
- **Entropy** $H(z) = -\int p(z) \log p(z)\, dz = E\left(-\log p\,(z)\right)$
- **Negentropy** $J(z_i) = H(z_i') - H(z_i)$
  - $z_i' \sim N(E z_i, var(z_i))$
- **Negentropy maximization**

$$\max_V \sum_{j=1}^{p} J(z_j) = \min_V \sum_{j=1}^{p} H(z_j) = \min_V \sum_{j=1}^{p} E\left(-\log p\,(z_j)\right)$$

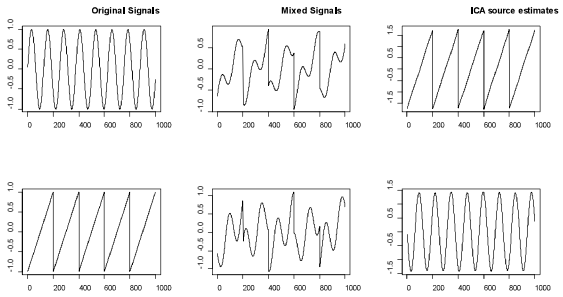
ICA

- Example



Independent component analysis: R

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



Principal component regression. Uncertainty estimation

Lecture 2d

Principle component regression

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

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

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

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

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

PCR: comments

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

Partial Least Squares Regression (PLS)

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

PCR $$\max_{\alpha} \mathrm{Var}(\mathbf{X}\alpha)$$
$$\text{subject to } ||\alpha|| = 1,\ \alpha^T \mathbf{S} v_\ell = 0,\ \ell = 1, \ldots, m-1,$$

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

Partial least squares regression (PLS)

Step 1: Standardize features to mean zero and variance one

Step 2: Compute the first derived feature by setting

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

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

Step 3: Orthogonalize $\mathbf{x}_1 \ldots \mathbf{x}_m$ with respect to z_1

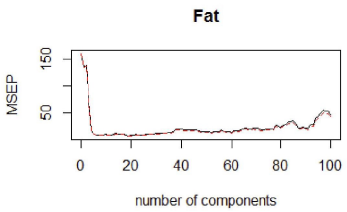
Step 4: repeat from step 2 and find $\mathbf{z}_2 .. \mathbf{z}_M$

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

PCR and PLS: R

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

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

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



PCR and PLS: R

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

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

```
TRAINING: % variance explained
     1 comps  2 comps  3 comps  4 comps  5 comps  6 comp
s
X      98.65    99.59    99.88    99.99   100.00   100.0
0
Fat    16.79    16.98    56.44    89.97    93.62    95.2
6
```


PCR

- Select 3 components

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

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

Coefficients in the original variables

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

                Fat
Channel1   -2.61109872
Channel2   -2.56607281
Channel3   -2.52081831
Channel4   -2.47510841
Channel5   -2.42831883
Channel6   -2.37920922
Channel7   -2.32674365
Channel8   -2.27010925
Channel9   -2.20867856
Channel10  -2.14358679
```


PCR

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

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

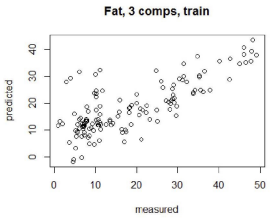
Scores matrix (new coordinates)

```
> scores(pcr.fit1)
           Comp 1       Comp 2       Comp 3
155  -9.66855445  0.092805568 -0.1012873027
188  -1.04084544 -0.307978589 -0.3180672681
163  -1.92212872 -0.243714308  0.0124365801
214   1.27768585 -0.232939083 -0.4315433137
97    2.55797242 -0.741279740  0.1582517775
35  -11.17415228  2.124582502  0.2004058835
68    2.96005563 -0.312953959  0.1945895756
106   3.71331162  0.015766621 -0.1130155332
```


PCR

```
> Yloadings(pcr.fit1)

Loadings:
    Comp 1  Comp 2  Comp 3
Fat  -1.015   1.114 -28.847
```

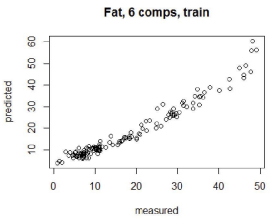
What is the regression equation in the new coordinates?



Is fit OK?

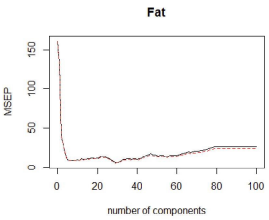
PCR

- Now 6 components




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


PLS




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


Probabilistic models

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

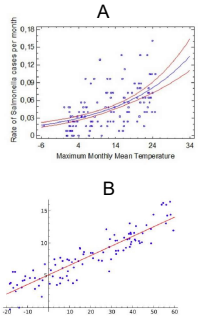
$$y = f(x, w)$$

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

Probabilistic models

Arguments against deterministic models:

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



Probabilistic models

Probabilistic model

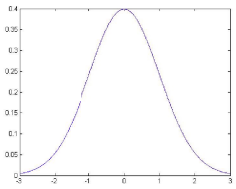
$$Y \sim Distribution(f(x, w), \theta)$$

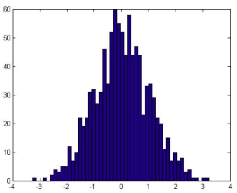
- Data is fully explained (error as well)
- Automatic principle for finding parameters: MLE , MAP or Bayes theorem
- Automatic principle for finding uncertainty (conf. limits)
  - **Bootstrap**
  - Posterior probability
- Possibility to generate new data of the same type
  - Further testing of the model

Uncertainty estimation

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

The bootstrap: general principle



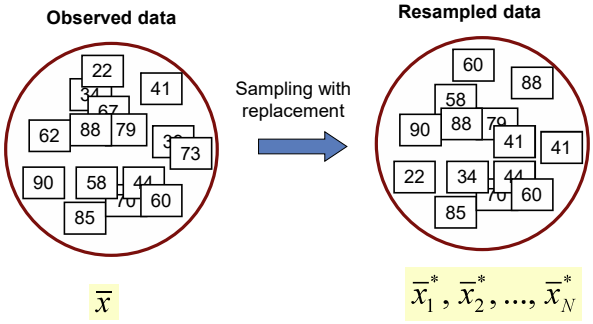


We want to determine uncertainty of $\hat{f}(D, X)$

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

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

Nonparametric bootstrap



$\bar{x}$

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

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

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 $\mathbf{D_1}$ =(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

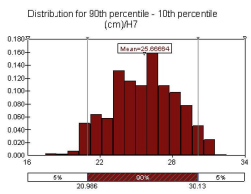
- To estimate 100(1-α) confidence interval for w

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

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

Look at the plot…



Bootstrap: regression context

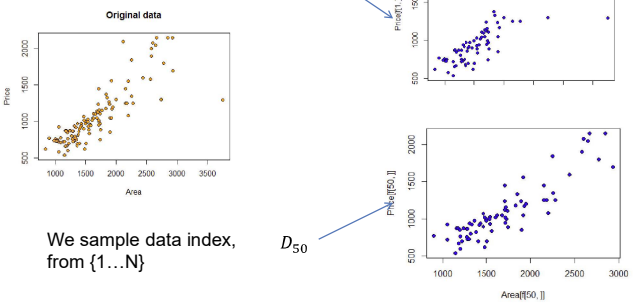
- Model $Y \sim F(X, w)$
- Data $D = \{(Y_i, X_i), i = 1, \dots, n\}$
- Idea: produce several bootstrap sets that are similar to D

Nonparametric bootstrap:

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

Uncertainty estimation

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



We sample data index, from {1…N}

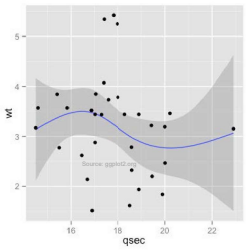
Bootstrap: regression context

Parametric bootstrap

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

Confindence intervals in regression

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



Confindence intervals in regression

Estimation

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

Bootstrap: R

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

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

Nonparametric bootstrap:

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

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

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

Bootstrap: R

Parametric bootstrap:

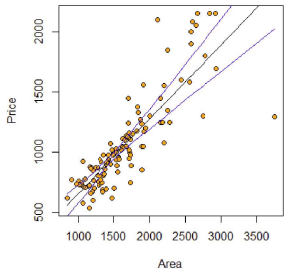
- Compute value *mle* that estimates model parameters from the data
- Write function *ran.gen* that depends on *data* and *mle* and which generates new data
- Write function *statistic* that depend on *data* which will be generated by *ran.gen* and should return the estimator

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

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

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

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


Uncertainty estimation: R

- Bootstrap cofidence bands for linear model

```
e=envelope(res) #compute confidence bands

fit=lm(Price~Area, data=data2)
priceP=predict(fit)

plot(Area, Price, pch=21, bg="orange")
points(data2$Area,priceP,type="l") #plot fitted line

#plot cofidence bands
points(data2$Area,e$point[2,], type="l", col="blue")
points(data2$Area,e$point[1,], type="l", col="blue")
```



Prediction bands

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

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

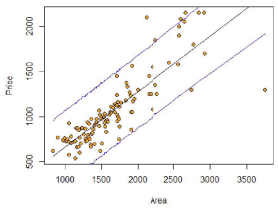
Prediction band for parametric bootstrap

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

Estimation of the model quality

Example: parametric bootstrap

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

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



Why wider band?

732A95/TDDE01 Introduction to Machine Learning

Lecture 3a Block 1: Kernel Methods

Jose M. Peña
IDA, Linköping University, Sweden

Contents

Literature

- Main source
  - Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006. Sections 2.5 and 6.1-6.2.
- Additional source
  - Devroye, L., Györfi, L. and Lugosi, G. *A Probabilistic Theory of Pattern Recognition*. Springer, 1996. Sections 6.4 and 10.0.
  - Hastie, T., Tibshirani, R. and Friedman, J. *The Elements of Statistical Learning*. Springer, 2009. Chapter 6.

Histogram Classification

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

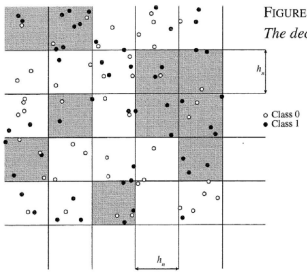


FIGURE 6.1. *A cubic histogram rule: The decision is 1 in the shaded area.*

- In other words,

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

Moving Window Classification

- The histogram rule is less accurate at the borders of the cube, because those points are not as well represented by the cube as the ones near the center. Then,
  - consider the points within a certain distance to the point to classify, and
  - classify the point according to majority vote.

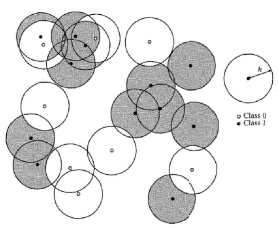


FIGURE 10.1. *The moving window rule in $\mathcal{R}^2$. The decision is 1 in the shaded area.*

- In other words,

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

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

Kernel Classification

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

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

where $k : \mathbb{R}^D \to \mathbb{R}$ is a kernel function, which is usually non-negative and monotone decreasing along rays starting from the origin. The parameter h is called smoothing factor or width.

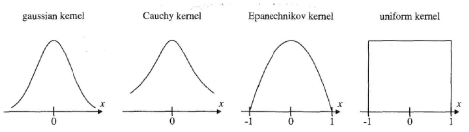


FIGURE 10.3. *Various kernels on $\mathcal{R}$.*

- Gaussian kernel: $k(u) = exp(-\|u\|^2)$ where $\|\cdot\|$ is the Euclidean norm.
- Cauchy kernel: $k(u) = 1/(1 + \|u\|^{D+1})$
- Epanechnikov kernel: $k(u) = (1 - \|u\|^2)\mathbf{1}_{\{\|u\| \leq 1\}}$
- Moving window kernel: $k(u) = \mathbf{1}_{\{u \in S(0,1)\}}$

Kernel Classification

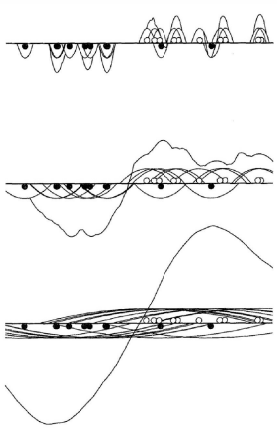


FIGURE 10.2. *Kernel rule on the real line. The figure shows $\sum_{i=1}^n (2Y_i - 1)K((x - X_i)/h)$ for $n = 20$, $K(u) = (1 - u^2)I_{\{|u| \leq 1\}}$ (the Epanechnikov kernel), and three smoothing factors h. One definitely undersmooths and one oversmooths. We took $p = 1/2$, and the class-conditional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

Histogram, Moving Window, and Kernel Regression

- Consider regressing an unidimensional continuous random variable on a D-dimensional continuous random variable.
- The best regression function under the squared error loss function is $y^*(\boldsymbol{x}) = \mathbb{E}_Y[y|\boldsymbol{x}]$.
- Since $\boldsymbol{x}$ may not appear in the finite training set $\{(\boldsymbol{x}_n, t_n)\}$ available, then we average over the points in $C(\boldsymbol{x}, h)$ or $S(\boldsymbol{x}, h)$, or kernel-weighted average over all the points.
- In other words,

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

or

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

or

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

Histogram, Moving Window, and Kernel Density Estimation

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

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

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

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

and thus

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

- Then,

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

or

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

or

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

assuming that $k(u) \geq 0$ for all u and $\int k(u) du = 1$.

Histogram, Moving Window, and Kernel Density Estimation

Figure 2.24 An illustration of the histogram approach to density estimation, in which a data set of 50 data points is generated from the distribution shown by the green curve. Histogram density estimates, based on (2.241), with a common bin width Δ are shown for various values of Δ.

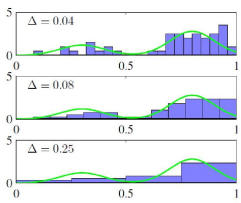
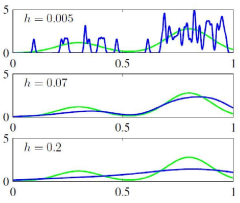


Figure 2.25 Illustration of the kernel density model (2.250) applied to the same data set used to demonstrate the histogram approach in Figure 2.24. We see that h acts as a smoothing parameter and that if it is set too small (top panel), the result is a very noisy density model, whereas if it is set too large (bottom panel), then the bimodal nature of the underlying distribution from which the data is generated (shown by the green curve) is washed out. The best density model is obtained for some intermediate value of h (middle panel).



Histogram, Moving Window, and Kernel Density Estimation

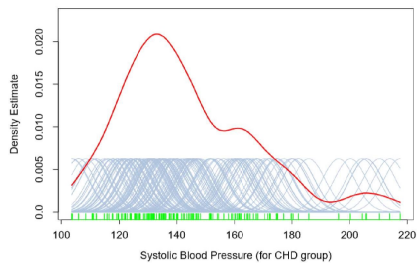


FIGURE 6.13. *A kernel density estimate for systolic blood pressure (for the CHD group). The density estimate at each point is the average contribution from each of the kernels at that point. We have scaled the kernels down by a factor of 10 to make the graph readable.*

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

Histogram, Moving Window, and Kernel Density Estimation

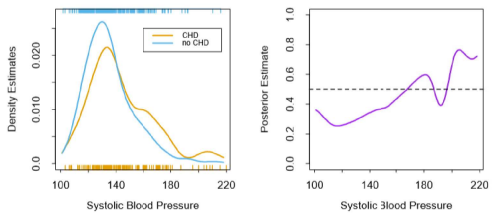
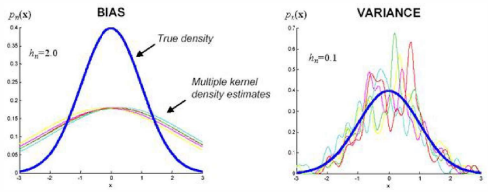


FIGURE 6.14. *The left panel shows the two separate density estimates for systolic blood pressure in the CHD versus no-CHD groups, using a Gaussian kernel density estimate in each. The right panel shows the estimated posterior probabilities for CHD, using (6.25).*

Kernel Selection

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



- Small width implies considering few points. So, the variance will be large (similar to the variance of a single point). The bias will be small since the points considered are close to $\boldsymbol{x}$.
- Large width implies considering many points. So, the variance will be small and the bias will be large.

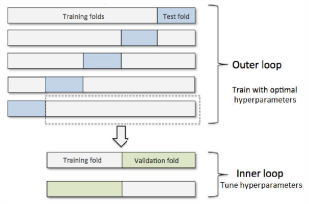
Kernel Selection

- Recall the following from previous lectures.
- Cross-validation is a technique to estimate the prediction error of a model.

- If the training set contains N points, note that cross-validation estimates the prediction error when the model is trained on $N - N/K$ points.
- Note that the model returned is trained on N points. So, cross-validation overestimates the prediction error of the model returned.
- This seems to suggest that a large K should be preferred. However, this typically implies a large variance of the error estimate, since there are only N/K test points.
- Typically, $K = 5, 10$ works well.

Kernel Selection

- Model: For example, ridge regression with a given value for the penalty factor λ. Only the parameters (weights) need to be determined (closed-form solution).
- Model selection: For example, determine the value for the penalty factor λ. Another example, determine the kernel and width for kernel classification, regression or density estimation. In either case, we do not have a continuous criterion to optimize. Solution: **Nested** cross-validation.

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

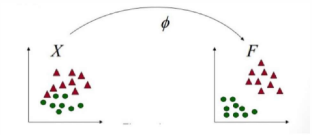
- Error overestimation may not be a concern for model selection. So, $K = 2$ may suffice in the inner loop.
- Which is the fitted model returned by nested cross-validation ?

Kernel Trick

- The kernel function $k\left(\frac{\mathbf{x}-\mathbf{x}'}{h}\right)$ is invariant to translations, and it can be generalized as $k(\boldsymbol{x}, \boldsymbol{x}')$. For instance,
  - Polynomial kernel: $k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T\boldsymbol{x}' + c)^M$
  - Gaussian kernel: $k(\boldsymbol{x}, \boldsymbol{x}') = exp(-\|\boldsymbol{x} - \boldsymbol{x}'\|^2/2\sigma^2)$
- If the matrix

$$\begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \vdots & \dots & \vdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{pmatrix}$$

is symmetric and positive semi-definite for all choices of $\{\boldsymbol{x}_n\}$, then $k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T\phi(\boldsymbol{x}')$ where $\phi(\cdot)$ is a mapping from the input space to the feature space.



- The feature space may be non-linear and even infinite dimensional. For instance,

$$\phi(\boldsymbol{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2c}x_1, \sqrt{2c}x_2, c)$$

for the polynomial kernel with $M = D = 2$.

Kernel Trick

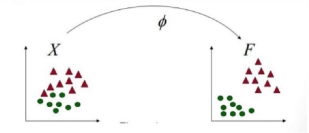
- Consider again moving window classification, regression, and density estimation.
- Note that $\boldsymbol{x}_n \in S(\boldsymbol{x}, h)$ if and only if $\|\boldsymbol{x} - \boldsymbol{x}_n\| \leq h$.
- Note that

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

- Then,

$$\begin{aligned} \|\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}_n)\| &= \phi(\boldsymbol{x}^T)\phi(\boldsymbol{x}) + \phi(\boldsymbol{x}_n^T)\phi(\boldsymbol{x}_n) - 2\phi(\boldsymbol{x}^T)\phi(\boldsymbol{x}_n) \\ &= k(\boldsymbol{x}, \boldsymbol{x}) + k(\boldsymbol{x}_n, \boldsymbol{x}_n) - 2k(\boldsymbol{x}, \boldsymbol{x}_n) \end{aligned}$$

- So, the distance is now computed in a (hopefully) more convenient space.



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

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:

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') &= ck_1(\mathbf{x}, \mathbf{x}') && (6.13) \\ k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}') && (6.14) \\ k(\mathbf{x}, \mathbf{x}') &= q(k_1(\mathbf{x}, \mathbf{x}')) && (6.15) \\ k(\mathbf{x}, \mathbf{x}') &= \exp(k_1(\mathbf{x}, \mathbf{x}')) && (6.16) \\ k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') && (6.17) \\ k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}') && (6.18) \\ k(\mathbf{x}, \mathbf{x}') &= k_3(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')) && (6.19) \\ k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}' && (6.20) \\ k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) && (6.21) \\ k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a)k_b(\mathbf{x}_b, \mathbf{x}'_b) && (6.22) \end{aligned}$$

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.

Summary

- Kernel methods: Smoothing models.
- Model selection: Nested cross-validation.
- Kernel trick: It allows to work in the feature space without constructing it.

732A95/TDDE01 Introduction to Machine Learning

Lecture 3b Block 1: Support Vector Machines

Jose M. Peña
IDA, Linköping University, Sweden

Contents

Literature

- Main source
  - Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006. Section 7.1.

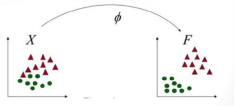
Support Vector Machines for Classification

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

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

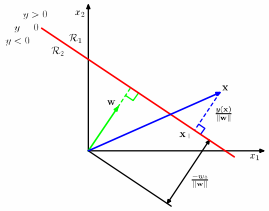
so that a new point $\boldsymbol{x}$ is classified according to the sign of $y(\boldsymbol{x})$.

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



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

Support Vector Machines for Classification



- The perpendicular distance from any point to the hyperplane is given by

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

- Then, the maximum margin separating hyperplane is given by

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

- 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.

Support Vector Machines for Classification

- Then, the maximum margin separating hyperplane is given by

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

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

- To minimize the previous expression, we minimize

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

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

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

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

$$0 = \sum_n a_n t_n$$

Support Vector Machines for Classification

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

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

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

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

Support Vector Machines for Classification

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

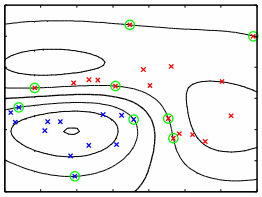
$$a_n(t_n y(\boldsymbol{x}_n) - 1) = 0$$

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

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

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

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



Support Vector Machines for Classification

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

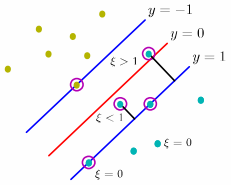
$$1 = t_n y(\boldsymbol{x}_n) = t_n\Big(\sum_{m\in\mathcal{S}} a_m t_m k(\boldsymbol{x}_n, \boldsymbol{x}_m) + b\Big)$$

and multiplying both sides by t_n, we have that

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

- We now drop the assumption of linear separability in the feature space, e.g. to avoid overfitting. We do so by introducing the slack variables $\xi_n \geq 0$ to penalize (almost-)misclassified points as

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

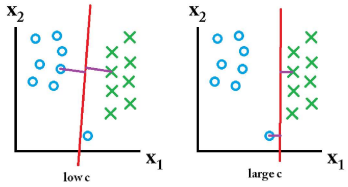


Support Vector Machines for Classification

- The optimal separating hyperplane is given by

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

subject to $t_n y(\boldsymbol{x}_n) \geq 1 - \xi_n$ and $\xi_n \geq 0$ for all n, and where $C > 0$ controls regularization. Its value can be decided by cross-validation. Note that the number of misclassified points is upper bounded by $\sum_n \xi_n$.



- To minimize the previous expression, we minimize

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

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

Support Vector Machines for Classification

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

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

$$0 = \sum_n a_n t_n$$

$$a_n = C - \mu_n$$

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

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

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

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

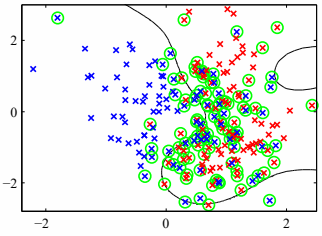
$$a_n(t_n y(\boldsymbol{x}_n) - 1 + \xi_n) = 0$$

$$\mu_n\xi_n = 0$$

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

Support Vector Machines for Classification

- Since the optimal $\boldsymbol{w}$ takes the same form as in the linearly separable case, classifying a new point is done the same as before. Finding b is done the same as before by considering any support vector $\boldsymbol{x}_n$ with $0 < a_n < C$.



- Not covered topics:
  - Classifying into more than two classes.
  - Returning class posterior probabilities.

Support Vector Machines for Regression

- Consider regressing an unidimensional continuous random variable on a D-dimensional continuous random variable.
- Consider a training set $\{(\boldsymbol{x}_n, t_n)\}$. Consider using the linear model

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

- Instead of minimizing the classical regularized error function

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

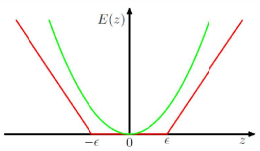
consider minimizing the ϵ-insensitive regularized error function

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

where $C > 0$ controls regularization and

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

Figure 7.6 Plot of an ϵ-insensitive error function (in red) in which the error increases linearly with distance beyond the insensitive region. Also shown for comparison is the quadratic error function (in green).



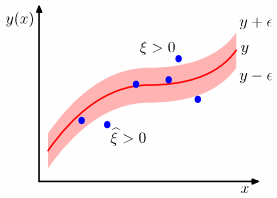
Support Vector Machines for Regression

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

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

and

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



Support Vector Machines for Regression

- The optimal regression curve is given by

$$\arg\min_{\boldsymbol{w}} C \sum_n (\xi_n + \widehat{\xi}_n) + \frac{1}{2}\|\boldsymbol{w}\|^2$$

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

- To minimize the previous expression, we minimize

$$C \sum_n (\xi_n + \widehat{\xi}_n) + \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_n (\mu_n \xi_n + \widehat{\mu}_n \widehat{\xi}_n)$$

$$- \sum_n a_n (y(\boldsymbol{x}_n) + \epsilon + \xi_n - t_n) - \sum_n \widehat{a}_n (t_n - y(\boldsymbol{x}_n) + \epsilon + \widehat{\xi}_n)$$

where $\mu_n \geq 0$, $\widehat{\mu}_n \geq 0$, $a_n \geq 0$ and $\widehat{a}_n \geq 0$ are Lagrange multipliers.

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

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

Support Vector Machines for Regression

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

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

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

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

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

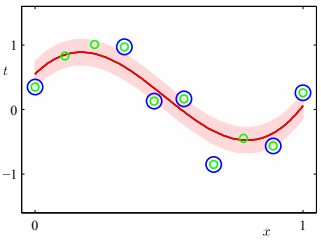
- Then, $a_n > 0$ if and only if $y(\boldsymbol{x}_n) + \epsilon + \xi_n - t_n = 0$, which implies that $\boldsymbol{x}_n$ lies on or above the upper margin of the ϵ-tube. Similarly for $\widehat{a}_n > 0$.

Support Vector Machines for Regression

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

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

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



- 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,

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

Summary

- Kernel trick: It allows to work in the feature space without constructing it.
- Quadratic objective function: It allows to obtain the global optimum for a given kernel and C/ϵ (which are obtained by cross-validation).
- Sparse model: Only the support vectors are needed for classification/regression (compare with kernel models).

732A95/TDDE01 Introduction to Machine Learning

Lecture 3c Block 2: Neural Networks

Jose M. Peña
IDA, Linköping University, Sweden

Contents

Literature

- Main source
  - Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006. Sections 5.1-5.3.3 and 5.5.3.
- Additional source
  - Hastie, T., Tibshirani, R. and Friedman, J. *The Elements of Statistical Learning*. Springer, 2009. Chapter 11.

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

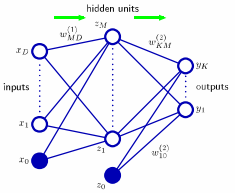
$$y(\boldsymbol{x}) = \text{sgn}\left(\sum_{m \in \mathcal{S}} a_m t_m k(\boldsymbol{x}, \boldsymbol{x}_m) + b\right)$$

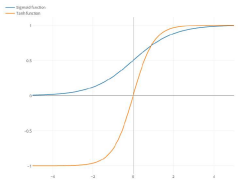
- Consider regressing an unidimensional continuous random variable on a D-dimensional continuous random variable. Consider a training set $\{(\boldsymbol{x}_n, t_n)\}$
- For a new point $\boldsymbol{x}$, SVMs predict

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

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

Neural Networks





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

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

- Evaluating the previous expression is known as forward propagation. The NN is said to have a feed-forward architecture.
- All the previous is, of course, generalizable to more layers.

Neural Networks

- 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 ?!

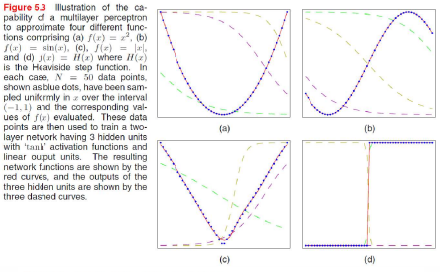


Figure 5.3 Illustration of the capability of a multilayer perceptron to approximate four different functions comprising (a) $f(x) = x^2$, (b) $f(x) = \sin(x)$, (c), $f(x) = |x|$, and (d) $f(x) = H(x)$ where $H(x)$ is the Heaviside step function. In each case, $N = 50$ data points, shown asblue dots,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

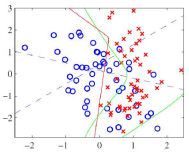
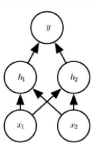
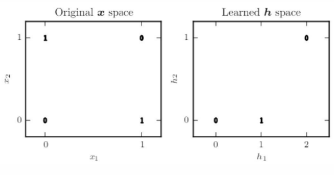


Figure 5.4 Example of the solution of a simple two-class classification problem involving synthetic data using a neural network having two inputs, two hidden units with 'tanh' activation functions, and a single output having a logistic sigmoid activation function. The dashed blue lines show the $z =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

Neural Networks

- Solving the XOR problem with NNs.
- No line shatters the points in the original space.
- The NN represents a mapping of the input space to an alternative space where a line can shatter the points. Note that the points (0,1) and (1,0) are mapped both to the point (1,0).
- It resembles SVMs.





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

Backpropagation Algorithm

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

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

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

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

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

- Sequential, stochastic or online gradient descent

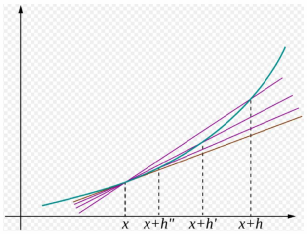
$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta_n \nabla E_n(\boldsymbol{w}^t)$$

where n is chosen randomly or sequentially.

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

Backpropagation Algorithm

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



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

Backpropagation Algorithm

- 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, then

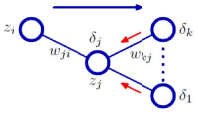
$$\delta_k = \frac{\partial E_n}{\partial a_k} = y_k - t_k$$

- 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.

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 Illustraton 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.

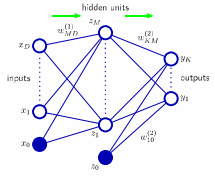


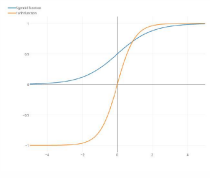
- Since $y_k = \sigma(a_k)$ for classification, then

$$\delta_k = \frac{\partial E_n}{\partial a_k} = \sigma(a_k)(1 - \sigma(a_k))$$

- This is an example of embarrassingly parallel algorithm.

Backpropagation Algorithm





- Example: $y_k = a_k$, and $z_j = h(a_j) = \tanh(a_j)$ where $\tanh(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)}$.
- Note that $h'(a) = 1 - h(a)^2$.
- Backpropagation:
  1. Forward propagation, i.e. compute
  $$a_j = \sum_i w_{ji} x_i \text{ and } z_j = h(a_j) \text{ and } y_k = \sum_j w_{kj} z_j$$
  2. Compute
  $$\delta_k = y_k - t_k$$
  3. Backpropagate, i.e. compute
  $$\delta_j = (1 - z_j^2) \sum_k w_{kj} \delta_k$$
  4. Compute
  $$\frac{\partial E_n}{\partial w_{kj}} = \delta_k z_j \text{ and } \frac{\partial E_n}{\partial w_{ji}} = \delta_j x_i$$

Backpropagation Algorithm

- The weight space is non-convex and has many symmetries, plateaus and local minima. So, the initialization of the weights in the backpropagation algorithm is crucial.
- Hints based on experimental rather than theoretical analysis:
  - Initialize the weights to different values, otherwise they would be updated in the same way because the algorithm is deterministic, and so creating redundant hidden units.
  - Initialize the weights at random, but
    - too small magnitude values may cause losing signal in the forward or backward passes, and
    - too big magnitude values may cause the activation function to saturate and lose gradient.
  - Initialize the weights according to prior knowledge: Almost-zero for hidden units that are unlikely to interact, and bigger magnitude values for the rest.
  - Initialize the weights to almost-zero values so that the initial model is almost-linear, i.e. the sigmoid function is almost-linear around the zero. Let the algorithm to introduce non-linearities where needed.
    - Note however that this initialization makes the sigmoid function take a value around half its saturation level. That is why the hyperbolic tangent function is sometimes preferred in practice.

Regularization

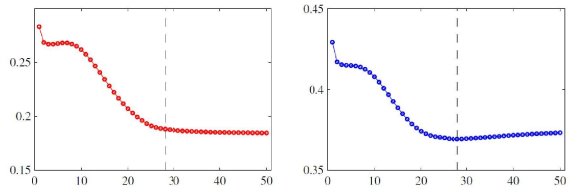


Figure 5.12 An illustration of the behaviour of training set error (left) and validation set error (right) during a typical training session, as a function of the iteration step, for the sinusoidal data set. The goal of achieving the best generalization performarce suggests that training should be stopped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

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

$$E(\boldsymbol{w}) + \frac{\lambda_1}{2}||\boldsymbol{w}^{(1)}||^2 + \frac{\lambda_2}{2}||\boldsymbol{w}^{(2)}||^2$$

    instead of the classical

$$E(\boldsymbol{w}) + \frac{\lambda}{2}||\boldsymbol{w}||^2$$

    and choose λ_1 and λ_2 by cross-validation.

Regularization

- To see why, let $z_j = h(\sum_i w_{ji} x_i + w_{j0})$ and $y_k = \sum_j w_{kj} z_j + w_{k0}$.
- Consider the linear transformation $x_i \rightarrow a x_i + b$.
- Transform $w_{j0} \rightarrow w_{j0} - \frac{b}{a}\sum_i w_{ji}$ and $w_{ji} \rightarrow \frac{1}{a} w_{ji}$.
- Both NNs define the same mapping, but they may receive different penalty for complexity $||\boldsymbol{w}||^2$.
- Solution: Penalize according to $\frac{\lambda_1}{2}||\boldsymbol{w}^{(1)}||^2 + \frac{\lambda_2}{2}||\boldsymbol{w}^{(2)}||^2$ and let $\lambda_1 \rightarrow a^{1/2}\lambda_1$.
- Finally, note that the effect of the penalty is simply to add $\lambda_1 w_{ji}$ and $\lambda_2 w_{kj}$ to the appropriate derivatives.

Summary

- NNs: Nonlinear mapping from input to output.
- Extremely expressive.
- Training: Backpropagation algorithm, and regularization.

732A95/TDDE01 Introduction to Machine Learning

Lecture 3d Block 1: Deep Learning

Jose M. Peña
IDA, Linköping University, Sweden

Contents

Literature

- Main sources
  - Bengio, Y. Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning*, 2:1-127, 2009. Chapters 1-3, 4.2, 4.5-4.6, 6.2-6.3.
  - Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006. Section 5.5.6.
  - LeCun, Y., Bengio, Y. and Hinton, G. Deep Learning. *Nature*, 521:436-44, 2015.
- Additional source
  - Goodfellow, I., Bengio, Y. and Courville, A. *Deep Learning*. Book in preparation for MIT Press, 2016. Available at `www.deeplearningbook.org`. Chapters 1, 6.

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

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

- How many hidden units has such a NN ?
- How much data do we need to learn such a NN (and avoid overfitting) via the backpropagation algorithm ?
- How fast does the backpropagation algorithm converge to such a NN ? Assuming that it does not get trapped in a local minimum...
- The answer to the last two questions depends on the first: More hidden units implies more training time and higher generalization error.

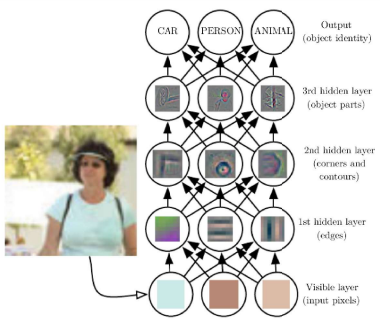
Limitations of Neural Networks

- How many hidden units does the NN need ?
- Any Boolean function can be written in disjunctive normal form (OR of ANDs) or conjunctive normal form (AND of ORs). This is a depth-two logical circuit.
- For most Boolean functions, the size of the circuit is exponential in the size of the input.
- However, there are Boolean functions that have a polynomial-size circuit of depth k and an exponential-size circuit of depth $k - 1$.
- Then, there is no universally right depth. Ideally, we should let the data determine the right depth.

Theorem (No free lunch theorem)

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

Deep Neural Networks



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

Deep Neural Networks

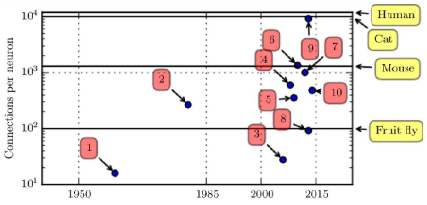


Figure 1.10: Initially, the number of connections between neurons in artificial neural networks was limited by hardware capabilities. Today, the number of connections between neurons is mostly a design consideration. Some artificial neural networks have nearly as many connections per neuron as a cat,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

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

Deep Neural Networks

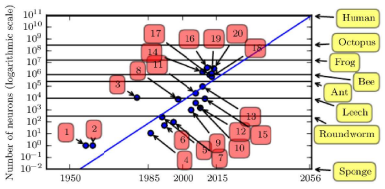


Figure 1.11: Since the introduction of hidden units, artificial neural networks have doubled in size roughly every 2.4 years. Biological neural network sizes from Wikipedia (2015).

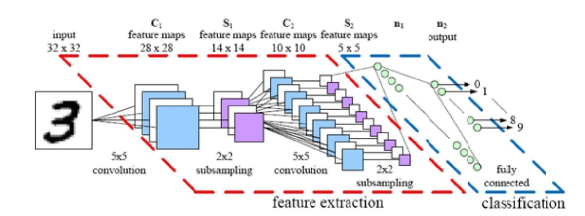
1. Perceptron (Rosenblatt, 1958 1962)
2. Adaptive linear element (Widrow and Hoff, 1960)
3. Neocognitron (Fukushima, 1980)
4. Early back-propagation network (Rumelhart et al., 1986b)
5. Recurrent neural network for speech recognition (Robinson and Fallside, 1991)
6. Multilayer perceptron for speech recognition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

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

Deep Neural Networks

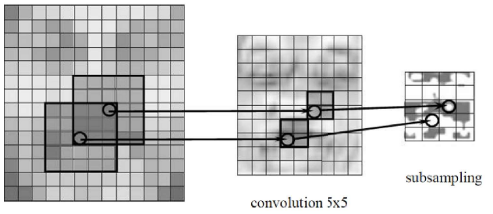
- Training DNNs is difficult:
  - Typically, poorer generalization than (shallow) NNs.
  - The gradient may vanish/explode as we move away from the output layer, due to multiplying small/big quantities. E.g. the gradient of σ and tanh is in $[0, 1)$. So, they may only suffer the gradient vanishing problem. Other activations functions may suffer the gradient exploding problem.
  - There may be larger plateaus and many more local minima than with NNs.
- Training DNNs is doable:
  - Convolutional networks, particularly suitable for image processing.
  - Rectifier activation function, a new activation function.
  - Layer-wise pre-training, to find a good starting point for training.
- In addition to performance, the computational demands of the training must be considered, e.g. CPU, GPU, memory, parallelism, etc.
  - The authors state that GoogLeNet was trained "using modest amount of model and data-parallelism. Although we used a CPU based implementation only, a rough estimate suggests that the GoogLeNet network could be trained to convergence using few high-end GPUs within a week, the main limitation being the memory usage".

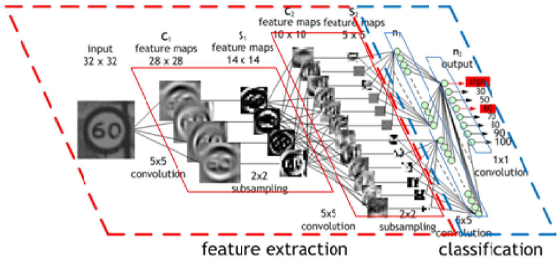
Convolutional Networks



- DNNs suitable for image recognition, since they exhibit invariance to translation, scaling, rotations, and warping.
- Convolution: Detection of local features, e.g. a_j is computed from a 5x5 pixel patch of the image.
- To achieve invariance, the units in the convolution layer share the same activation function and weights.
- Subsampling: Combination of local features into higher-order features, e.g. a_k is compute from a 2x2 pixel patch of the convoluted image.
- There are several feature maps in each layer, to compensate the reduction in resolution by increasing in the number of features being detected.
- The final layer is a regular NN for classification.

Convolutional Networks





Convolutional Networks

- DNNs allow increased depth because
  - they are sparse, which allows the gradient to propagate further, and
  - they have relatively few weights to fit due to feature locality and weight sharing.
- The backpropagation algorithm needs to be adapted, by modifying the derivatives with respect to the weights in each convolution layer m.
- Since E_n depends on $w_i^{(m)}$ only via $a_j^{(m)}$, and $a_j^{(m)} = \sum_{i \in L_j^{(m)}} w_i^{(m)} z_i^{(m-1)}$ where $L_j^{(m)}$ is the set of indexes of the input units, then

$$\frac{\partial E_n}{\partial w_i^{(m)}} = \sum_j \frac{\partial E_n}{\partial a_j^{(m)}} \frac{\partial a_j^{(m)}}{\partial w_i^{(m)}} = \sum_j \delta_j^{(m)} z_i^{(m-1)}$$

- Note that $w_i^{(m)}$ does not depend on j by weight sharing, whereas $i \in L_j^{(m)}$ by feature locality.

Rectifier Activation Function

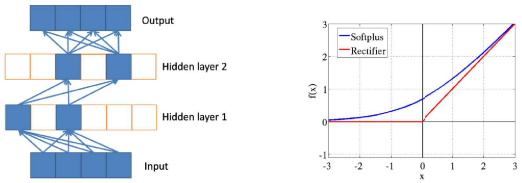


Figure 2: *Left:* **Sparse propagation of activations and gradients in a network of rectifier units.** The input selects a subset of active neurons and computation is linear in this subset. *Right:* **Rectifier and softplus activation functions.** The second one is a smooth version of the first.

- $rectifier(x) = \max\{0, x\}$, i.e. hidden units are off or operating in a linear regime.
- The most popular choice nowadays.
- Sparsity promoting: Uniform initialization of the weights implies that around 50 % of the hidden units are off.
- Piece-wise linear mapping: The input selects which hidden units are active, and the output is a liner function of the input in the selected hidden units.

Rectifier Activation Function

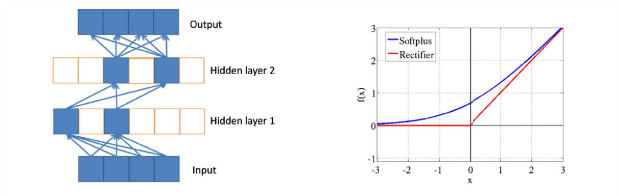


Figure 2: *Left:* **Sparse propagation of activations and gradients in a network of rectifier units.** The input selects a subset of active neurons and computation is linear in this subset. *Right:* **Rectifier and softplus activation functions.** The second one is a smooth version of the first.

- It simplifies the backpropagation algorithm as $h'(a_j) = 1$ for the selected units. So, there is no gradient vanishing on the paths of selected units.
- Note that $h'(0)$ does not exist since $h'_+(0) \neq h'_-(0)$. We can get around this problem by simply returning one of two one-sided derivatives. Or using a generalization of the rectifier function.
- Regularization is typically added to prevent numerical problems due to the activation being unbounded, e.g. when forward propagating.

Layer-Wise Pre-Training

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

Summary

- Direct application of the backpropagation algorithm to DNNs produces poor results.
- Convolutional networks: It makes the backpropagation algorithm more efficient by using local features and weight sharing. This also achieves invariance, which is particularly important for image processing.
- Rectifier activation function: Free of gradient vanishing problem and it simplifies the backpropagation algorithm.
- Layer-wise pre-training: Heuristic weight initialization to alleviate the local optima problem.

Lab 1 Correction notes

I had used the row instead of column in the distance matrix, giving me okay classification rates but a bad ROC curve. After that adjustment it looks much better. Especially the ROC curve which is now above the diagonal, instead of below. My knn implementation even gets better rates that the kknn algorithm.

Lab 1 Assignment 1

Question 3

knearest(train, 5, test)

Misclassification rate: 31.8%

	Actual spam	Actual not spam
Predicted spam	240	242
Predicted not spam	193	695

knearest(train, 5, train)

Misclassification rate: 20.2%

	Actual spam	Actual not spam
Predicted spam	306	158
Predicted not spam	119	787

Question 4

For K=1, comparing with the training data is way better than comparing with the test data. This is as I expected, since I guess that it compares almost each point with itself.

When comparing with test data, its better with K=5 than with K=1. I believe that it's because of that K=5 gives a better average than K=1.

knearest(train, 1, test)

Misclassification rate: 34.7%

	Actual spam	Actual not spam

Predicted spam	255	298
Predicted not spam	178	639

knearest(train, 1, train)

Misclassification rate: 0.6%

	Actual spam	Actual not spam
Predicted spam	423	6
Predicted not spam	2	939

Question 5

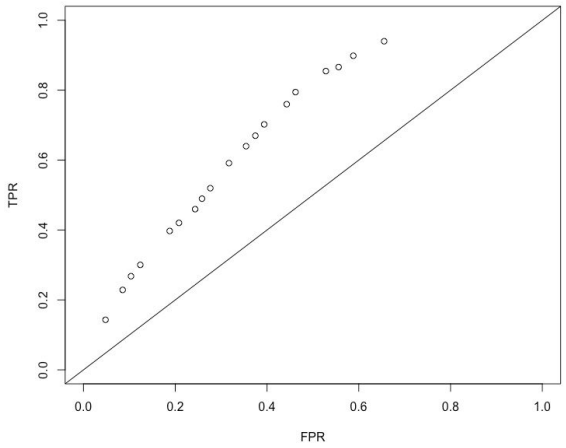
Since the misclassification rate is higher, the kknn-library does a worse jobb of classifying than my knn algorithm.

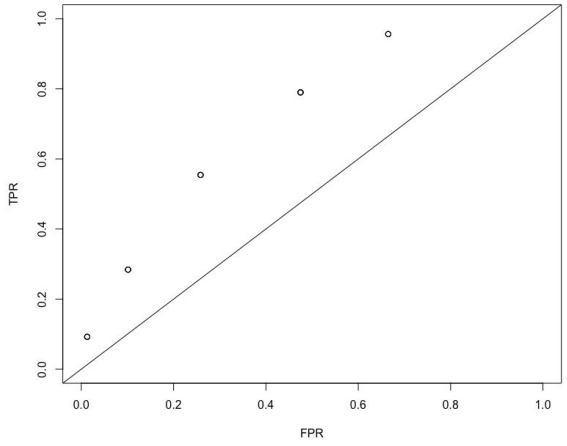
Misclassification rate: 34.6%

	Actual spam	Actual not spam
Predicted spam	256	297
Predicted not spam	177	640

Question 6

The left image is kknn and the right one is my implementation of knn. The kknn package does a worse job than my implementation.

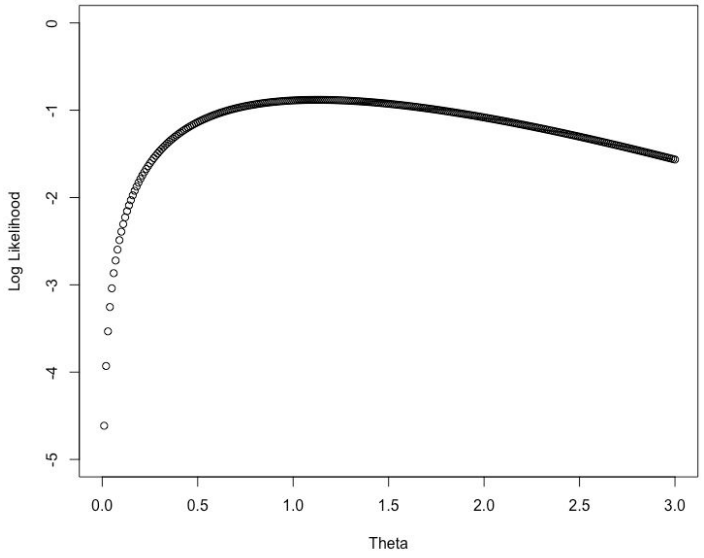




Lab 1 Assignment 2

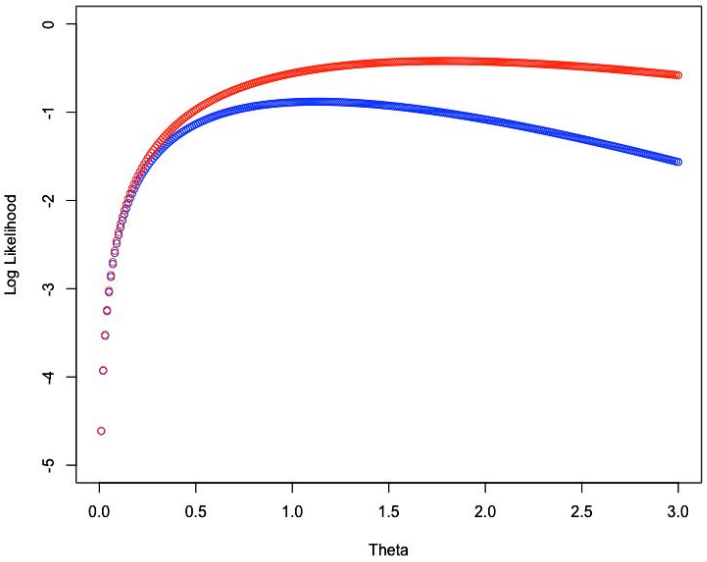
Question 2

Theta on maximum likelihood is 1.13



Question 3

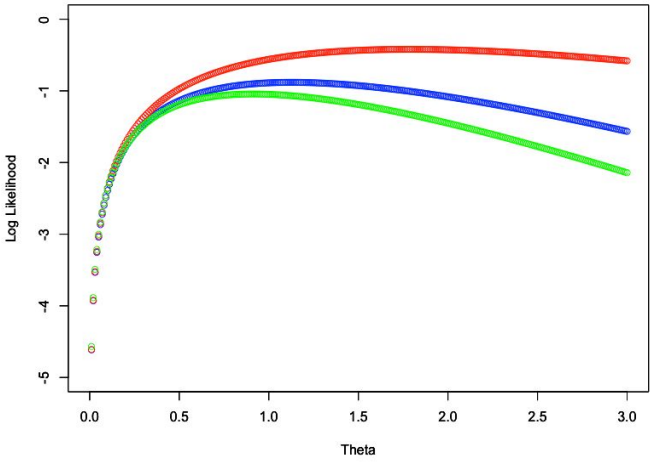
The blue curve contains all the samples. The red one only has the first six samples from the same data set. This curve has its maximum likelihood at theta = 1.79. The solution with all samples is more reliable.



Question 4

Heres the same graph with the bayesian model added(in green). It show the log likelihood when the previous, previous value was 10. The maximum likelihood is found at theta = 0.91. This

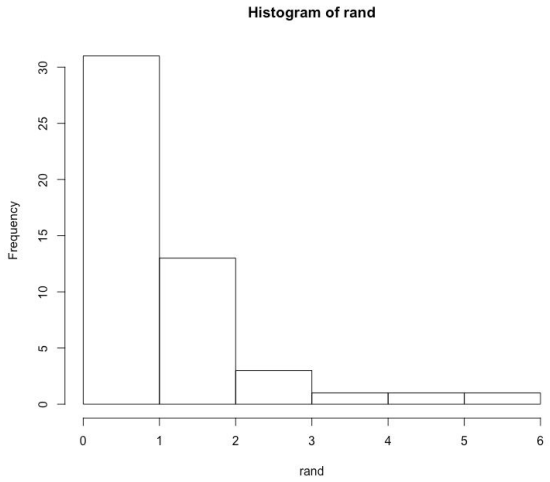
curve reaches its peak faster and then decreases faster, which I would interpret as that its easier to determine a “best” value, the peak better defined.

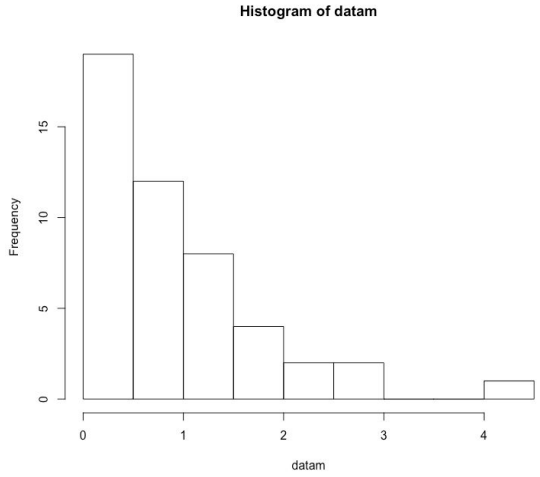


Question 5

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.

I used set.seed(12345) to make it replicable.





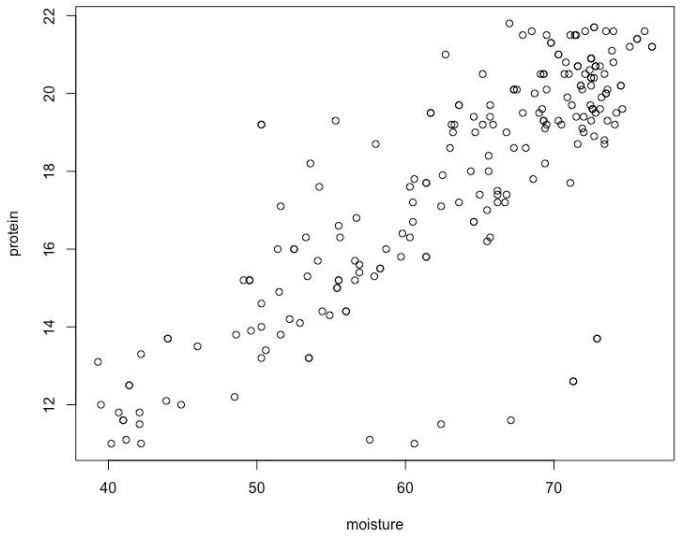
Lab 1 Assignment 4

Correction Notes

Changed probabilistic model for the data and added a sequence, between 0 and 1 with step=0.001, for variable lambda in question 7.

Question 1

I believe that the moisture-protein relation could be represented by a linear model.



Question 2 & 3

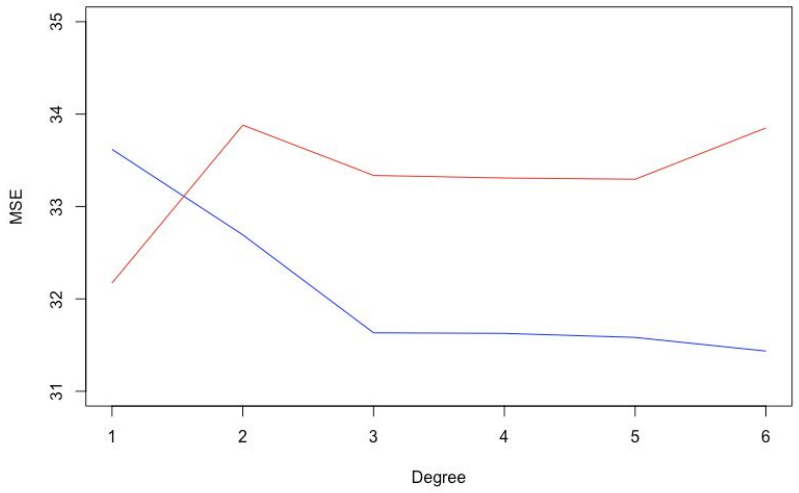
The data could be represented by a normal distribution with a polynomial function and some deviation:

$$Y \sim N(\beta_0 + \beta_1 x + ... + \beta_n x^n, \sigma^2)$$

MSE calculates error between two given data sets and can be used to compare several data sets, a predicted data set and a validation data set for example.

The blue curve is the MSE of prediction with training data and the red curve is with test data.

A lower degree seems to be better in this case. For the training data the MSE decreases with higher degrees, which is because the model becomes overfitted. For the test data however, MSE is lowest for degree=1, and worse for higher degrees for the same reason. The model has a stronger bias towards the training data in higher degrees.



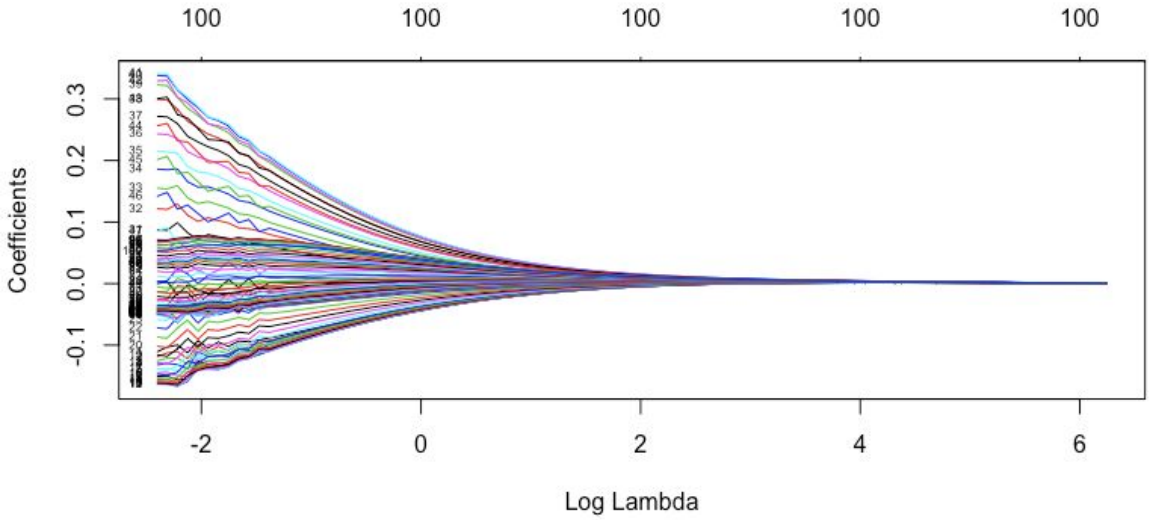
Question 4

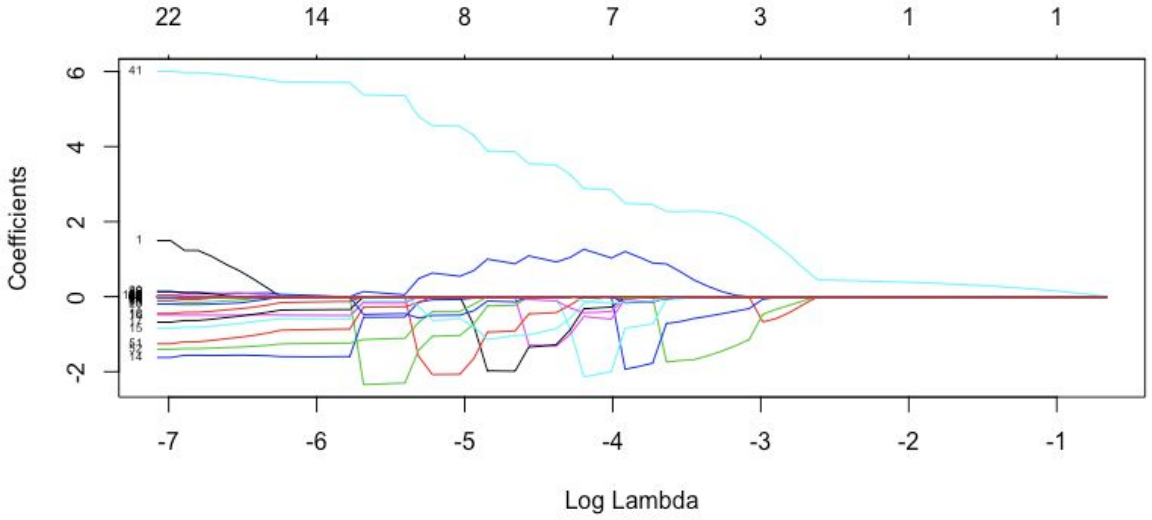
stepAIC() selected 63 variables:

```
F-statistic: 447.9 on 63 and 151 DF,  p-value: < 2.2e-16
```

Question 5 & 6

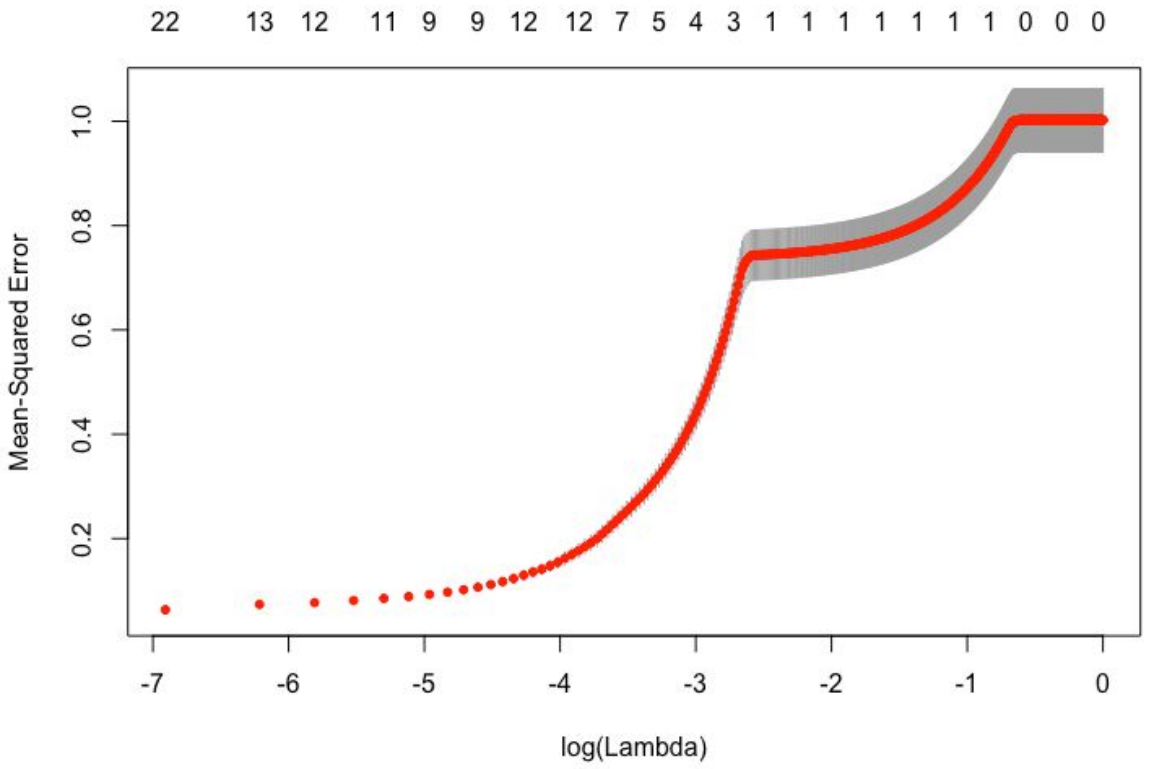
The LASSO model gives more weight to fewer variables than the ridge regression model.
For LASSO, channel 41 seems to be very important.





Question 7

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



Lab 1 Appendix

Assignment 1

KNN

```
library(readxl)
data <- read_excel("TDDE01/lab1/assignment1/spambase.xlsx")

#divide data into training and test
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]

knearest=function(data,k,newdata) {
  n1=dim(data)[1]
  n2=dim(newdata)[1]
  p=dim(data)[2]
  Prob=numeric(n2)
```

```
  X=as.matrix(data[,-p])
  Xclass=as.matrix(data[,p])
  X=X/matrix(sqrt(rowSums(X^2)), nrow=n1, ncol=p-1)

  Y=as.matrix(newdata[,-p])
  Yclass=as.matrix(newdata[,p])
  Y=Y/matrix(sqrt(rowSums(Y^2)), nrow=n1, ncol=p-1)

  class = numeric(n2)

  C <- X %*% t(Y)
  D <- 1-C

  for (i in 1:n2 ){
    Drow <- D[,i, drop=F]
    Drow <- cbind(Drow,Xclass)
    Drow <- Drow[order(Drow[,1]),]
    Drow <- head(Drow,k)

    #classifies
    class[i] <- if(sum(Drow[,2] > 0)/k > 0.5) 1 else 0;
  }

  CM <- table(factor(Yclass, labels=c("Actual not spam", "Actual spam")),
              factor(class, labels=c("Pred not spam", "Pred spam")));

  #print classification rate and CM
  print(1-(CM[1,1]+CM[2,2])/sum(CM))
  print(CM)
}

knearest(train, 1, train)
knearest(train, 5, train)
knearest(train, 1, test)
knearest(train, 5, test)
```

KNN ROC

```
#imports the excel-file
library(readxl)
data <- read_excel("TDDE01/lab1/assignment1/spambase.xlsx")

#divide data into training and test
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]

knearest_multik=function(data,t,k,newdata) {
  n1=dim(data)[1]
  n2=dim(newdata)[1]
  p=dim(data)[2]
```

```
  Prob=numeric(n2)

  X=as.matrix(data[,-p])
  Xclass=as.matrix(data[,p])
  X=X/matrix(sqrt(rowSums(X^2)), nrow=n1, ncol=p-1)

  Y=as.matrix(newdata[,-p])
  Yclass=as.matrix(newdata[,p])
  Y=Y/matrix(sqrt(rowSums(Y^2)), nrow=n1, ncol=p-1)

  TPR=numeric(length(t))
  FPR=numeric(length(t))

  C <- X %*% t(Y)
  D <- 1-C

  for(j in 1:length(t)){
    class <- rep(0,n2)

    for (i in 1:n2 ){
      Drow <- D[,i, drop=F]
      Drow <- cbind(Drow,Xclass)
      Drow <- Drow[order(Drow[,1]),]
      Drow <- head(Drow,k)

      class[i] <- if(sum(Drow[,2])/k > t[j]) 1 else 0;
    }
    CM <- table(factor(Yclass, labels=c("Actual not spam", "Actual spam")),
                factor(class, labels=c("Pred not spam", "Pred spam")));

    TPR[j] = (CM[2,2] / (CM[2,2] + CM[2,1]))
    FPR[j] = (CM[1,2] / (CM[1,2] + CM[1,1]))
  }

  plot(FPR, TPR, xlim=c(0,1), ylim=c(0,1))
  abline(a=0,b=1)

  return(CM)
}

p <- seq(from=0.05, to=0.95, by=0.05)

knearest_multik(train, p, 5, test)
```

KKNN + ROC

```
library(kknn)

ROC=function(Y, Yfit, p) {
  m=length(p)
  TPR=numeric(m)
  FPR=numeric(m)
```

```
  for(i in 1:m) {
    t=table(Yfit>p[i], Y)

    TP = t[2,2]
    FP = t[2,1]
    TN = t[1,1]
    FN = t[1,2]

    TPR[i]=TP/(TP+FN)
    FPR[i]=FP/(FP+TN)
  }
  plot(FPR, TPR, xlim=c(0,1), ylim=c(0,1))
  abline(a=0,b=1)
}

p <- seq(from=0.05, to=0.95, by=0.05)

model <- kknn(Spam ~ ., train, test, k = 5)
fit <- fitted(model)
result <- floor(fit + 0.5)

ROC(test$Spam, fit, p)

CM <- table(factor(test$Spam, labels=c("Actual not spam", "Actual spam")),
            factor(result, labels=c("Pred not spam", "Pred spam")));

print(CM)
```

Assignment 2

```
library(readxl)
data <- read_excel("TDDE01/lab1/machines.xlsx")

theta <- seq(from=0.00, to=3, by=0.01)

loglik <- function(x, theta) {
  ll <- numeric(length(theta))

  for(i in 1:(length(theta))){
    p <- theta[i]*exp((-theta[i]*x))
    ll[i] <- (log(prod(p))/dim(x)[1])
  }

  return(list(LogLik=ll, theta=theta))
}

loglik_b <- function(x, theta) {
  ll <- numeric(length(theta))

  for(i in 1:(length(theta))){
    p <- theta[i]*exp((-theta[i]*x))
    pre <- 10*exp(1)**(-theta[i]*10)
```

```
    ll[i] <- (log(prod(p)*pre)/dim(x)[1])
  }

  return(list(LogLik=ll, theta=theta))
}

result.allData <- loglik(data, theta)
result.firstSix <- loglik(head(data,6), theta)
result.bayesian <- loglik_b(data, theta)

rand <- rexp(50,rate=1.13)
#View(rand)

#set.seed(12345)
#hist(rand)
#datam <- as.matrix(data)
#hist(datam)

plot(result.allData$theta, result.allData$LogLik, xlab="Theta", ylab="Log Likelihood",
xlim=c(0,3), ylim=c(-5,0), col="blue")
points(result.firstSix$theta, result.firstSix$LogLik, col="red")
points(result.bayesian$theta, result.bayesian$LogLik, col="green")

#View(result.allData)
#View(result.firstSix)
#View(cbind(result.bayesian$LogLik, result.bayesian$theta))
```

Assignment 4

```
library(readxl)
library(glmnet)
data <- read_excel("TDDE01/lab1/tecator.xlsx", header=F)

#step
fit <- lm(Fat ~., data=data[,2:102])
step <- stepAIC(fit, direction="both")
step$anova
summary(step)

#ridge & lasso
covariates=scale(data[,2:101])
response=scale(data[,102])

model.ridge=glmnet(as.matrix(covariates),response, alpha=0)
model.lasso=glmnet(as.matrix(covariates),response, alpha=1)

par(mfrow=c(2,1))
plot(model.ridge, xvar="lambda", label=T)
plot(model.lasso, xvar="lambda", label=T)
par(mfrow=c(1,1))

#lambdas
```

```
lasso.model=cv.glmnet(as.matrix(covariates), response, alpha=1, lambda = seq(0,1,0.001))
View(lasso.model)
print(lasso.model$lambda.min)
coef(lasso.model, s="lambda.min")

plot(lasso.model)

library(readxl)
data <- read_excel("TDDE01/lab1/tecator.xlsx")

#MSE for poly models
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))

train=data[id,]
test=data[-id,]

#lengths were not equal
test=test[1:dim(train)[1],]

for(i in 1:6) {
  model <- lm(Moisture ~ poly(Protein, i), data=train)

  MSEtrain[i] <- mean((train$Moisture - predict(model, train))**2)
  MSEtest[i] <- mean((test$Moisture - predict(model, test))**2)
}

plot(MSEtrain, ylab="MSE", xlab="Degree", ylim=c(31,35), col="blue", type="l")
lines(MSEtest, col="red")
```

Lab 2 Assignment 2

Decision trees

Fitted decision trees with deviance and gini index as measurements of impurity.

Missclassification rates:

- Gini index
    - Training: 19.4%
    - Test: 32.0%
- Deviance
    - Training: 17.2%
    - Test: 25.2%

training gini index	pred bad	pred good
actual bad	74	64
actual good	33	331

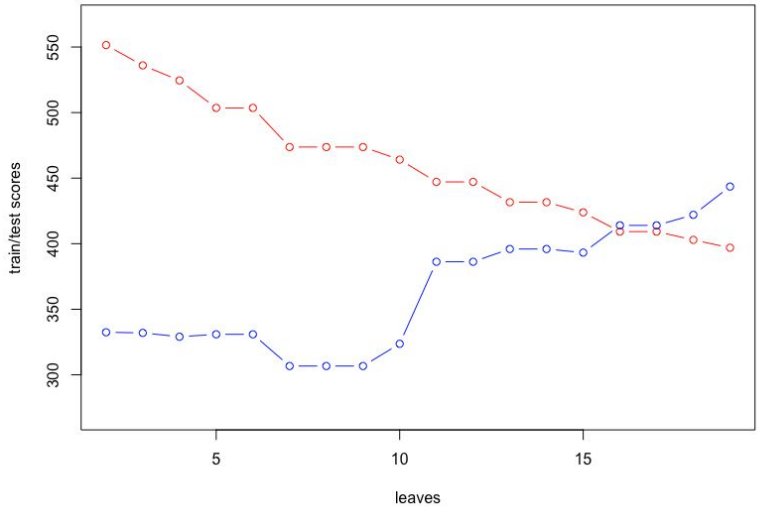
training deviance	pred bad	pred good
actual bad	77	59
actual good	27	337

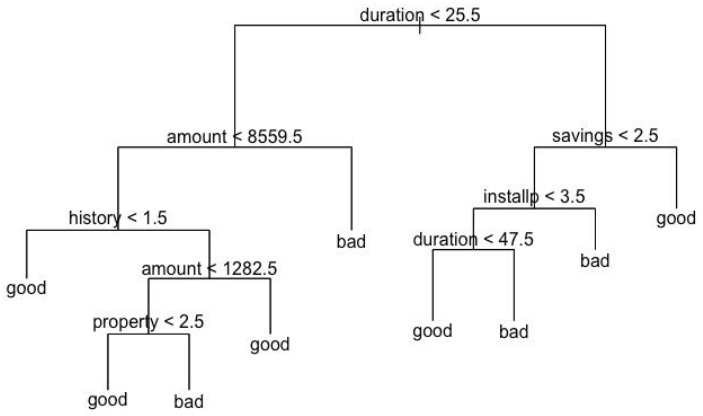
test gini index	pred bad	pred good
actual bad	23	54
actual good	26	147

test deviance	pred bad	pred good
actual bad	29	48
actual good	15	158

I will use **deviance** as impurity measurement since it gives better classification rates.

Here is a graph displaying the scores of train and test, against different amount of leaves.
Train is red and test is blue. As expected train always becomes better due to overfitting.
I would say that its optimal around 7-9 leaves. Below that is the 8-leaf-tree.





Naïve Bayes

Misclassification rates:

- Training: 26.6%
- Test: 29.6%

training	pred bad	pred good
actual bad	74	62
actual good	71	293

test	pred bad	pred good
actual bad	40	37
actual good	37	136

When using training data and comparing with the results from the decision tree, Naïve Bayes seems to be a worse choice because it has higher misclassification rates. An inpection of the confusion matrices shows that Naïve Bayes is worse at everything.

With loss matrix

Misclassification rates:

- Training: 51.6%
- Test: 46.4%

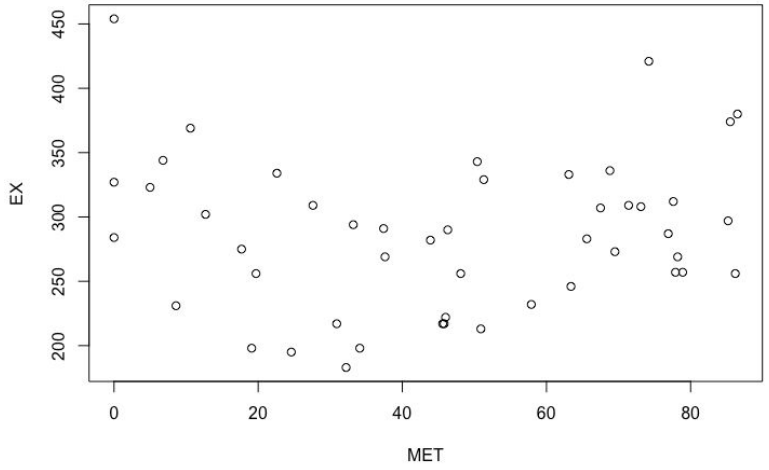
training	pred bad	pred good
actual bad	121	15

actual good	243	121

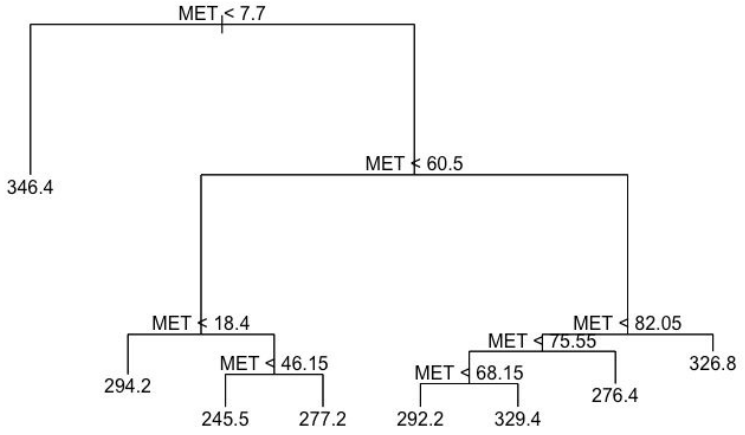
test	pred bad	pred good
actual bad	69	8
actual good	108	64

Lab 2 Assignment 3

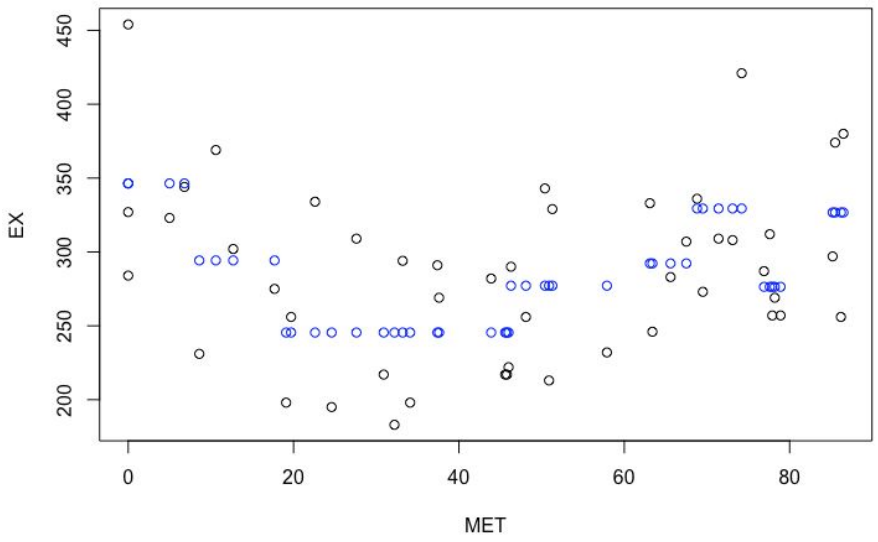
I think that the data could be respresented by a polynomial, probably quadratic equation with some distribution. I think its hard too read the distribution from this data set but I would probably try a normal distribution.



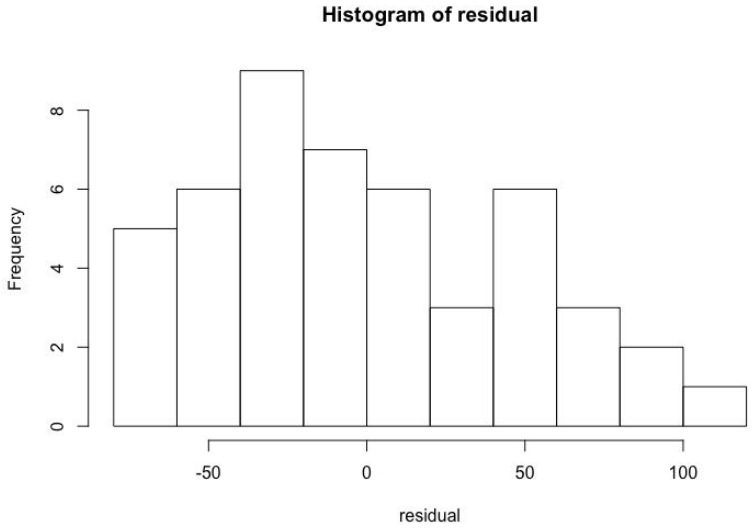
This is the selected tree. It has 8 leafs and depth 6.



The following plot contains the original data in black and the fitted data points in blue. The quality looks fairly good.

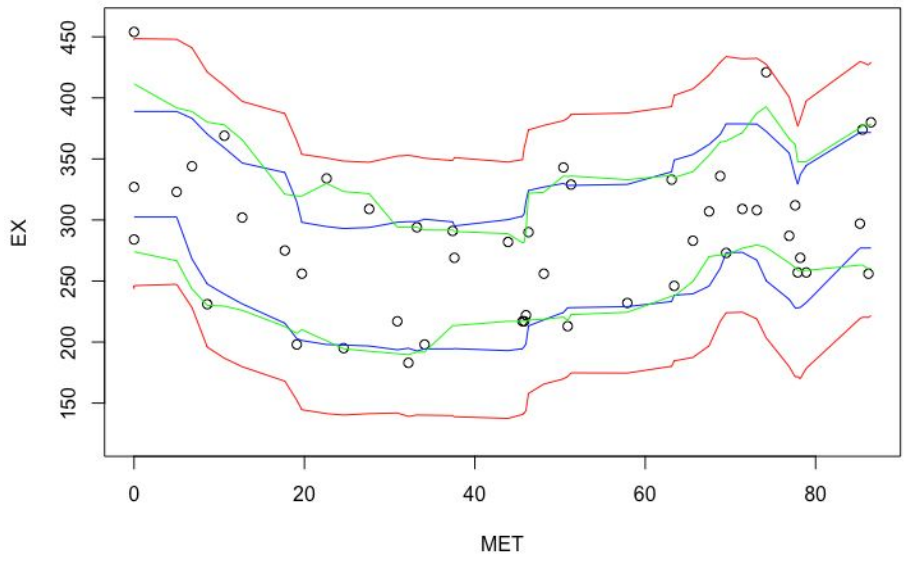


The histogram of residuals looks like a log-normal or gamma distribution.



In the following graph, green is the 95% confidence interval with non-parametric boostrap. The others are parametric bootstrap, blue is the 95% confidence interval and red is prediction interval.

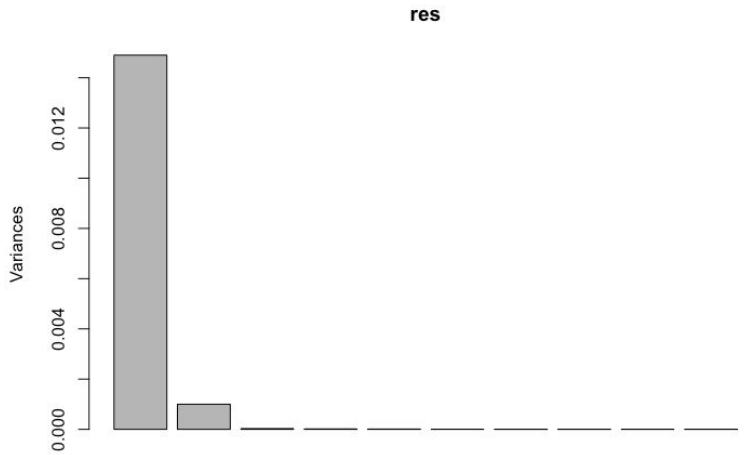
Only one point is outside the prediction band, i.e. more than 95% is inside the band and it should be like that.



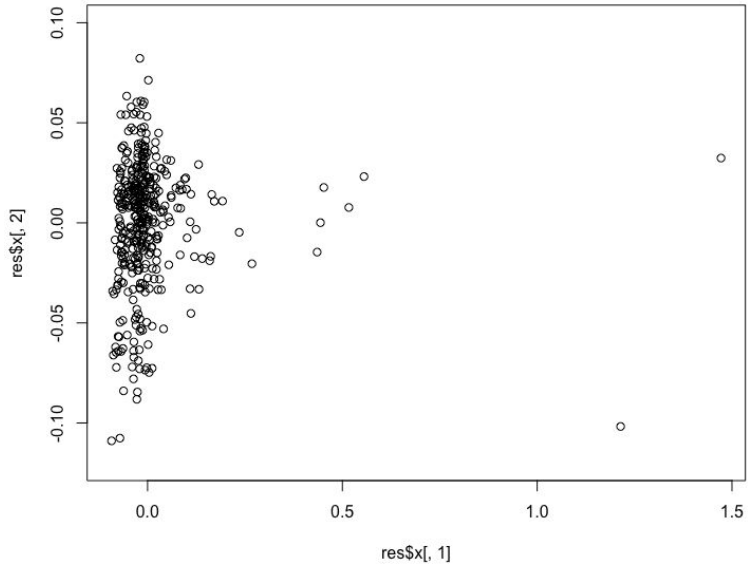
The histogram looks like a log-normal or gamma distribution, thus a parametric boostrap with such a distribution should be more appropriate.

Lab 2 Assignment 4

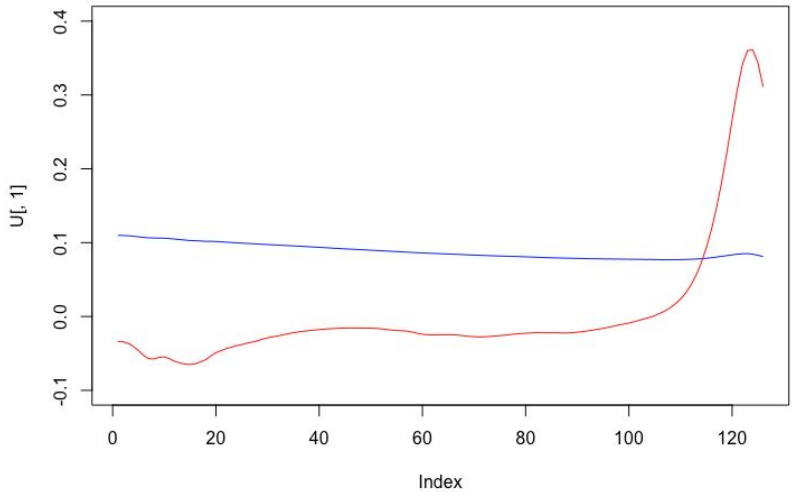
Through PCA I found that two variables explain 99% of the variance. In the plot, its easy to see that two variables are significant, but not their exact percentage. However, it could be easily read from the result vector.



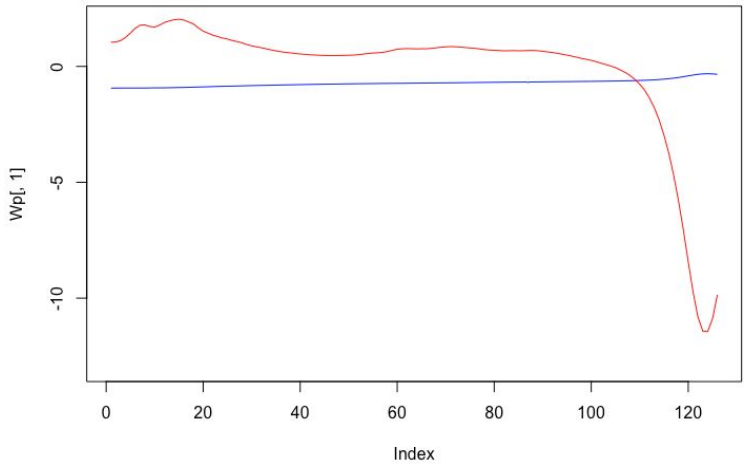
In the plot below, most data points are clustered, with a few outliers. They could be unusual diesel fuels or the result of bad reads.



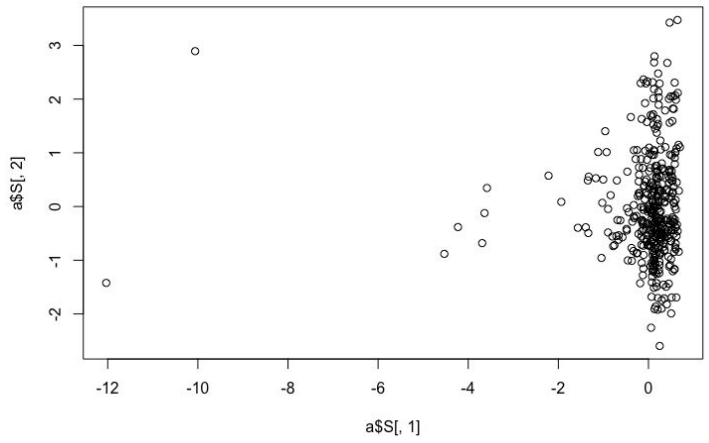
The next plot shows how the principal components are correlated to features. PC1 seems to be fairly constant while PC2 depends mostly on higher index features.



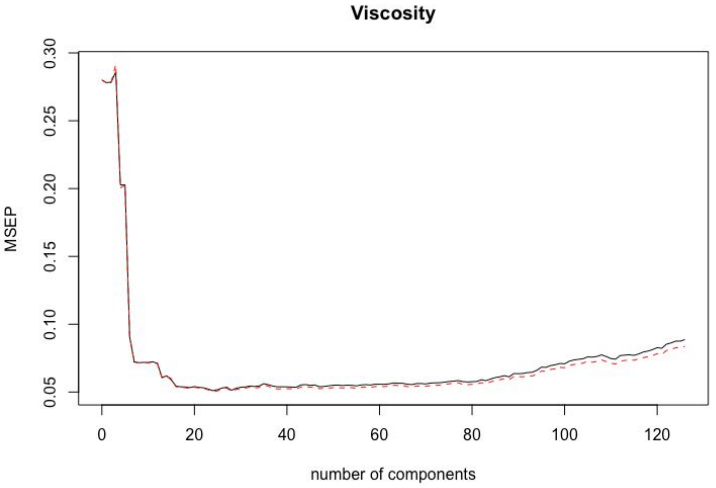
FastICA shows a very similar plot to PCA, but mirrored and amplified. The matrix W' shows how depentant a component is on a given feature.



The score-plot also seems to be mirrored and amplified.



The error reaches somewhat of a minimum around 20 components, which would be a reasonable amount to select.



Lab 2 Appendix

Assignment 2

```
library(e1071)
    library(readxl)
    library(tree)

    data = read_excel("TDDE01/lab2/assignment2/creditscoring.xls")
    data$good_bad = as.factor(data$good_bad)
    n = dim(data)[1]

    # splitting
    training = data[1:floor((1/2)*n),]
    validation = data[(floor((1/2)*n)+1):(floor((3/4)*n)),]
    test = data[(floor((3/4)*n)+1):n,]

    mysum = function(name, comp, pred) {
      print(name)
      tab = table(comp, pred)
      print(tab)
      print(1-sum(diag(2)*tab)/sum(tab))
    }

    q2 = function() {
      set.seed(12345)
      # fitting
      fit.gini = tree(good_bad ~ ., data = training, split = c("gini"))
      fit.dev = tree(good_bad ~ ., data = training, split = c("deviance"))

      # predictions
      pred.train.gini = predict(fit.gini, newdata=training, type="class")
      pred.train.dev = predict(fit.dev, newdata=training, type="class")

      pred.test.gini = predict(fit.gini, newdata=test, type="class")
      pred.test.dev = predict(fit.dev, newdata=test, type="class")

      # tables and misclassification
      mysum("train.gini", training$good_bad, pred.train.gini)
      mysum("train.dev", training$good_bad, pred.train.dev)

      mysum("test.gini", test$good_bad, pred.test.gini)
      mysum("test.dev", test$good_bad, pred.test.dev)

      fit.dev

      summary(fit.dev)

      # finding optimal size
      n_leaves = 19
```

```
  trainScore=rep(0,n_leaves)
  testScore=rep(0,n_leaves)

  for(i in 2:n_leaves) {
    prunedTree = prune.tree(fit.dev, best = i)
    pred = predict(prunedTree, newdata=validation, type="tree")
    trainScore[i] = deviance(prunedTree)
    testScore[i] = deviance(pred)
  }

  plot(2:n_leaves, trainScore[2:n_leaves], type="b", col="red", ylim=c(270,570),
ylab="train/test scores", xlab="leaves")
  points(2:n_leaves, testScore[2:n_leaves], type="b", col="blue")

  # plot of optimal tree
  pruned = prune.tree(fit.dev, best = 8)
  plot(pruned)
  text(pruned, pretty = 0)
}

q3 = function() {
  # fitting
  fit = naiveBayes(good_bad ~ ., data = training)

  # predictions
  pred.train = predict(fit, newdata = training)
  pred.test = predict(fit, newdata = test)

  # raw predictions
  pred2.train = predict(fit, newdata = training, type = "raw")
  pred2.test = predict(fit, newdata = test, type = "raw")

  # with a lower threshold
  res.train = apply(as.matrix(pred2.train[,1]), 1, function(x) ifelse( x < 0.1,
"good", "bad"))
  res.test = apply(as.matrix(pred2.test[,1]), 1, function(x) ifelse( x < 0.1,
"good", "bad"))

  # confusion matrices
  mysum("training", training$good_bad, pred.train)
  mysum("test", test$good_bad, pred.test)

  # confusion matrices with loss matrix
  mysum("training lm", training$good_bad, res.train)
  mysum("test lm", test$good_bad, res.test)
}
```

Assignment 3

```
library(tree)
    library(boot)

    data = read.csv2(file = "TDDE01/lab2/assignment3/State.csv", header=T, sep=";")
    data$EX = as.numeric(data$EX)
    data$MET = as.numeric(data$MET)
    data = data[order(data$MET),]

    set.seed(12345)

    mle = tree(EX ~ MET, data, control = tree.control(48, minsize = 8))

    f = function(data2, ind) {
      data1 = data2[ind, ]
      fit = tree(EX ~ MET, data1, control = tree.control(48, minsize = 8))
      pred = predict(fit, newdata = data)
      return(pred)
    }

    f1 = function(data1) {
      fit = tree(EX ~ MET, data1, control = tree.control(48, minsize = 8))
      pred = predict(fit, newdata = data)
      return(pred)
    }

    f2 = function(data1) {
      fit = tree(EX ~ MET, data1, control = tree.control(48, minsize = 8))
      pred = predict(fit, newdata = data)
      n = length(data$EX)
      predd = rnorm(n, pred, sd(resid(mle)))
      return(predd)
    }

    rng = function(data, mle) {
      data1 = data.frame(EX = data$EX, MET = data$MET)
      n = length(data$EX)
      data1$EX = rnorm(n, predict(mle, newdata=data1), sd(resid(mle)))
      return(data1)
    }

    q2 = function() {
      plot(data$MET, data$EX, ylab="EX", xlab="MET")

      fit = tree(EX ~ MET, data, control = tree.control(48, minsize = 8))
      fitted = predict(fit, newdata=data)
      plot(fit)
      View(fit)
      text(fit, pretty=0)

      points(data$MET, fitted, col="BLUE")
```

```
  residual = resid(fit)
  hist(residual)
}

# parametric
q4 = function() {
  mle = tree(EX ~ MET, data, control = tree.control(48, minsize = 8))

  res = boot(data, statistic = f1, R = 1000, mle=mle, ran.gen = rng,
sim="parametric")
  e = envelope(res)

  plot(data$MET, data$EX, ylab="EX", xlab="MET", ylim=c(120,460))
  lines(data$MET, e$point[2,], col="blue")
  lines(data$MET, e$point[1,], col="blue")

  res2 = boot(data, statistic = f2, R = 10000, mle=mle, ran.gen = rng,
sim="parametric")
  e2 = envelope(res2)

  lines(data$MET, e2$point[2,], col="red")
  lines(data$MET, e2$point[1,], col="red")

}

# non-parametric
q3 = function() {
  res = boot(data, f, R=1000)
  e = envelope(res)

  plot(data$MET, data$EX, ylab="EX", xlab="MET")
  lines(data$MET, e$point[2,], col="green")
  lines(data$MET, e$point[1,], col="green")
}
```

Assignment 4

```
library(fastICA)
    library(pls)
    data = read.csv2(file = "TDDE01/lab2/assignment4/NIRSpectra.csv", header=T,
    sep=";")

    Viscosity = data$Viscosity
    data$Viscosity = c()
    res = prcomp(data)
    set.seed(12345)

    screeplot(res)

    # PCA analysis
    q1 = function() {
      lambda = res$sdev^2
      lambda

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

      U = res$rotation
      head(U)

      plot(res$x[,1], res$x[,2], ylim=c(-0.12,0.10))
    }

    # trace plots
    q2 = function() {
      U = res$rotation
      plot(U[,1], ylim=c(-0.1,0.4), col="blue", type="l")
      lines(U[,2], col="red")
    }

    # ICA
    q3 = function() {
      a = fastICA(data, 2, alg.typ="parallel", fun="logcosh", alpha=1, method="R",
    row.norm=F, maxit=200, tol=0.0001, verbose=T)
      Wp = a$K %*% a$W

      plot(Wp[,1], ylim=c(-13,2), col="blue", type="l")
      lines(Wp[,2], col="red")

      plot(a$S[,1], a$S[,2])
    }

    # PCR
    q4 = function() {
      pcr.fit = pcr(Viscosity ~ ., data=data, validation = "CV")
      summary(pcr.fit)
      validationplot(pcr.fit, val.type="MSEP")
    }
```

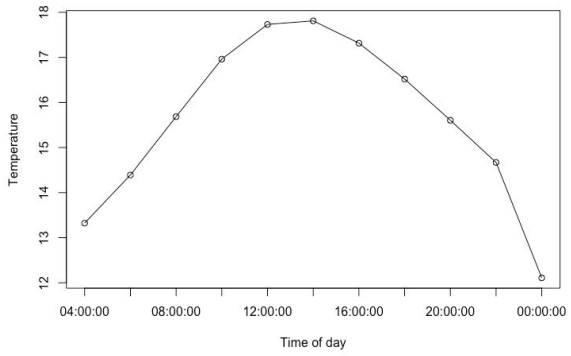
Lab 3 Part 1: Kernel Methods

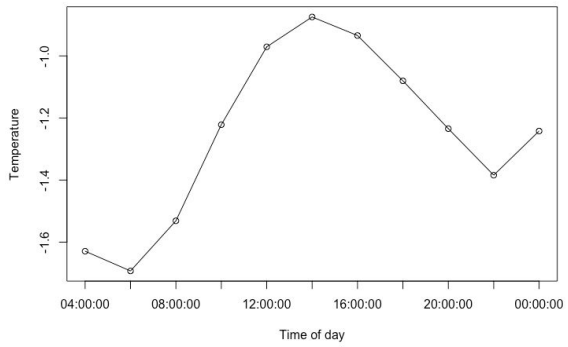
At first when trying with just adding the kernels together my graph does not show accurate values for the temperature, however when multiplying the the graph looks reasonable. I think that is since closeness in one or two kernels can give a high value when adding them, but when multiplying them a low value kernel with punish the product of all kernels.

My smoothing values:
h_distance <- 200*1000
h_date <- 20
h_time <- 4

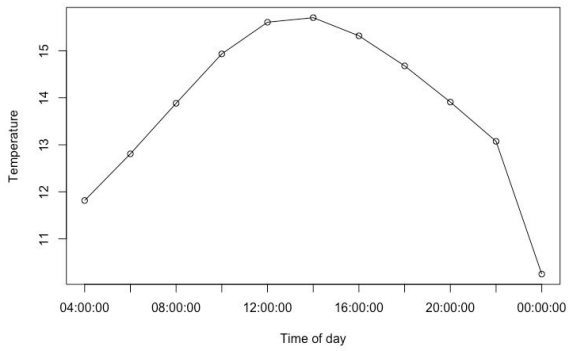
I found them by looking at the distances between data points and my target values, I reasoned around how much the data points importance should drop due to the distances in time, physical distance and date. I generated graphs for two locations which I think shows reasonable temperatures for those locations at the chosen dates. Also the temperature seems to rise/decline steeper during the winter, which I guess is because of the shorter day, it's even steeper for Umeå, which is further north and has an even shorter day during winter time.

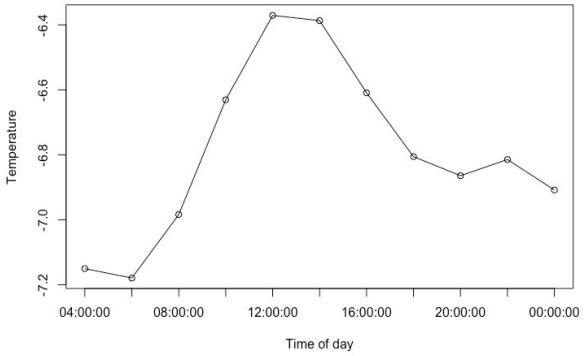
Plot for Vadstena (14.826, 58.4274) 2013-07-14 and 2013-01-04





Plot for Umeå (20.1534, 63.4942) 2013-07-14 and 2013-01-04

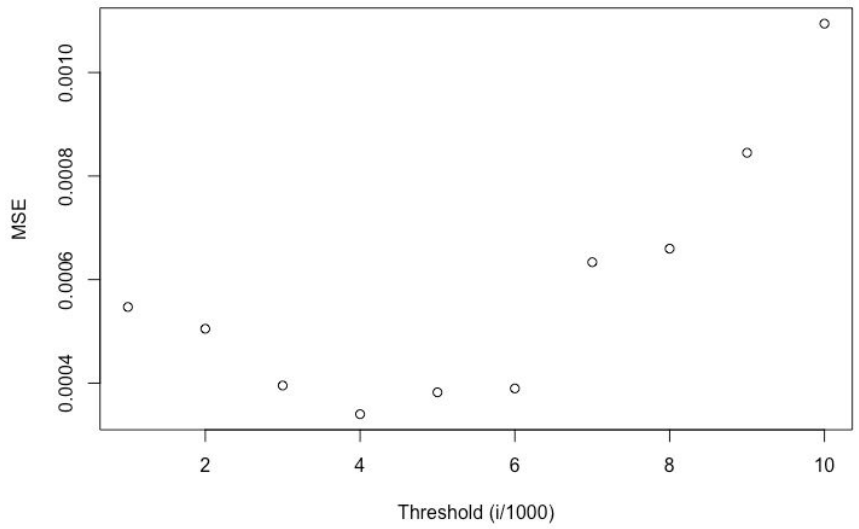




Lab 3 Part 2: Neural network

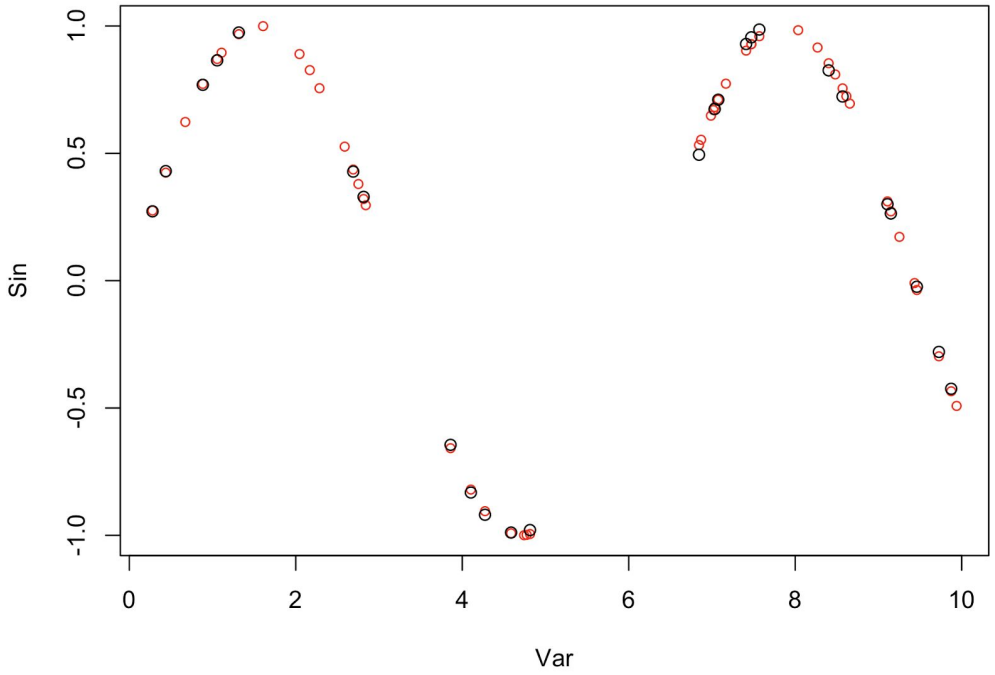
Corrections:
I have redone the threshold investigation and changed *i* to 4 (instead of 1).
All graphs are in this part have been updated, as well as the code in the appendix.

I used MSE between the computed datapoints and the “real” sine of the same variables. The network was trained with the training set and the new points were computed from the validation set. The MSE has a clear drop between *i*=3 and *i*=6, with the minima at *i*=4.

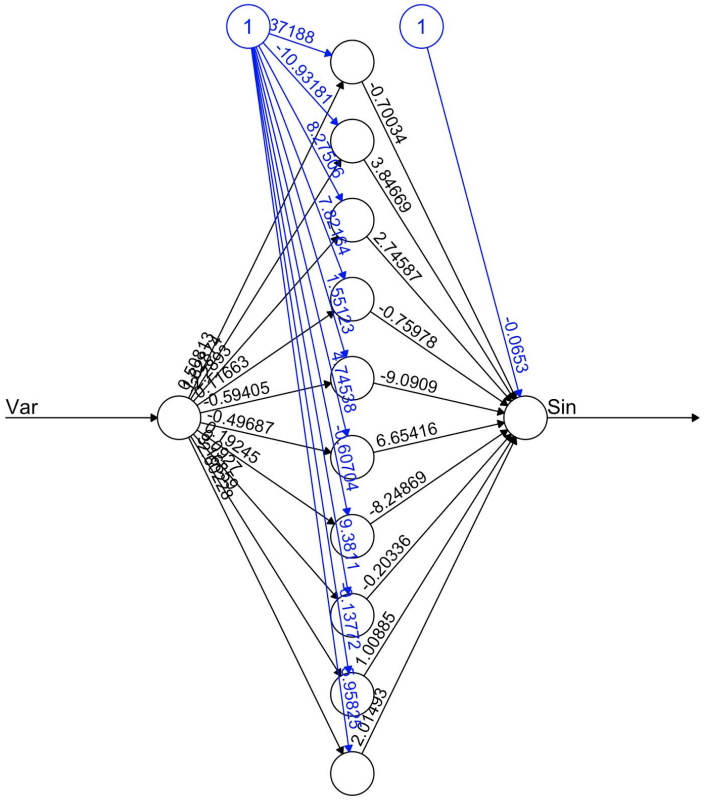


I chose *i*=4, which gave me an MSE of 0.0003400357697.

I ran the neural network again, with the training data and threshold=4/1000 and got the graph below. The predicted points are almost exactly on top of the original data points.



The plotted neural network:



Lab 3 Appendix

Part 1

```
set.seed(1234567890)
    library(geosphere)

    stations <- read.csv("TDDE01/lab3/stations.csv")
    temps <- read.csv("TDDE01/lab3/temps50k.csv")
    st <- merge(stations,temps,by="station_number")

    # parameters
    h_distance <- 200*1000
    h_date <- 20
    h_time <- 4

    position <- c(14.86, 58.4274) # vadstena
    #position <- c(20.1534, 63.4942) # umeå

    my_date <- "2013-01-14"
    #my_date <- "2013-07-14"

    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", "00:00:00")
    temp <- vector(length=length(times))

    st <- subset(st, as.Date(date) < as.Date(my_date))

    # ---- compares positions -----

    station.positions <- cbind(st[,5], st[,4])
    diff.dist = apply(station.positions, 1, function(x) distHaversine(position, x))

    # ---- compares dates -----

    convert_to_days = function(date) {
      tokens = as.numeric(strsplit(date, split="-")[[1]])
      days_in_month = 30.4 #avg days in a month
      return(as.integer((tokens[2]-1)*days_in_month + tokens[3]))
    }

    compare_days = function(x,y) {
      diff.forward = abs(x-y)
      diff.backward = abs((365-x)-y)
      return(min(c(diff.forward, diff.backward)))
    }

    days = sapply(st["date"], as.character)
    days = sapply(days, convert_to_days)
    day = convert_to_days(my_date)
```

```
diff.date = sapply(days, function(x) compare_days(x, day))

# ---- compares times -----

convert_to_hours = function(timestamp) {
  tokens = as.numeric(strsplit(timestamp, split=":")[[1]])
  return(tokens[1])
}

compare_hours = function(x,y) {
  return(abs(x-y))
}

time = sapply(st["time"], as.character)
time = sapply(time, convert_to_hours)

# ---- compares applies kernels, time above -----

smoothed.date = sapply(diff.date/h_date, function(x) exp(-x*x))
smoothed.dist = sapply(diff.dist/h_distance, function(x) exp(-x*x))

diff.time = matrix(nrow=dim(st)[1], ncol=length(times))
smoothed.time = matrix(nrow=dim(st)[1], ncol=length(times))

for(i in 1:length(times)) {
  comp_time = convert_to_hours(times[i])
  diff.time[,i] = sapply(time, function(x) compare_hours(x,comp_time))
  smoothed.time[,i] = sapply(diff.time[,i]/h_time, function(x) exp(-x*x))

  kernels = smoothed.time[,i]*smoothed.date*smoothed.dist
  temp[i] = sum(kernels*st["air_temperature"])/sum(kernels)
}

plot(temp, type="o", xaxt="n", xlab="Time of day", ylab="Temperature")
axis(1, at=1:length(temp), labels=times)
```

Part 2

```
library(neuralnet)

    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
    winit <- runif(31,-1,1)

    for(i in 1:10) {
      nn <- neuralnet(Sin~Var, data=tr, hidden = 10, threshold = i/1000, startweights
    = winit)
      pred <- compute(nn, va$Var)$net.result
      MSE[i] <- mean((pred - va$Sin)**2)
    }

    print(MSE)
    print(which.min(MSE))
    plot(MSE, xlab="Threshold (i/1000)")

    nn <- neuralnet(Sin~Var, data=tr, hidden = 10, threshold = which.min(MSE)/1000,
    startweights = winit)
    plot(nn)

    # Plot of the predictions (black dots) and the data (red dots)
    plot(trva, col = "red", cex = 0.8)
    points(prediction(nn)$rep1)
```

```r
library(pls)
library(tree)

# Part 1: Regression Tree Model

set.seed(12345)
glass <- read.csv2("glass.csv")
glassy <- glass[3] ; glassx <- glass[-3]
training_indices <- sample(1:nrow(glass), nrow(glass) / 2)
remaining_indices <- setdiff(1:nrow(glass), training_indices)
testing_indices <- sample(remaining_indices, length(remaining_indices) / 2)
validation_indices <- setdiff(remaining_indices, testing_indices)
validation <- glass[validation_indices,]
training <- glass[training_indices,]
testing <- glass[testing_indices,]

fit <- tree(Al ~ ., data = training)
validation_deviances <- rep(0, 8)
training_deviances <- rep(0, 8)

for (leaves in 2:8) {
    pruned <- prune.tree(fit, best = leaves)
    predicted <- predict(pruned, newdata = validation, type = "tree")
    validation_deviances[leaves] <- deviance(predicted)
    training_deviances[leaves] <- deviance(pruned)
}

png("training_deviance.png")
plot(2:8,training_deviances[2:8],
     xlab = "Leaves", ylab = "Deviance",
     main = "Training Tree Size")
dev.off()

png("validation_deviance.png")
plot(2:8,validation_deviances[2:8],
     xlab = "Leaves", ylab = "Deviance",
     main = "Validation Tree Size")
dev.off()

# Part 2: Analysis of the Results

# Derived from the above graphs.
fit <- prune.tree(fit, best = 5)
summary(fit) # Gives variables.

png("optimal_tree.png")
plot(fit) # Optimal P!
text(fit) # Nice tree!
title("Optimal Tree")
dev.off()
```

```
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)
```

- **1. Lecture 1d - Regression and regularization**
    - Ridge
    - Lasso
- **6. Lecture 1e - Model selection**
    - Estimator
    - Overfitting
    - Loss functions
    - Model selection
    - Holdout
    - Cross-validation
- **11. Lecture 2a - Linear classification methods**
    - Discriminant Analysis models
    - Logistic regression
    - Generalized Linear Model
- **15. Lecture 2b - Naive Bayes classifiers, Decision trees**
- **20. Lecture 2c - Latent variable models**
    - Principal Component Analysis (PCA)
    - Probabilistic PCA
    - Independent component analysis (ICA)
- **26. Lecture 2d - Principal component regression, Uncertainty estimation**
    - Bootstrap
- **31. Lecture 3a - Kernel methods**
    - Histogram
    - Moving window
    - Kernel classification
    - Kernel regression
    - Kernel density estimation
    - Kernel selection
    - Kernel trick
- **34. Lecture 3b - Support vector machines**
    - Support vector machines for
        - Classification
        - Regression
- **37. Lecture 3c - Neural networks**
    - Neural networks
    - Backpropagation algorithm
    - Regularization
- **40. Lecture 3d - Deep learning**
    - Limitations of Neural Networks
    - Deep Neural Networks
    - Convolutional Networks
    - Rectifier Activation Function
    - Layer-Wise Pre-Training
- **44. L1A1**
    - KNN

    - KKNN
    - ROC
- **46. L1A2**
    - Maximum likelihood
    - Histogram of residuals
- **49. L1A4**
    - Distribution
    - MSE
    - stepAIC
    - Lasso
    - Ridge
    - Cross-validation
- **52. L1 code**
- **57. L2A2**
    - Decision trees
    - Pruned tree
    - Optimal tree
    - Naive bayes
- **61. L2A3**
    - Tree
    - Distribution
    - Bootstrap
    - Prediction band
    - Confidence band
- **64. L2A4**
    - PCA
    - ICA
- **67. L2 code**
- **72. L3A1**
    - Kernels
- **73. L3A2**
    - Neural network
- **76. L3 code**
- **79. Old**

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]`

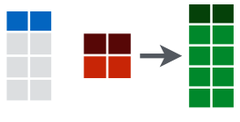
`nrow(df)`
Number of rows.

`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

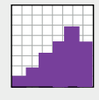
	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	`rnorm`	`dnorm`	`pnorm`	`qnorm`
Poison	`rpois`	`dpois`	`ppois`	`qpois`
Binomial	`rbinom`	`dbinom`	`pbinom`	`qbinom`
Uniform	`runif`	`dunif`	`punif`	`qunif`

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