

Paket

library('kknn') – function kknn för k-nearest neighbor
library('glmnet') – funktion för ridge(alpha=0) och LASSO(alpha=1)
library('MASS') – för linear discriminant analysis (LDA)
library(tree) – för göra träd – även Rpart
library(e1071) – för naiveBayes
library(fastICA) – independent Component analysis (ICA) – function fastICA
library(geosphere) – disthaversine – räkna avstånd på Jorden
library(neuralnet) – neurala nätverk
library(pls) – för principal component regression (function pcr)

Bra funktioner:

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

```

# Filter out all dates after the requested date
sts = sts[as.Date(sts$date) < as.Date(date_in),]
# Converts the time column from string to time, makes comparison easier later
sts$time = strptime(sts$time, format = "%H:%M:%S")

# Remove all saved variables and plots
dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)

# Vector av nollar
v1 = rep(0, antal)

#Random
Var <- runif(50, 0, 10) # Fördelat
rand_obs = rexp(50, rate = 1.13) #??

Dataframe=read.csv2("spambase.csv")
n=dim(Dataframe)[1]
id=sample(1:n, floor(n*0.5))
train=Dataframe[id,]
test=Dataframe[-id,]

```

Lab 1

```

# ----- Assignment 1 -----
# Remove all saved variables and plots
dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)

```

The data file spambase.xlsx contains information about the frequency of various words, characters etc for a total of 2740 e-mails. Furthermore, these e-mails have been manually classified as spams (spam = 1) or regular e-mails (spam = 0).

1. Import the data into R and divide it into training and test sets (50%/50%) by using the following code:

```

Dataframe=read.csv2("spambase.csv")
n=dim(Dataframe)[1]
id=sample(1:n, floor(n*0.5))
train=Dataframe[id,]
test=Dataframe[-id,]

```

2. Use logistic regression (functions `glm()`, `predict()`) to classify the training and test data by the classification principle:

$Y = 1 \text{ if } (Y=1|X)>0.5 \text{ otherwise } Y=0$

```

#Functions
# Predict and Classify the data
predict_with_classif = function(model_in, data_in, classif_val) {
  prediction = predict(model_in, newdata = data_in)
  classif = ifelse(prediction > classif_val, 1, 0)
}
# Create and print confusion matrix
conf_matrix = function(title, true, predicted) {
  t_table = table(true, predicted)
  print(title)
  print(t_table)
}
# Calculate and print missclassification rate
missclass_rate = function(title ,v1, v2) {
  t_table = table(v1, v2)
  missclass = 1-sum(diag(t_table))/sum(t_table)
  print(paste0("missclassification rate for ", title, ": ", missclass))
}
task2n3 = function(model_in, title, data_in, classif_val) {
  prediction = predict_with_classif(model_in, data_in, classif_val)
  conf_matrix(title, data_in$Spam, prediction)
  missclass_rate(title, data_in$Spam, prediction)
}
model_glm = glm(Spam~., data = train)
task2n3(model_glm, "Training data", train, 0.5)
task2n3(model_glm, "Test data", test, 0.5)

```

3. Use logistic regression to classify the test data by the classification principle:

$Y = 1$ if $(Y=1|X)>0.8$ otherwise $Y=0$

and report the confusion matrices (use table()) and the misclassification rates for training and test data. Compare the results. What effect did the new rule have?

```

task2n3(model_glm, "Training data", train, 0.8)
task2n3(model_glm, "Test data", test, 0.8)

```

Answer: Due to the new rule almost nothing gets classified as not spam. The missclassification rate is higher as result

4. Use standard classifier kknn() with K=30 from package kknn, report the the misclassification rates for the training and test data and compare the results with step 2.

```

library('kknn')
step4n5 = function(title, data_train, data_test, threshold) {
  prediction = kknn(Spam ~ ., train = data_train, test=data_test, k = threshold)
  prediction = ifelse(prediction$fitted.values > 0.5, 1, 0)
  conf_matrix(title, data_test$Spam, prediction)
  missclass_rate(title, data_test$Spam, prediction)
}
step4n5("Training data", train, train, 30)
step4n5("Test data", train, test, 30)

```

5. Repeat step 4 for K=1 and compare the results with step 4. What effect does the decrease of K lead to and why?

```
step4n5("Training data", train, train, 1)
step4n5("Test data", train, test, 1)
```

```
# ----- Assignment 2 -----
# Remove all saved variables and plots
dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)
```

The data file machines.xlsx contains information about the lifetime of certain machines, and the company is interested to know more about the underlying process in order to determine the warranty time. The variable is following:

Length: shows lifetime of a machine

1. Import the data to R.

```
Dataframe=read.csv2("machines.csv")
```

2. Assume the probability model $p(x|\theta) = \theta e^{-\theta x}$

for $x = \text{Length}$ in which observations are independent and identically distributed. What is the distribution type of x ? Write a function that computes the log-likelihood $\log(p(x|\theta))$ for a given θ and a given data vector x . Plot the curve showing the dependence of log-likelihood on θ where the entire data is used for fitting. What is the maximum likelihood value of θ according to the plot?

```
loglike = function(x, theta) {
  n = length(x)
  return (n*log(theta)-theta*sum(x))
}
```

```
step2n3 = function(data_in, thetas) {
  LLH = vector("numeric", length =0)
  for(theta in thetas) {
    LLH = c(LLH, loglike(data_in, theta))
  }
  print(thetas[which.max(LLH)])
  return(LLH)
}
```

```
step2 = function(data_in, thetas) {
  LLH = step2n3(data_in$Length, thetas)
  plot(thetas, LLH, type="l", col="blue")
}
thetas = seq(0, 30, 0.01)
step2(Dataframe, thetas)
```

3. Repeat step 2 but use only 6 first observations from the data, and put the two log-likelihood curves (from step 2 and 3) in the same plot. What can you say about reliability of the maximum likelihood solution in each case?

```
step3 = function(data_in, thetas) {  
  LLH = step2n3(data_in, thetas)  
  points(thetas, LLH, type="l", col="red")  
}  
step3(Dataframe[0:6], thetas)
```

4. Assume now a Bayesian model with $p(x|\theta)=\theta^*e^{(-\theta^*x)}$ and a prior $p(\theta)=\lambda e^{(-\lambda\theta)}$. Write a function computing $l(\theta)=\log(p(x|\theta)p(\theta))$. What kind of measure is actually computed by this function? Plot the curve showing the dependence of $l(\theta)$ on θ computed using the entire data and overlay it with a plot from step 2. Find an optimal theta and compare your result with the previous findings.

```
bayes = function(x, theta) {  
  lambda = 10  
  log_prior = log(lambda)-lambda*theta  
  return(log_prior+loglike(x,theta))  
}  
  
LLH = vector("numeric", length=0)  
for(theta in thetas) {  
  LLH = c(LLH, bayes(Dataframe$Length, theta))  
}  
  
points(thetas, LLH, type="l", col="green")
```

5. Use theta value found in step 2 and generate 50 new observations from $p(x|\theta)=\theta^*e^{(-\theta^*x)}$ (use standard random number generators). Create the histograms of the original and the new data and make conclusions.

```
rand_obs = rexp(50, rate = 1.13)  
hist(rand_obs,  
  col=rgb(1,0,0,0.5),  
  xlim=c(0,6),  
  ylim=c(0,30),  
  main="Overlapping Histogram",  
  xlab="Variable")  
  
hist(Dataframe$Length,  
  col=rgb(0,0,1,0.5),  
  add=T)  
box()
```

----- Assignment 3 -----

```
# Remove all saved variables and plots
```

```

dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)

Implement an R function that performs feature selection (best subset selection) in linear regression by using k-fold cross-validation without using any specialized function like lm() (use only basic R functions). Your function should depend on:

X: matrix containing X measurements.
Y: vector containing Y measurements
Nfolds: number of folds in the cross-validation.

You may assume in your code that matrix X has 5 columns. The function should plot the CV scores computed for various feature subsets against the number of features, and it should also return the optimal subset of features and the corresponding cross-validation (CV) score. Before splitting into folds, the data should be permuted, and the seed 12345 should be used for that purpose. Report the resulting plot and interpret it. Report the optimal subset of features and comment whether it is reasonable that these specific features have largest impact on the target.

mylin=function(X,Y, Xpred){
  Xpred1 = cbind(1,Xpred)
  X1 = cbind(1, X)
  beta = solve( t(X1) %*% X1 ) %*% t(X1) %*% Y # obtained using the "training" data matrix X
  Res = Xpred1%*%beta # y_hat for the "test" data
  return(Res)
}

myCV=function(X,Y,Nfolds){
  n=length(Y) # number of observations (rows)
  p=ncol(X) # number of covariates (variables or columns)
  set.seed(12345)
  ind=sample(n,n) # indexes are randomized
  X1=X[ind,] # randomize the order of the observations
  Y1=Y[ind]
  sF=floor(n/Nfolds) # number of observations inside each fold
  MSE=numeric(2^p-1) # vector of the length of 2^p-1 combinations
  Nfeat=numeric(2^p-1)
  Features=list() # features that will be selected
  curr=0 # current
  folds_obs <- cut(1:n,breaks=Nfolds,labels=FALSE)
  #we assume 5 features.
  for (f1 in 0:1)
    for (f2 in 0:1)
      for(f3 in 0:1)
        for(f4 in 0:1)
          for(f5 in 0:1){
            model= c(f1,f2,f3,f4,f5)
            if (sum(model)==0) next()
            SSE=0
            for (k in 1:Nfolds){
              #MISSING:compute which indices should belong to current fold

```

```

indices <- ind[which(folds_obs==k)] #indeces of the observations in fold k
#MISSING:implement cross-validation for model with features in "model" and iteration i.
X_mylin <- X[-indices,which(model==1)]
XPred_mylin <- X[indices,which(model==1)]
Y_mylin <- Y[-indices]
#MISSING:Get the predicted values for fold k, Ypred, and the original values for fold 'k', Yp.
Ypred <- mylin(X_mylin, Y_mylin, XPred_mylin)
Yp <- Y[indices]
SSE=SSE+sum((Ypred-Yp)^2)
}
curr=curr+1
MSE[curr]=SSE/n
Nfeat[curr]=sum(model)
Features[[curr]]=model
}
plot(Nfeat, MSE) #MISSING: plot MSE against number of features
abline(h=MSE[which.min(MSE)], col="red")
i=which.min(MSE)
return(list(CV=MSE[i], Features=Features[[i]]))
}

```

2. Test your function on data set swiss available in the standard R repository:

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

```

data("swiss")
myCV(as.matrix(swiss[,2:6]), swiss[[1]], 5)

```

----- Assignment 4 -----

```

# Remove all saved variables and plots
dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)

```

The Excel file tecator.xlsx contains the results of study aimed to investigate whether a near infrared absorbance spectrum can be used to predict the fat content of samples of meat. For each meat sample the data consists of a 100 channel spectrum of absorbance records and the levels of moisture (water), fat and protein. The absorbance is -log10 of the transmittance measured by the spectrometer. The moisture, fat and protein are determined by analytic chemistry.

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

```
Dataframe=read.csv2("tecator.csv")
plot(Dataframe$Moisture, Dataframe$Protein)
```

2. Consider model M_i in which Moisture is normally distributed, and the expected Moisture is a polynomial function of Protein including the polynomial terms up to power i (i.e M_1 is a linear model, M_2 is a quadratic model and so on). Report a probabilistic model that describes M_i . Why is it appropriate to use MSE criterion when fitting this model to a training data?

```
#  $M_i = a*i * protein^i + a*(i-1) * protein^{i-1} + \dots + a * protein$ 
```

3. Divide the data into training and validation sets(50%/50%) and fit models M_i , $i=1..6$. For each model, record the training and the validation MSE and present a plot showing how training and validation MSE depend on i (write some R code to make this plot). Which model is best according to the plot? How do the MSE values change and why? Interpret this picture in terms of bias-variance tradeoff.

```
mean_square_error = function(v1, v2) {
  SE = (v1-v2)
  SE = SE^2
  SE = mean(SE)
  return(SE)
}

calc_MSE = function(the_model, data) {
  SE = predict(the_model, newdata = data)
  SE = data$Moisture - SE
  SE = SE^2
  MSE = mean(SE)
  return(MSE)
}

n=dim(Dataframe)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=Dataframe[id,]
test=Dataframe[-id,]
test = test[1:dim(train)[1],]
MSEs = vector("numeric", length=0)

step3 = function(data_in, data_train) {
  for(i in seq(1,6)) {
    model_i = lm(Moisture ~ poly(Protein,i), data = data_train)
    prediction = predict(model_i, newdata = data_in)
    MSEs = c(MSEs, mean_square_error(prediction, data_in$Moisture))
  }
  return (MSEs)
}
```

```

MSE_train = step3(train, train)
MSE_test = step3(test, train)

plot(seq(1,6), MSE_train, type="l", col="blue", ylim=c(20,45))
lines(seq(1,6), MSE_test, type="l", col="red")

```

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

```

t_data = Dataframe[,0:(dim(Dataframe)[2]-2)]
model_lm = lm(Fat ~ ., data=t_data)
AIC_lm = stepAIC(model_lm)
print(paste0(length(AIC_lm$coefficients), " were selected"))

```

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

```
library('glmnet')
```

```

step5n6 = function(data_in, alpha_in) {
  t_model = glmnet(as.matrix(data_in[2:101]), data_in[,102], alpha=alpha_in, family="gaussian")
  plot(t_model, xvar="lambda", label=TRUE)
}

```

```
step5n6(t_data, 0)
```

6. Repeat step 5 but fit LASSO instead of the Ridge regression and compare the plots from steps 5 and 6. Conclusions?

```
step5n6(t_data, 1)
```

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

```

lambda = seq(0.50, 0.01)
cv_lasso = cv.glmnet(as.matrix(t_data[,2:101]), t_data[,102], alpha=1, family="gaussian",
lambda=lambda)
plot(cv_lasso, type="b", col="Green")

```

Lab 2

----- Assignment 1 -----

```

# Remove all saved variables and plots
dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)

```

The data file australian-crabs.csv contains measurements of various crabs, such as Frontal lobe, Rear width and others

```
Dataframe=read.csv2("australian-crabs.csv")
```

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

```
plot(Dataframe$CL, Dataframe$RW, col=Dataframe$sex)
```

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

```
missclass_rate = function(title ,v1, v2) {  
  t_table = table(v1, v2)  
  missclass = 1-sum(diag(t_table))/sum(t_table)  
  print(paste0("missclassification rate for ", title, ": ", missclass))  
}
```

```
step2n3 = function(t_model) {  
  predict_lda = predict(model_lda, newdata = Dataframe)  
  plot(Dataframe$CL, Dataframe$RW, col=predict_lda$class)  
  missclass_rate("Lda", predict_lda$class, Dataframe$sex)  
}
```

```
library('MASS')  
model_lda = lda(sex ~ RW+CL, data = Dataframe)  
step2n3(model_lda)
```

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

```
model_lda = lda(sex ~ RW+CL, data = Dataframe, prior = c(0.1,0.9))  
step2n3(model_lda)
```

4. Make a similar kind of classification by logistic regression (use function glm()), plot the classified data and compute the misclassification error. Compare these results with the LDA results. Finally, report the equation of the decision boundary and draw the decision boundary in the plot of the classified data.

```
model_glm = glm(sex ~ RW+CL, data = Dataframe, family = "binomial")  
predict_glm = predict(model_glm, newdata = Dataframe)  
plot(predict_glm, type = "l")  
predict_glm = ifelse(predict_glm > 0.5, "Male", "Female")  
missclass_rate("GLM", predict_glm, Dataframe$sex)  
plot(Dataframe$CL, Dataframe$RW, col=as.factor(predict_glm))  
slope <- coef(model_glm)[2]/(-coef(model_glm)[3])  
intercept <- coef(model_glm)[1]/(-coef(model_glm)[3])  
print(paste0("Equation: ", intercept, " + ", slope, "x"))  
abline(intercept, slope)
```

----- Assignment 2 -----

```
# Remove all saved variables and plots  
dev.off()  
rm(list=ls())  
# Setting up rng and seed  
RNGversion('3.5.1')  
set.seed(12345)
```

The data file creditscoring.xls contains data retrieved from a database in a private enterprise. Each row contains information about one customer. The variable good/bad indicates how the customers have managed their loans. The other features are potential predictors. Your task is to derive a prediction model that can be used to predict whether or not a new customer is likely to pay back the loan.

1. Import the data to R and divide into training/validation/test as 50/25/25: use data partitioning code specified in Lecture 1e.

```
Dataframe=read.csv2("creditscoring.csv")  
n=dim(Dataframe)[1]  
id=sample(1:n, floor(n*0.4))  
train=Dataframe[id,]  
id1=setdiff(1:n, id)  
id2=sample(id1, floor(n*0.3))  
valid=Dataframe[id2,]  
id3=setdiff(id1,id2)  
test=Dataframe[id3,]
```

2. Fit a decision tree to the training data by using the following measures of impurity

- a. Deviance
- b. Gini index

and report the misclassification rates for the training and test data. Choose the measure providing the better results for the following steps.

```
library(tree)  
# Calculate and print missclassification rate  
missclass_rate = function(title ,v1, v2) {  
  t_table = table(v1, v2)  
  missclass = 1-sum(diag(t_table))/sum(t_table)  
  print(paste0("missclassification rate for ", title, ": ", missclass))  
}  
  
class_and_value = function(title, t_tree, data_in, threshold) {  
  classification = predict(t_tree, newdata = data_in)  
  prediction = ifelse(classification[,2] > threshold, "good", "bad")  
  missclass_rate(title, prediction, data_in$good_bad)  
}
```

```

step2 = function(impurity) {
  t_tree = tree(good_bad ~ ., data = train, split = impurity)
  for(d in list(train, test)) {
    class_and_value(impurity, t_tree, d, 0.5)
  }
}

for(imp in c("deviance", "gini")) {
  step2(imp)
}
# Deviance is the better choice
t_tree = tree(good_bad ~ ., data = train, split = "deviance")

```

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

```

calcScore = function(t_tree, data_in) {
  prediction = predict(t_tree, newdata = data_in, type="tree")
  return(deviance(prediction))
}

score_train = rep(0,20)
score_valid = rep(0,20)

for(i in 2:20) {
  pruned_tree = prune.tree(t_tree, best = i)
  score_train[i] = calcScore(pruned_tree, train)
  score_valid[i] = calcScore(pruned_tree, valid)
}

plot(score_train, type="l", col="blue", xlim = c(2,20))
plot(score_valid, type="l", col="red", xlim = c(2,20))

t_tree = prune.tree(t_tree, best = 4)
plot(t_tree)
text(t_tree, pretty=0)
# Depth is 3, variables used are: duration, history and savings
prediction = predict(t_tree, newdata = test)
prediction = ifelse(prediction[,2] > 0.5, "good", "bad")
missclass_rate("pruned tree", prediction, test$good_bad)

```

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

```

library(e1071)
navbay = naiveBayes(good_bad ~ ., data = train)
for(d in list(train, test)) {
  prediction = predict(navbay, newdata = d)
}

```

```

missclass_rate("naive bayes", prediction, d$good_bad)
}

```

5. Use the optimal tree and the Naïve Bayes model to classify the test data by using the following principle: $Y=1$ if $p(Y="good" | x) > \pi$ otherwise $Y=0$, where $\pi = 0.05, 0.1, \dots, 0.9, 0.95$. Compute the TPR and FPR values for the two models and plot the corresponding ROC curves. Conclusion?

```

conf_matrix = function(true, predicted) {
  t_table = table(true, predicted)
  return(t_table)
}

step5 = function(pi, model_in, d, pred_type) {
  TPR = rep(1:length(pi))
  FPR = rep(1:length(pi))
  for(i in 1:length(pi)) {
    prediction = predict(model_in, newdata = d, type=pred_type)
    prediction = ifelse(prediction[,2] > pi[i], "good", "bad")
    confusion = conf_matrix(d$good_bad, prediction)
    TPR[i] = confusion[1,1]/sum(confusion[1,])
    FPR[i] = confusion[2,1]/sum(confusion[2,])
  }
  return(c(TPR, FPR))
}

pi = seq(0.05,0.95,0.05)
vals_navy = step5(pi, navy, train, "raw")
vals_tree = step5(pi, t_tree, train, "vector")
plot(vals_navy[1:19], vals_navy[20:38], type = "l", col="blue")
plot(vals_tree[1:19], vals_tree[20:38], type = "l", col="blue")

```

6. Repeat Naïve Bayes classification as it was in step 4 but use the following loss matrix: $L = (0,1; 10,0)$ and report the confusion matrix for the training and test data. Compare the results with the results from step 4 and discuss how the rates has changed and why.

```

for(d in list(train, test)) {
  prediction = predict(navy, newdata = d, type = "raw")
  classified = rep(1:dim(prediction)[1])
  for(i in 1:dim(prediction)[1]) {
    classified[i] = ifelse(prediction[i,1]*10 < prediction[i,2]*1, "good", "bad")
  }
  print(conf_matrix(d$good_bad, classified))
}

```

----- Assignment 4 -----

```

# Remove all saved variables and plots
dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)

```

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

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

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

```
#PCA is unsupervised learning
Dataframe$Viscosity = c()
pca = prcomp(Dataframe)
lambda = pca$sdev^2
plot(pca)
sprintf("%2.3f",lambda/sum(lambda)*100)
# We see from the line above that the first 2 components cover 99%
plot(pca$x[,1], pca$x[,2], ylim=c(-0.15,0.15))
```

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

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

3. Perform Independent Component Analysis with the number of components selected in step 1 (set seed 12345). Check the documentation for the fastICA method in R and do the following:

a. Compute $W' = K^*W$ and present the columns of W' in form of the trace plots. Compare with the trace plots in step 2 and make conclusions. What kind of measure is represented by the matrix W' ?

b. Make a plot of the scores of the first two latent features and compare it with the score plot from step 1.

```
library(fastICA)
fica = fastICA(Dataframe, 2)
# Use %*% when you want to use matrix operation
Wtick = fica$K %*% fica$W
plot(Wtick[,1], main="PC1")
plot(Wtick[,2], main="PC2")
```

Lab 3

----- Assignment 1 -----

```
# Remove all saved variables and plots
dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)
```

Implement a kernel method to predict the hourly temperatures for a date and place in Sweden. To do so, you are provided with the files stations.csv and temps50k.csv. These files contain information about weather stations and temperature measurements in the stations at different days and times.

The data have been kindly provided by the Swedish Meteorological and Hydrological Institute (SMHI). You are asked to provide a temperature forecast for a date and place in Sweden. The forecast should consist of the predicted temperatures from 4 am to 24 pm in an interval of 2 hours. Use a kernel that is the sum of three Gaussian kernels:

The first to account for the distance from a station to the point of interest.

The second to account for the distance between the day a temperature measurement was made and the day of interest.

The third to account for the distance between the hour of the day a temperature measurement was made and the hour of interest.

Choose an appropriate smoothing coefficient or width for each of the three kernels above. Answer to the following questions:

Show that your choice for the kernels' width is sensible, i.e. that it gives more weight to closer points. Discuss why your definition of closeness is reasonable.

Instead of combining the three kernels into one by summing them up, multiply them. Compare the results obtained in both cases and elaborate on why they may differ. Note that the file temps50k.csv may contain temperature measurements that are posterior to the day and hour of your forecast.

You must filter such measurements out, i.e. they cannot be used to compute the forecast. Feel free to use the template below to solve the assignment.

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

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

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

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

# Function to calculate mod, required for days
mod = function(Lside, Rside) {
  returnValue = Lside - Rside * floor(Lside/Rside)
  return(returnValue)
}

# Functions to calculate the differences
calc_distance = function(input, sts) {
```

```

dist = distHaversine(data.frame(sts$longitude,sts$latitude), input)
return(dist)
}

# Take into consideration the difference in days between years
# Two days ahead five years ago should give difference 2 days
# Also that it allows days backwards from 25/02 to 24/02 is 1 day, not 364
calc_day = function(input, sts) {
  days = difftime(sts$date, input)
  days = mod(days, 365)
  days = ifelse(days > 365/2, (365 - days), days)
  return(days)
}
calc_time = function(input, sts) {
  time = as.numeric(difftime(input, sts$time, units = "secs"))
  time = time /3600
  time = ifelse(abs(time) > 12, (24 -abs(time)), abs(time))
  return(time)
}

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

  for(i in 1:length(times_in)) {
    # Calculating the difference and kernel values for time
    time = calc_time(times_in[i],sts)
    kernel_time = kernel_gauss(time, h_time)
    # Summing/ multiplying all kernel value and calculate the predicted temperature
    k_sum = kernel_distance+kernel_day+kernel_time
    k_mul = kernel_distance*kernel_day*kernel_time
    temps_sum[i] = sts$air_temperature%*% k_sum / sum(k_sum)
    temps_mul[i] = sts$air_temperature%*% k_mul /sum(k_mul)
  }
  #Plot and print result
  plot(temps_sum, type="o")
  print("With summerizing:")
}

```

```

print(tempsum)
plot(tempsmul, type="o")
print("With multiplying:")
print(tempsmul)
}

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

```

----- Assignment 2 -----

```

# Remove all saved variables and plots
dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)

```

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

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

```

data(spam)
data <- spam
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.70))
train=data[id,]
test=data[-id,]
ksvm.model <- function(x) {
  model <- ksvm(type~, data = train, kernel = "rbfdot", kpar=list(sigma=0.05), C = x)
  pred.model <- predict(model, test)
  confmat <- table(test$type, pred.model)
  misclass <- 1 - sum(diag(confmat)/sum(confmat))
  cat("\nModel misclassification rate for C = ", x, "is: ", misclass)
  confmat
}
ksvm.model(0.5)

```

```

ksvm.model(1)
ksvm.model(5)
model.new <- ksvm(type~, data = train, kernel = "rbfdot", kpar=list(sigma=0.05), C = 1)
model.new

```

----- Assignment 3 -----

```

# Remove all saved variables and plots
dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)

```

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

```

mean_square_error = function(v1, v2) {
  SE = (v1-v2)
  SE = SE^2
  SE = mean(SE)
  return(SE)
}

# Function to predict and return the MSE
# For NN use compute
pred_MSE = function(model_in, data_in) {
  prediction = compute(model_in, covariate=data_in)$net.result
  return(mean_square_error(prediction, data_in$Sin))
}

library(neuralnet)
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation
#Hidden = 10 from assignment
hidden_val = 10
i_vals = seq(1,10)
# Random initialization of the weights in the interval [-1, 1]
# Number of starting weights is number of connections in the plot of the nn
winit = runif(31, -1, 1)
MSE_tr = rep(0, length(i_vals))

```

```

MSE_va = rep(0, length(i_vals))
for(i in i_vals) {
  nn <- neuralnet(Sin ~ Var, data = tr, hidden = hidden_val, startweights = winit, threshold = i/1000)
  MSE_tr[i] = MSE_tr[i] + pred_MSE(nn, tr)
  MSE_va[i] = MSE_va[i] + pred_MSE(nn, va)
}
plot(MSE_tr, type= "l")
plot(MSE_va, type= "l")
# Lowest MSE for i = 5
nn <- neuralnet(Sin ~ Var, data = tr, hidden = hidden_val, startweights = winit, threshold = 5/1000)
plot(nn)
# Plot of the predictions (black dots) and the data (red dots)
plot(prediction(nn)$rep1)
points(trva, col = "red")

```

Tentauppgifter

The data file video.csv contains characteristics of a sample of Youtube videos. Import data to R and divide it randomly (50/50) into training and test sets.

1. Perform principal component analysis using the numeric variables in the training data except of “utime” variable. Do this analysis with and without scaling of the features. How many components are necessary to explain more than 95% variation of the data in both cases? Explain why so few components are needed when scaling is not done.

```

data0=read.csv("video.csv")
data1=data0
data1$codec=c()
n=dim(data1)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data1[id,]
test=data1[-id,]
data11=data1
data11$utime=c()
res=prcomp(data11)
lambda=res$sdev^2
sprintf("%2.3f",cumsum(lambda)/sum(lambda)*100)
res=prcomp(scale(data11))
lambda=res$sdev^2
sprintf("%2.3f",cumsum(lambda)/sum(lambda)*100)

```

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

```
library(pls)
```

```

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

```

3. Use PCR model with $MM=8$ and report a fitted probabilistic model that shows the connection between the target and the principal components.

```

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

```

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

```

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

```

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

```

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

```

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

```

library(tree)
m4=tree(as.factor(class)~frames+duration, data=data2)
set.seed(12345)

```

```

cv.res=cv.tree(m4)
plot(cv.res$size, cv.res$dev, type="b", col="red")
print(m4)
plot(m4)
missclass(data2$class, predict(m4, type="class"))

#-----

```

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

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

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

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

```

library(kernlab)
set.seed(1234567890)
data(spam)
# Model selection
index <- sample(1:4601)
tr <- spam[index[1:2500], ]
va <- spam[index[2501:3501], ]
te <- spam[index[3502:4601], ]
filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=0.5)
mailtype <- predict(filter,va[,-58])
t <- table(mailtype,va[,58])
(t[1,2]+t[2,1])/sum(t)
filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=1)
mailtype <- predict(filter,va[,-58])
t <- table(mailtype,va[,58])
(t[1,2]+t[2,1])/sum(t)
filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=5)
mailtype <- predict(filter,va[,-58])
t <- table(mailtype,va[,58])
(t[1,2]+t[2,1])/sum(t)
# Error estimation
filter <- ksvm(type~.,data=spam[index[1:3501], ],kernel="rbfdot",kpar=list(sigma=0.05),C=1)
mailtype <- predict(filter,te[,-58])
t <- table(mailtype,te[,58])
(t[1,2]+t[2,1])/sum(t)
# Final model
filter <- ksvm(type~.,data=spam,kernel="rbfdot",kpar=list(sigma=0.05),C=1)

#-----

```

Train a neural network (NN) to learn the trigonometric sine function. To do so, sample 50 points uniformly at random in the interval [0, 10]. Apply the sine function to each point. The resulting pairs

are the data available to you. Use 25 of the 50 points for training and the rest for validation. The validation set is used for early stop of the gradient descent. Consider threshold values $i/1000$ with $i = 1, \dots, 10$. Initialize the weights of the neural network to random values in the interval $[-1, 1]$. Consider two NN architectures: A single hidden layer of 10 units, and two hidden layers with 3 units each. Choose the most appropriate NN architecture and threshold value. Motivate your choice. Feel free to reuse the code of the corresponding lab.

- (1p) Estimate the generalization error of the NN selected above (use any method of your choice).
- (1p) In the light of the results above, would you say that the more layers the better ? Motivate your answer.

```
library(neuralnet)
# two layers
set.seed(1234567890)
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation
# plot(trva)
# plot(tr)
# plot(va)
restr <- vector(length = 10)
resva <- vector(length = 10)
winit <- runif(22, -1, 1) # Random initializaiton of the weights in the interval [-1, 1]
for(i in 1:10) {
  nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = c(3,3), startweights = winit, threshold =
i/1000, lifesign = "full")
  # nn$result.matrix
  aux <- compute(nn, tr[,1])$net.result # Compute predictions for the trainig set and their squared
error
  restr[i] <- sum((tr[,2] - aux)**2)/2
  aux <- compute(nn, va[,1])$net.result # The same for the validation set
  resva[i] <- sum((va[,2] - aux)**2)/2
}
plot(restr, type = "o")
plot(resva, type = "o")
restr
resva

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

```

winit <- runif(41, -1, 1) # Random initializaiton of the weights in the interval [-1, 1]
for(i in 1:10) {
  nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = c(10), startweights = winit, threshold =
  i/1000, lifesign = "full")
  # nn$result.matrix
  aux <- compute(nn, tr[,1])$net.result # Compute predictions for the trainig set and their squared
  error
  restr[i] <- sum((tr[,2] - aux)**2)/2
  aux <- compute(nn, va[,1])$net.result # The same for the validation set
  resva[i] <- sum((va[,2] - aux)**2)/2
}
plot(restr, type = "o")
plot(resva, type = "o")
restr
resva
# estimate generalization error for the best run above (one layer with threshold 4/1000)
Var <- runif(50, 0, 10)
te <- data.frame(Var, Sin=sin(Var))
winit <- runif(31, -1, 1)
nn <- neuralnet(formula = Sin ~ Var, data = trva, hidden = 10, startweights = winit, threshold =
  4/1000, lifesign = "full")
sum((te[,2] - compute(nn, te[,1])$net.result)**2)/2

```

Basic concepts

Lecture 1a

Course leader: Oleg Sysoev

732A99/TDDE01

1

Course topics

Block 1

- Basic concepts in machine learning. Software for ML.
- Regression, regularization and model selection
- Classification methods
- Dimensionality reduction and uncertainty estimation
- Support vector machines and kernel methods
- Neural networks and deep learning

Block 2

- Splines and additive models. High-dimensional problems
- Mixture models and online learning. Ensemble methods

732A99/TDDE01

2

Course organization

- 1 topic= 4-5 lectures +1 lab (2h* 3)+seminar
- Course given as
 - 732A99 (9 ECTS): Block 1+Block2
 - 732A68 (9 ECTS): Block 1+Block2
 - TDDE01 (6 ECTS): Block 1
- **Labs**
 - SU rooms used
 - Take around 8h
 - Individual and group reports
 - Sharing only ideas in the group, not text or codes
 - Bring your own laptop if you have – limited amount of computers in the rooms
 - Deadlines
 - Individual Special Tasks (optional)– if you solve all of them and get at least 14 points at the exam, you get 2 points more.
 - Published a couple of days in advance – try doing before coming to the first lab session!
 - Submission via LISAM

732A99/TDDE01

3

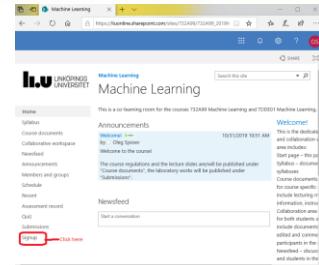
Course organization

- **Lectures**
 - Available as PowerPoint or PDF, normally at LISAM
- **Seminars**
 - Speaker and opponent groups
 - Is a laboratory part, obligatory attendance for speakers and opponents
 - Discussion of the latest lab.
 - Note: lab assignments are slightly different for TDDE01/732A99 but all kinds of assignments may appear at the exam!
 - Define your group (3 persons) as soon as possible via Lisam (see next two slides)
 - **Difficult to find a group? Put your name in some empty group item**

732A99/TDDE01

4

Define your group

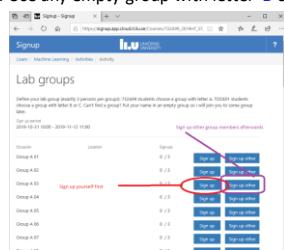


732A99/TDDE01

5

Define your group

- 732A99: Use any empty group with letter **A**
- TDDE01: Use any empty group with letter **B** or **C**

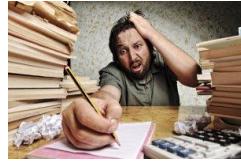


732A99/TDDE01

6

Course organization

- Examination
 - TDDE01, 732A99: laboratory part + computer-based exam
- Lecture 1c is 'Introduction to R'
- Lecture 1b is 'Basic Statistics'



732A99/TDDE01

7

What is Machine Learning ?

- Machine learning is a subfield of **computer science** that evolved from the study of **pattern recognition** and computational learning theory in **artificial intelligence**.
- Machine learning explores the study and construction of **algorithms** that can **learn** from and make **predictions** on **data**. Such algorithms operate by building a model from example inputs in order to make data-driven predictions or **decisions**, rather than following strictly static program instructions.

Wikipedia (Oct 15, 2016).

732A99/TDDE01

8

Machine Learning and Statistics

- ML=intersection of **computer science**, **statistics** and **artificial intelligence**.
 - Related: **data mining**, **knowledge discovery** and **data science**.
- ML uses mainly **statistical (probabilistic) models** for **analyzing data**.
 - Data mining and knowledge discovery tend to use less rigorous, but often effective, algorithms.
 - ML is not a discovery of a hidden information (Data Mining)
- ML vs Statistics: ML has a **heavier focus on prediction**, and lesser on interpretation.
- ML applications often involve large sets → **computational complexity** of algorithms is important.
 - Statistics often does not care about runtime

732A99/TDDE01

9

Why probability models?

- Probability models and statistical inference provide a **framework**
- A principled **way to think** about any problem in machine learning
 - Probabilistic model → Estimation → Prediction
- Probabilistic models **quantify uncertainties**.
 - Deterministic answers may often be inappropriate



The currency exchange rate tomorrow will be 10.41!

732A99/TDDE01

10

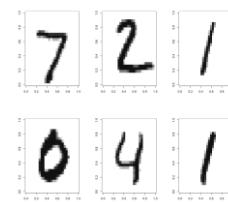
Why probability models?

As robotics is now moving into the open world, the issue of **uncertainty** has become a major stumbling block for the design of capable robot systems. Managing uncertainty is possibly the most important step towards robust real-world robot systems.
from the book *Probabilistic Robotics* by Thrun et al.

732A99/TDDE01

11

Example: classifying handwritten digits



732A99/TDDE01

12

Example: classifying handwritten digits

Training data: 60000 images.

Test data: 10000 images.

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

Methods:

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

13

Example: classifying handwritten digits

- Confusion matrix

		PREDICTION									
		0	1	2	3	4	5	6	7	8	9
TRUTH	0	966	0	8	1	1	7	9	2	4	6
	1	0	1121	0	1	0	2	3	13	7	7
2	2	2	957	13	5	4	1	21	21	7	8
3	0	2	9	947	0	29	1	3	12	18	1
4	0	0	12	1	948	5	5	9	8	32	1
5	6	1	3	19	1	816	9	1	24	9	1
6	4	4	13	1	7	12	926	0	18	1	1
7	1	0	9	18	2	2	0	954	5	13	1
8	1	4	17	11	2	18	1	3	892	4	1
9	0	1	3	6	24	5	0	22	5	927	1

14

Example: smartphone typing predictions



15

Example: smartphone typing predictions

- Assume a simple (Markov) model of a sentence:
 $p(w_1, \dots, w_n) = p(w_1)p(w_2|w_1) \dots p(w_n|w_{n-1})$

Intuition:

- $p(\text{person}|\text{crazy}) = 0.1$ Highest P(?)|Donald ?
- $p(\text{horse}|\text{crazy}) = 0.0001$

- Probability for sentence depends only on $p(w_n|w_{n-1})$

- How to compute? Investigate a lot of data!

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

- In practice, more advanced model used

- Neural networks for ex.

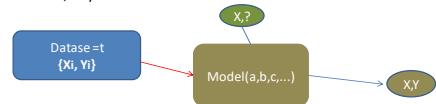
16

Types of learning

- Supervised learning** (classification, regression)

- Compute parameters from data
- Given features of a new object, predict target
- Classification** (Y=categorical), **Regression** (Y=continuous)

- Most of ML models: Neural Nets, Decision Trees, Support Vector Machines, Bayesian nets



17

Types of learning

- Unsupervised learning** (\rightarrow Data Mining)

- No target
- Aim is to extract interesting information about
 - Relations of parameters to each other
 - Grouping of objects

Ex: clustering, density estimation, association analysis

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

18

732A99/TDDE01

Types of learning

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

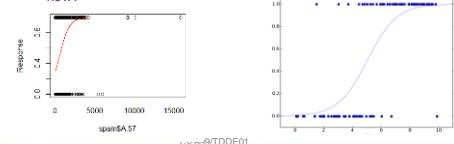
732A99/TDDE01

19

Logistic regression

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

We can also make point predictions
-how?



22

Basic ML ingredients

- Data D :** observations (cases)
 - Features X_1, \dots, X_p
 - Targets Y_1, \dots, Y_r
 - ...
- Model $P(x|w_1, \dots, w_k)$ or $P(y|x, w_1, \dots, w_k)$**
 - Example: Linear regression $p(y|x, w) = N(w_0 + w_1x, \sigma^2)$
- Learning procedure** (data → get parameters \hat{w} or $p(w|D)$)
 - Maximum likelihood, Bayesian estimation...
- Prediction** of new data X^{new} by using the fitted model

732A99/TDDE01

20

K-nearest neighbor density estimation

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

732A99/TDDE01

23

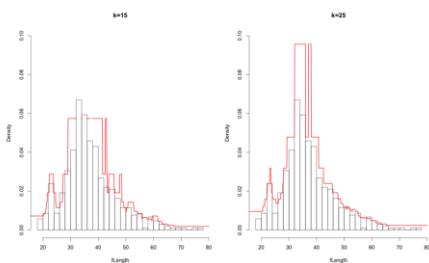
Types of data sets

- Training data** (training set D): used for fitting the model
 - Supervised learning: w_i in $P(y|x, w_1, \dots, w_k)$ estimated using D
- Test data** (test set T): used for predictions
 - Supervised learning: estimate $p(Y)$ or \hat{Y} for new x

732A99/TDDE01

21

K-nearest neighbor density estimation

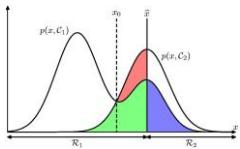


24

732A99/TDDE01

K-nearest neighbor density estimation

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

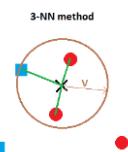


732A99/TDDE01

25

K-nearest neighbor classification

- Given N observations (X_j, Y_j)
 - $Y_j = C_i$, where C_1, \dots, C_m are possible class values
- Model assumptions
 - Apply K-NN density estimation:
$$p(X = x|Y = C_i) = \frac{K_i}{N_i V}, p(C_i) = \frac{N_i}{N}$$
 - V : volume of the sphere
 - K_i : #obs from training data of $Y = C_i$ in the sphere
 - N_i : #obs from training data of $Y = C_i$



732A99/TDDE01

26

Bayesian classification

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

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

$$p(C_i|x) = \frac{p(x|C_i)p(C_i)}{p(x)}$$
- We get

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

732A99/TDDE01

27

K-nearest neighbor classification

Algorithm

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

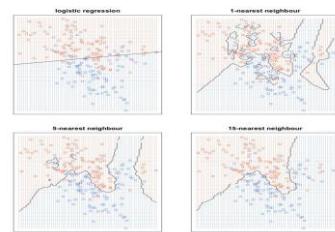
Majority voting: prediction for x is defined by majority voting of K neighbors

732A99/TDDE01

28

K-nearest neighbor example

Why classification results are so different for K-NN?

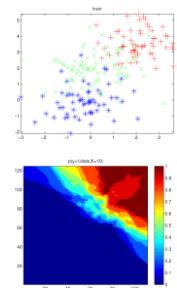


732A99/TDDE01

29

Model types

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

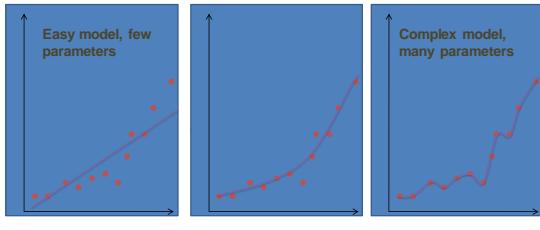


732A99/TDDE01

30

Overfitting

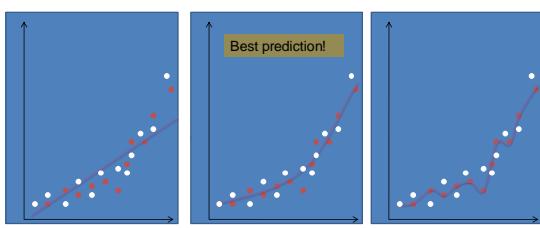
- Which model feels appropriate?



31

Overfitting

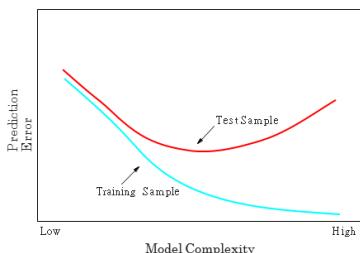
Now new data from the same process



32

Overfitting

- Observed:



33

Model selection

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

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

Comment:

- Approach works well for moderate/large data

34

Typical error functions

- Regression, **MSE** :

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

- Classification, **misclassification rate**

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

732A99/TDDE01

35

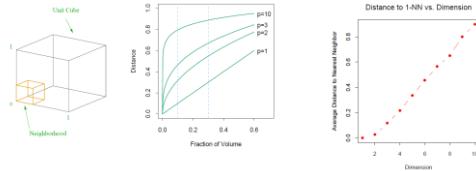
Curse of dimensionality

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

732A99/TDDE01

36

Curse of dimensionality



37

Curse of dimensionality

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

38

Basics of Statistics

Lecture 1b

39

Probability

How likely it is that some event will happen?

Idea:

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

40

Probability

Example: Tossing a coin two times



Example:

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

41

Properties and definitions

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

732499/TDDE01

42

42

Bayes theorem

Example:

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

43

732A99/TDDE01

43

Bayes theorem

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

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

44

732A99/TDDE01

44

Random variables

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

Examples:

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

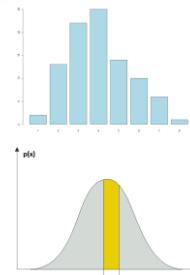
45

732A99/TDDE01

45

Distributions

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



46

732A99/TDDE01

46

Expected value and variance

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

47

732A99/TDDE01

47

Probabilities

- Laws of probabilities
 - Sum rule (compute marginal probability)

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

$$p(X) = \int p(X, Y)dY$$
 - Product rule

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

Combination 1:

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

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

48

732A99/TDDE01

48

Bayes theorem

For random variables:

Bayes Theorem

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

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

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



49

732A99/TDDE01

49

Some conventional distributions

Bernoulli distribution

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

$$\begin{aligned} - E(X) &= p \\ - Var(X) &= 1 - p \end{aligned}$$

Examples: Tossing coin, winning a lottery,..

50

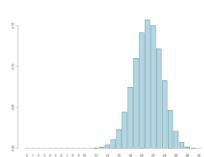
732A99/TDDE01

50

Some conventional distributions

Binomial distribution

- Sequence of n Bernoulli events
 - $X=(\text{Amount of successes among these events})$, $X=0, \dots, n$
- $$P(X=r) = \frac{n!}{(n-r)!r!} p^r (1-p)^{n-r}$$
- $EX = np$
 - $Var(X) = np(1-p)$



732A99/TDDE01

51

51

Poisson distribution

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

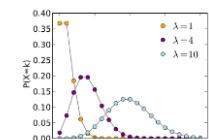
732A99/TDDE01

52

52

Poisson distribution

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



732A99/TDDE01

53

Poisson distribution

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

732A99/TDDE01

54

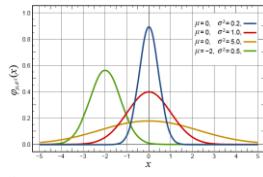
54

Normal distribution

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

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

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



732A99/TDDE01 55

55

Probabilistic models

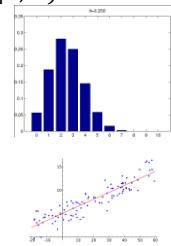
- A distribution $p(x|w)$ or $p(y|x, w)$

- Example:

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

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

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



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

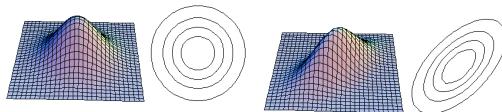
732A99/TDDE01

58

58

Multivariate distributions

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



732A99/TDDE01 56

56

Fitting a model

- Given dataset D and model $p(x|w)$ or $p(y|x, w)$

- **Frequentist approach:** which combination of parameter values fits my data best?

- **Bayesian approach:** parameters are random variables, all feasible values are acceptable
 - Different parameter values have different probabilities

732A99/TDDE01

59

59

Basic ML ingredients

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

Case	X_1	X_2	Y
1			
2			
...			

732A99/TDDE01 57

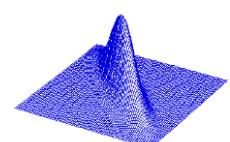
57

Fitting a model

- Frequentist principle: **Maximum likelihood** principle
 - Compute likelihood $p(D|w)$

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

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



- Maximize the likelihood and find the optimal w^*

732A99/TDDE01

60

60

Fitting a model

Remarks:

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

61

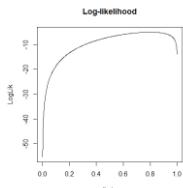
732A99/TDDE01

61

Fitting a model

Example: tossing a coin.

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



62

IE01

62

Bayesian probabilities

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

63

732A99/TDDE01

63

Fitting a model

• Bayesian principle

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

• Use bayes theorem

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

• $p(D)$ is **marginal likelihood**

- $p(D) = \int p(D|w)p(w)dw$ or
- $p(D) = \sum_i p(D|w_i)p(w_i)$

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

64

732A99/TDDE01

64

Fitting a model

• How to chose the prior?

- Expert knowledge about the phenomenon
- Forcing a model to have a certain structure
 - Example: decision trees: prior prefers smaller trees
- http://en.wikipedia.org/wiki/Conjugate_prior
- Conjugacy
 - Distribution of the posterior is the same type as the distribution of the likelihood or prior

• Prior is the most controversial about Bayesian methods, but

- When $N \rightarrow \infty$, data overwhelms the prior

65

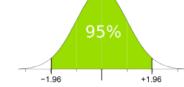
732A99/TDDE01

65

Measuring uncertainty

• Confidence interval (frequentist)

- Model $p(x|w)$ is known
- \hat{w} is a function of x by ML
- Derive distribution of \hat{w}
- Compute quantiles



• Credible interval (Bayes)

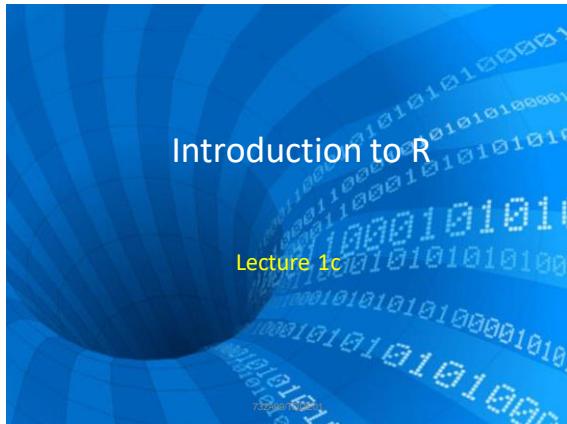
• Prediction interval (models)

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

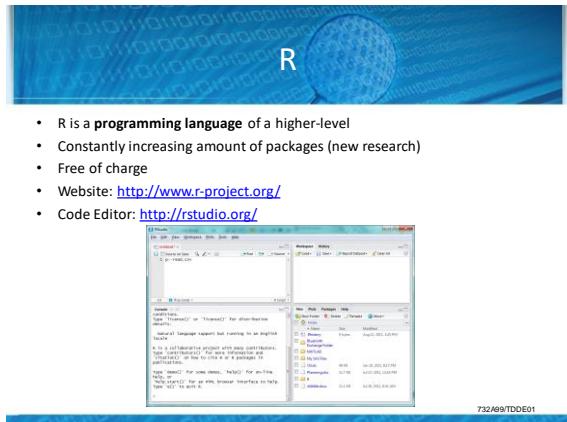
66

732A99/TDDE01

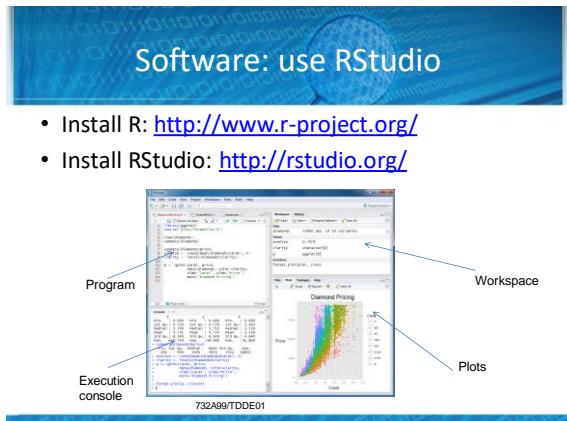
66



67



68



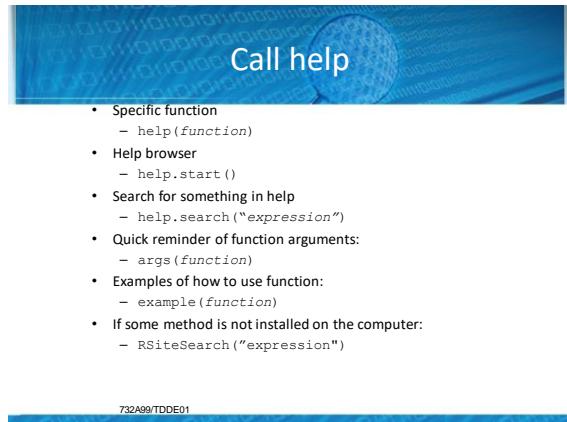
69

**Important to know:**

- Create a new file and save it (File menu)
- Running one line or entire code (Edit menu)
- Running one line in console
- Workspace (Observe, Save, Clear)
- Setting current directory (Tools)
- Installing new package (Packages tabs)

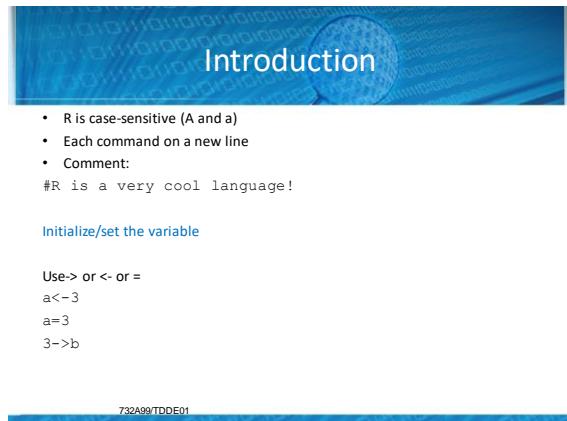
73A99/TDDE01

70



73A99/TDDE01

71



73A99/TDDE01

72

Vectors

- Create a vector
 $x<-c(1,3)$
 - See the result
 x
`print(x)`
- ```
> x<-c(1,3)
> x
[1] 1 3
> print(x)
[1] 1 3
```
- Create an empty vector  
 $y<-numeric(10)$   
 $y$
- ```
> y<-numeric(10)
> y
[1] 0 0 0 0 0 0 0 0 0 0
```

732A99/TDDE01

73

Matrices

Use `matrix()`

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

Values should be listed columnwise
 $nrow=$ and $ncol=$ can be skipped

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

- Create empty matrix

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

732A99/TDDE01

76

Sequence

- Either': ' or `seq()`

R R Console

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

732A99/TDDE01

74

Matrix operations

Usual vector operations can also be applied:

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

```
> m1<-matrix(c(1,2,0,1), nrow=2)
> m2<-matrix(c(2,2,5,1), nrow=2)
> m1
[1,1] [1,2]
[1,] 1 0
[2,] 2 1
> m2
[1,1] [1,2]
[1,] 2 5
[2,] 2 1
> m2*m1
[1,1] [1,2]
[1,] 2 0
[2,] 4 1
> |
```

732A99/TDDE

01

77

Operation with vectors

- indexing
 - Element-wise: $+$ - $*$ - $^{\wedge}$
 - log exp sin cos
 - length –number of elements
 - sum – sum of all elements
 - max min sort order
 - which.min which.max
- Logicals:**
 TRUE or FALSE:
 $A=TRUE;$
- ```
== > >= < <= != & (and) | (or)
```

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

732A99/TDDE
   
01

75

## Matrix operations

- Matrix operators/functions:

- transpose  $b=t(a)$

 $b = a^T$ 

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

 $b=solve(a)$ 

- Solve  $d=a^{-1}b$

 $d=solve(a,b)$ 

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

732A99/TDDE
   
01

78

## Indexing for matrices

- Positive index  
`x[1, 6] x[2:10, ]`
- Negative index  
`x[2, -(1:5)] row 2 and all columns except 1:5`
- Entire column or row  
`y=x[, 2] entire row 2`
- Extraction  
`> b  
[1] 1 4 -1 3 0  
> dmb(b>0)  
> d  
[1] 1 4 3`

79

## Replication

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

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

```

80

Matrix operations

- Dimension
`- dim(mat)`
 - Row/column statistics
`- colMeans, rowMeans, colSums, rowSums`
 - Apply a function over vector/matrix
`- Sapply()`
`- Normally used when function works only element-wise`
- ```

> m2
 [,1] [,2]
[1,] 3 3
[2,] 4 4
[3,] 3 3
[4,] 4 4
> ns=dim(m2)
> ns
[1] 2
> cm=colMeans(m2)
[1] 3.5
> cs=colSums(m2)
[1] 7.0
> rsums(m2)
[1] 3 7 11

```
- ```

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

```

81

Vector/matrix operations

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

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

```

82

## Factors

### Text values

```

> f1<-c("Man", "woman")
> f1
[1] "Man" "woman"
> f2=c("Man", "woman", "Man")
> f2
[1] "Man" "woman" "Man"
> F2
F2
 Man Woman
 2 1
> f3=factor(c(1,0,1,1,0), levels=c(0,1), labels=c("Man", "woman"))
> f3
[1] Woman Man Woman Woman Man
Levels: Man Woman

```

732A99/TDDE01

83

## Lists

### List is a collection of objects

```

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

```

732A99/TDDE01

84

## Data frame

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

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

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

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

732A99/TDDE01

85

## Data frame

- Any column in the data frame can be retrieved by `dataframe$object`

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

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

732A99/TDDE01

86

## Read data from Excel file

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

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

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

732A99/TDDE01

87

## Conversion between types

```
> v6<-c(1,4,2,2,1)
> #lets factor(v6)
> f6
[1] 1 4 2 2 1
Levels: 1 2 4
> x6<-c("1", "0", "1", "1", "1")
> x6
[1] "1" "0" "1" "1" "1"
> x6<-as.list(f6)
[1] "1"
[1] "0"
[1] "1"
[1] "1"
[1] "1"
> df1
[1] 1 4 2 2 1
[1] 1 4 2 2 1
[1] 1 4 2 2 1
[1] 1 4 2 2 1
[1] 1 4 2 2 1
> df1<-data.frame(x6)
> df2$X
[1,] 1 1
[2,] 2 2 0
[3,] 3 3 1
> m5<-as.matrix(df1)
> m5
 X Y
[1,] 1 1
[2,] 2 0
[3,] 3 1
> df3<-data.frame(m5)
> df3$X
[1] 1 2 3
> as.numeric(df3)
[1] 1 3
```

732A99/TDDE01

88

## Loops

```
for (name in expr1)
{
...
}
> for (i in 1:5) {
+ y<-seq(i,8)
+ print(y)
[1] 1 2 3 4 5 6 7 8
[1] 2 3 4 5 6 7 8
[1] 3 4 5 6 7 8
[1] 4 5 6 7 8
[1] 5 6 7 8
> |
```

732A99/TDDE01

89

## Conditioning and loops

```
if(x==3) {
...
}
else {
}
}
> m4<-matrix(c(1,2,0,1), nrow=2)
> m4
 [,1] [,2]
[1,] 1 0
[2,] 2 1
> n<-dim(m4)[1]
> i<-numeric(n)
> for (i in 1:n) {
+ if(max(m4[i,>1]) >i-1
+ +
+ i
[1] 0 1
while(x!=29) {
...
}
```

732A99/TDDE01

90

## Random number generation

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

732A09/TDDE01

91

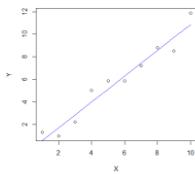
## Using a function

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

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

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

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



92

## Writing your own functions

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

```
> myfun <- function(x>25, y, z)
+ {
+ if(x)
+ z=y
+ else
+ z<-x+y
+ t
+ result<-log(z)*y
+ result
+ c<-sqrt(z), TRUE)
> x
[1] 3
> myfun(x>FALSE, y=0)
> x
[1] 0
> |
```

732A09/TDDE01

93

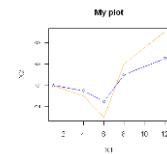
## Graphical procedures

### Some common procedures:

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

```
x<-c(1,4,7,8,10);
y<-c(4,3,1,6,9);
```

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



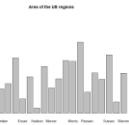
732A09/TDDE01

94

## Graphical parameters

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

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



732A09/TDDE01

95

## Graphical parameters

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

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



732A09/TDDE01

96

## Some more examples

- Dividing training/test

```

data=data.frame(X=c(1,1,2,2,3), Y=c("M","F","M","M","F"))
n=nrow(data)[1]
set.seed(123)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]

• Computing misclassification rate

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

 > X=c(1,1,1,2,3,1,2,2,1,1,3)
 > X1=c(2,3,2,1,2,3,1,2,2,1,1,1)
 > missclass(X,X1)
 [1] 0.2307692
}

```

732A99/TDDE01

97

## Regression and regularization

Lecture 1d

98

## Overview

- Linear regression
- Ridge Regression
- Lasso
- Variable selection

732A99/TDDE01

99

## 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

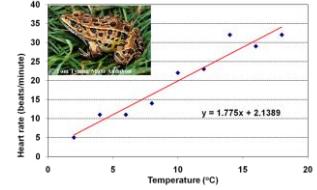
$$p(y|x, w) = N(w_0 + w_1 x, \sigma^2)$$

### Terminology:

 $w_0$ : intercept (or bias) $w_1$ : regression coefficient

### Response

The target responds directly and linearly to changes in the feature



732A99/TDDE01

100

100

## Ordinary least squares regression (OLS)

### Model:

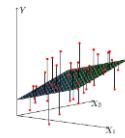
$$y \sim N(\mathbf{w}^T \mathbf{x}, \sigma^2)$$

where

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

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

Why is "1" here?



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

732A99/TDDE01

101

101

## Ordinary least squares regression

Given data set  $D$ 

| Case | $X_1$    | $X_2$    | $\vdots$ | $X_p$    | $Y$   |
|------|----------|----------|----------|----------|-------|
| 1    | $x_{11}$ | $x_{21}$ |          |          | $y_1$ |
| 2    | $x_{12}$ | $x_{22}$ |          |          | $y_2$ |
| 3    | $x_{13}$ | $x_{23}$ |          |          | $y_3$ |
| $N$  | $x_{1N}$ | $x_{2N}$ |          | $x_{pN}$ | $y_N$ |

Estimation: maximizing the likelihood

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

Is equivalent to minimizing

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

732A99/TDDE01

102

102

99

## Matrix formulation of OLS regression

Optimality condition:

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

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

732A99/TDDE01

103

103

## Parameter estimates and predictions

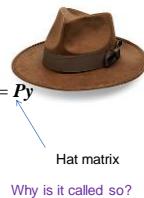
- Least squares estimates of the parameters

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

- Predicted values

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

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



732A99/TDDE01

104

104

## Degrees of freedom

Definition:

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

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

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

- For linear regression, degrees of freedom

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

732A99/TDDE01

105

105

## Different types of features

- Interval variables

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

- Dummy coded qualitative variables

### Example of dummy coding:

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

### 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:

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

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

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

732A99/TDDE01

106

106

## Basis function expansion

- In general  $\phi_1(\dots)$  may be a function of several  $x$  components

- Having data given by  $\mathbf{X}$ , compute new data

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

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

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

732A99/TDDE01

107

107

## 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
```

732A99/TDDE01

108

108

## An example of ordinary least squares regression

```

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

> summary(fit1)
Call:
lm(formula = Price ~ Year, data = mydata)

Residuals:
 Min 1Q Median 3Q Max
-167683 -16681 20056 35933 72317

Coefficients:
 Estimate Std. Error t value Pr(>|t|)
(Intercept) 8446038 8446038 -0.232 6.00e-13 ***
year 39246 4226 9.288 5.25e-13 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 57230 on 58 degrees of freedom
Multiple R-squared: 0.9997 , Adjusted R-squared: 0.9992
F-statistic: 86.26 on 1 and 57 DF, p-value: 5.248e-13

```

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

**Inputs:**  
 $X_1$  = Manufacturing year  
 $X_2$  = Mileage (km)  
 $X_3$  = Equipment (0 or 1)

109

109

## An example of ordinary least squares regression

```

> summary(fit2)
Call:
lm(formula = Price ~ Year + Mileage + Equipment, data = mydata)

Residuals:
 Min 1Q Median 3Q Max
-66223 -10325 14128 65332

Coefficients:
 Estimate Std. Error t value Pr(>|t|)
(Intercept) -2.038e-07 6.309e-06 -3.302 0.00169 **
year 1.062e+04 3.154e+03 3.366 0.00139 **
Mileage 5.790e+04 1.041e+04 5.563 8.08e-07 ***
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: 59270 on 55 degrees of freedom
Multiple R-squared: 0.8987 , Adjusted R-squared: 0.8982
F-statistic: 164.5 on 3 and 55 DF, p-value: < 2.2e-16

```

110

110

## An example of ordinary least squares regression

- Prediction

```

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

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

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

```

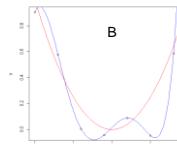
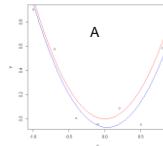
111

111

## Ridge regression

- Problem: linear regression can overfit:

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



732A99/TDDE01

112

112

## Ridge regression

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

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

### $\lambda_2$ regularization

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

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

- $\lambda > 0$  is penalty factor

732A99/TDDE01

113

113

## Ridge regression

### Equivalent form

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

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

### Solution

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

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

Hat matrix

How do we compute degrees of freedom here?

732A99/TDDE01

114

114

## Ridge regression

### Properties

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

115

732A99/TDDE01

115



Build model predicting performance

## Ridge regression

### Properties

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

116

732A99/TDDE01

116



Coefficients

Log Lambda

## Ridge regression

- Bayesian view
  - Ridge regression is just a special form of Bayesian Linear Regression with constant  $\sigma^2$ :
$$y \sim N(y | w_o + Xw, \sigma^2 I)$$

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

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

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

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

117

732A99/TDDE01

117

732A99/TDDE01

120

&gt; model\$lambda.min

[1] 0.046

&gt;

## Ridge regression

- How good is this model in prediction?

```
ind=sample(289, floor(289*0.5))
data1=scale(data[,3:9])
train=data1[1:ind]
test=data1[-1: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))
ytest[,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)
Note that data are so small so numbers
change much for other train/test

#Coeficient of determination
sum((ynew-mean(y))^2)/sum((y-mean(y))^2)
[1] 0.5438148
> sum((ynew-y)^2)
[1] 18.04988
> 1
```

732A99/TDDE01

121

121

## LASSO

- Idea:** Similar idea to Ridge
- Minimize minus loglikelihood plus linear penalty factor  $\rightarrow \ell_1$  regularization

- Given that model is Gaussian, we get LASSO (least absolute shrinkage and selection operator):

$$\hat{w}^{\text{Lasso}} = \underset{\lambda}{\operatorname{argmin}} \left\{ \sum_{j=1}^N (y_j - w_0 - w_1 x_{1j} - \dots - w_p x_{pj})^2 + \lambda \sum_{j=1}^p |w_j| \right\}$$

- $\lambda > 0$  is penalty factor



732A99/TDDE01

122

122

## LASSO

- Equivalently

$$\hat{w}^{\text{Lasso}} = \underset{s}{\operatorname{argmin}} \sum_{j=1}^N (y_j - w_0 - w_1 x_{1j} - \dots - w_p x_{pj})^2 \\ \text{subject to } \sum_{j=1}^p |w_j| \leq s$$

732A99/TDDE01

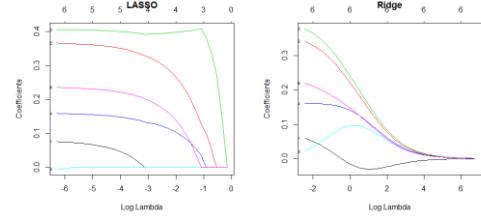
123

123

## LASSO vs Ridge

- LASSO yields sparse solutions!

**Example** Computer hardware data



732A99/TDDE01

124

124

## LASSO vs Ridge

- Only 5 variables selected by LASSO

```
> coef(model, s="lambda.min")
7 x 1 sparse Matrix of class "dgCMatrix"
(Intercept) -0.091825e-17
V3 6.350488e-02
V4 3.578607e-01
V5 4.03367e-01
V6 1.541329e-01
V7 2.287134e-01
V8 0.5826904
> sum((ynew-mean(y))^2)/sum((y-mean(y))^2)
[1] 0.5826904
> sum((ynew-y)^2)
[1] 16.63756
```

732A99/TDDE01

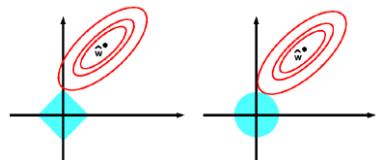
125

125

## LASSO vs Ridge

- Why Lasso leads to sparse solutions?

- Feasible area for Ridge is a circle (2D)
- Feasible area for LASSO is a polygon (2D)



732A99/TDDE01

126

126

## LASSO properties

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

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

- No explicit formula for  $\hat{w}^{\text{Lasso}}$ 
  - Optimization algorithms used

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

732499/TDDE01

127

127

## 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

732499/TDDE01

128

128

## Variable selection

### Alternative 1: Variable subset selection

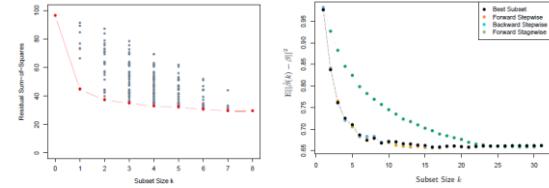
- Best subset selection:
  - Consider different subsets of the full set of features, fit models and evaluate their quality
    - Problem: computationally difficult for  $p$  around 30 or more
    - How to choose the best model size? Some measure of predictive performance normally used (ex. AIC).
- Forward 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")

732499/TDDE01

129

129

## RSS and MSE depend on k



732499/TDDE01

130

130

## 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.frame(data1))

Residuals:
 Min 1Q Median 3Q Max
-1.20332 -0.15512 0.03579 0.18567 2.42280

Coefficients:
 Estimate Std. Error t value Pr(>|t|)
(Intercept) -5.735e-17 2.574e-02 0.000 1.0000
V3 3.651e-01 4.312e-02 8.490 4.34e-15 ***
V4 4.950e-01 4.312e-02 11.470 3.67e-26 ***
V5 1.951e-01 3.394e-02 4.687 3.07e-06 ***
V6 2.360e-01 3.394e-02 7.011 3.06e-11 ***
V8 9.840 38.101 -345.74 <2e-16 ***
V9 10.4713 38.388 -343.58 <2e-16 ***

Step: AIC=-407.25
Step: AIC=-407.25
> fit <- stepAIC(fit, direction="both")
start <- AIC=-405.35
V9 ~ V3 + V4 + V5 + V6 + V7 -405.35
- V7 1 0.0139 28.103 -405.35
<none> 1 1.0819 29.183 -399.46
- V3 1 2.9180 31.041 -386.37
- V5 1 3.1550 31.160 -387.77
- V6 1 6.8472 34.964 -363.70
- V4 1 6.7492 37.852 -345.11
- V5 1 10.4837 38.586 -341.99
Step: AIC=-407.25
Step: AIC=-407.25
> fit <- stepAIC(fit, direction="both")
start <- AIC=-405.35
V9 ~ V3 + V4 + V5 + V6 + V8
<none> 1 0.0139 28.317 -405.35
- V7 1 1.0958 29.193 -405.26
- V6 1 3.1550 31.160 -387.77
- V8 1 6.8472 34.964 -363.70
- V4 1 6.9810 38.101 -345.74
- V5 1 10.4713 38.388 -343.58
```

732499/TDDE01

131

131

## Model selection

### Lecture 1:

132

## Overview

- Model fitting
- Model selection

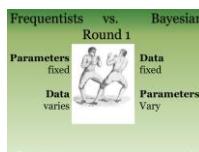
732A99/TDDE01

133

133

## 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?

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

732A99/TDDE01

134

134

## An estimator

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

**Example:** Linear regression

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

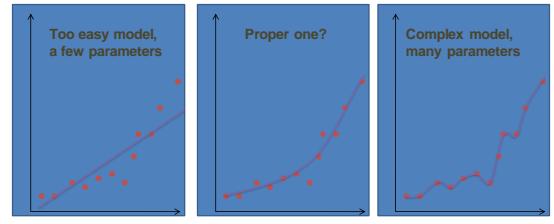
732A99/TDDE01

135

135

## Overfitting

- Complex model can overfit your data



732A99/TDDE01

136

136

## 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
  - Holdout method
  - Cross-validation

732A99/TDDE01

137

137

## Model selection

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

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

732A99/TDDE01

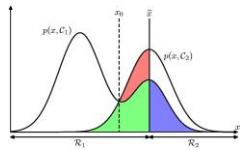
138

138

## Model selection

**Example:** Spam classification

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



139

732A99/TDDE01

139

## Model selection

- Problem:** true model and true  $w$  are unknown → can not compute expected loss!

- How to find an optimal model?

- Consider what expected loss (**risk**) depends on  $R(Y, \hat{y}) = E[L(Y, \hat{y}(X, D))]$

- Random factors:

- $D$  – **training set**
- $Y, X$  – data to be predicted (**validation set**)

142

142

## Loss functions

- How to define loss function?

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

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

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

**Example:** Compute loss function related to

- Normal distribution

Guess why such loss function was chosen

140

732A99/TDDE01

140

## Loss functions

- Classification problems

- Common loss function  $L(Y, \hat{y}) = \begin{cases} 0, & Y = \hat{y} \\ 1, & Y \neq \hat{y} \end{cases}$

- When minimizing the loss, equivalent to misclassification rate

141

732A99/TDDE01

141

## 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**)

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

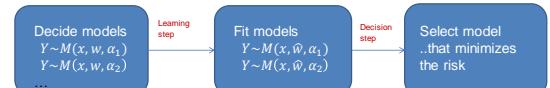
- Estimator is fit by Maximum Likelihood using training set
- Risk estimated by using validation set
- Model with minimum empirical risk is selected

143

143

## General model selection strategy

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



- When fitting data, Maximum Likelihood is usually used

- $\alpha_l$  can be different things:

- Type of distribution
- Number of variables in the model
- Regularization parameter value
- ...

144

144

732A99/TDDE01

## Holdout method

Divide into training, validation and test sets

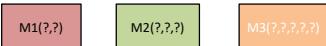


- Choose proportions in some way

145

## Holdout method

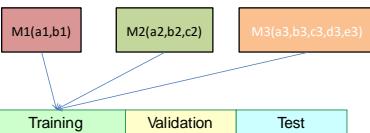
- Given: training, validation, test sets and models to select between



146

## Holdout method

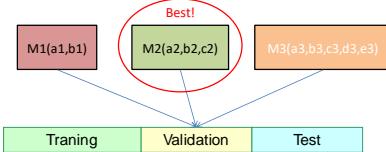
- Training set is used for fitting models to the dataset by using maximum likelihood



147

## Holdout method

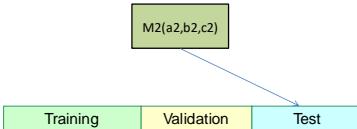
- Validation set is used to choose the best model (lowest risk)



148

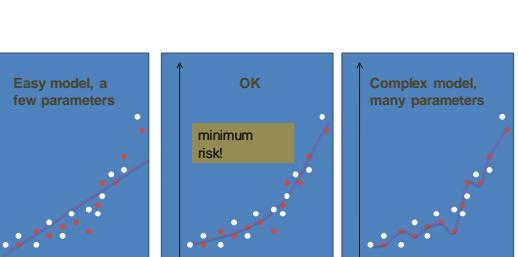
## Holdout method

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



149

## Holdout method



150

## Holdout in R

- How to partition into train/test?

– Use `set.seed(12345)` in the labs to get identical results

```
random(data)[1]
set.seed(12345)
id1=sample(1:n, floor(n*0.7))
traindata[id,]
testdata[-id,]
```

- How to partition into train/valid/test?

```
random(data)[1]
set.seed(12345)
id1=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,]
```

732A99/TDDE01

151

151

## Bias-variance tradeoff

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

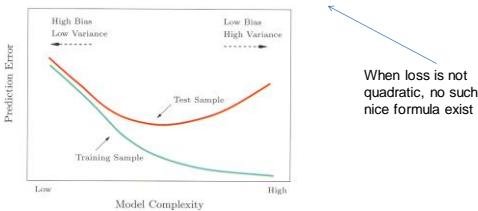
732A99/TDDE01

152

152

## Bias-variance tradeoff

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



732A99/TDDE01

153

153

## Cross-validation

- Compared to holdout method:

– Why do we use only some portion of data for training- can we use more (increase accuracy)?

### Cross-validation (Estimates Err)

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

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

732A99/TDDE01

154

154

## Cross-validation

### Cross-validation



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

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

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

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

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

732A99/TDDE01

155

155

## Cross-validation vs Holdout

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

732A99/TDDE01

156

156

## Analytical methods

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

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

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

157

732A99/TDDE01 157

## Analytical methods

- One can show that
 
$$R_{in}(Y, \hat{Y}) \approx R_{train} + \frac{2}{N} \sum_i cov(\hat{y}_i, Y_i)$$
 where  $R_{train} = \frac{1}{N} \sum_{X_i, Y_i \in T} L(Y_i, \hat{Y}_i)$
- Recall, **degrees of freedom**  $df(model) = \frac{1}{\sigma^2} \sum_i cov(\hat{y}_i, Y_i)$ 
  - When model is linear,  $df$  is the number of parameters.
- If loss is defined by minus two loglikelihood,
 
$$AIC \equiv -2loglik(D) + 2df(model)$$

158

732A99/TDDE01 158

## Model selection

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

- Cycle time
- Memory
- Channels
- ...

Build model predicting performance



159

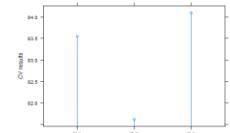
732A99/TDDE01 159

## Cross-validationat

- 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, ydata$V9, data=data,K=10,
foldType="consecutive")
f2=cvFit(fit2, ydata$V9, data=data,K=10,
foldType="consecutive")
f3=cvFit(fit3, ydata$V9, data=data,K=10,
foldType="consecutive")
res=cvSelect(f1,f2,f3)
plot(res)
```



160

732A99/TDDE01 160

## Linear classification methods

Lecture 2a

161

732A99/TDDE01

162

## Overview

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

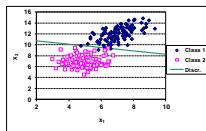
## Classification

- Given data  $D = \{(X_i, Y_i), i = 1 \dots N\}$

- $- Y_i = Y(X_i) = C_j \in \mathcal{C}$
- $- \text{Class set } \mathcal{C} = (C_1, \dots, C_K)$

**Classification problem:**

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



732A99/TDDE01

163

163

## Classifiers

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

**Disadvantages of deterministic classifiers:**

- Sometimes simple mapping is not enough (risk of cancer)
- Difficult to embed loss  $\rightarrow$  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  $\rightarrow$  ???
  - $P(\text{Spam} | A) = 0.99, P(\text{Spam} | B) = 0.45 \rightarrow$  better decision can be made

732A99/TDDE01

164

164

## Bayesian decision theory

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

732A99/TDDE01

165

165

## Loss matrix

- Costs of classifying  $Y = C_k$  to  $C_j$ :**

- Rows: true, columns: predicted

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

- Example 1: 0/1-loss**

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

- Example 2: Spam**

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

732A99/TDDE01

166

166

## Loss and decision

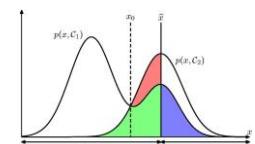
- Expected loss minimization

- $R_j : \text{classify to } C_j$

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

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

- Two classes



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

732A99/TDDE01

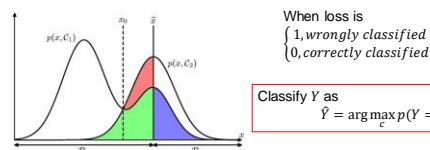
167

167

## Loss and decision

- Loss minimization

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



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

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

732A99/TDDE01

168

168

## Loss and decision

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

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

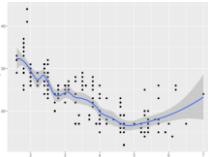
732A99/TDDE01

169

169

## Loss and decision

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



732A99/TDDE01

170

170

## ROC curves

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

### Confusion matrix

|   |   | PREDICTED |    | Total |
|---|---|-----------|----|-------|
|   |   | 1         | 0  |       |
| T | 1 | TP        | FN | $N_+$ |
|   | 0 | FP        | TN | $N_-$ |

732A99/TDDE01

171

171

## ROC curves

- True Positive Rates (TPR) = sensitivity = recall**  
– Probability of detection of positives: TPR=1 positives are correctly detected  
 $TPR = TP/N_+$
- False Positive Rates (FPR)**  
– Probability of false alarm: system alarms (1) when nothing happens (true=0)  
 $FPR = FP/N_-$
- Specificity**  
 $Specificity = 1 - FPR$
- Precision**  
 $Precision = \frac{TP}{TP + FP}$

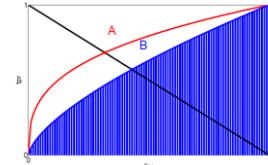
732A99/TDDE01

172

172

## ROC curves

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



732A99/TDDE01

173

173

## Types of supervised models

- Generative models:** model  $p(X|Y, w)$  and  $p(Y|w)$   
– Example: k-NN classification  
 $p(X = x|Y = C_i, K) = \frac{K_i}{N_i V}, p(C_i|K) = \frac{N_i}{N}$
- From Bayes Theorem,  
 $p(Y = C_i|x, K) = \frac{K_i}{K}$
- Discriminative models:** model  $p(Y|X, w)$ ,  $X$  constant  
– Example: logistic regression  
 $p(Y = 1|w, x) = \frac{1}{1 + e^{-w^T x}}$

732A99/TDDE01

174

174

## Generative vs Discriminative

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

732A99/TDDE01

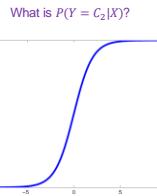
175

175

## Logistic regression

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

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



732A99/TDDE01

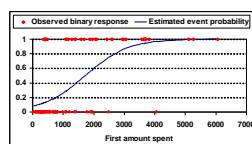
176

176

## Logistic regression

- Logistic model- yet another form
 
$$\ln \frac{p(Y = 1|X = x)}{p(Y = 0|X = x)} = \ln \frac{p(Y = 1|X = x)}{1 - p(Y = 1|X = x)} = \text{logit}(p(Y = 1|X = x)) = w^T x$$
- Here  $\text{logit}(t) = \ln \left( \frac{t}{1-t} \right)$
- Note  $p(Y|X)$  is connected to  $w^T x$  via logit link

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



732A99/TDDE01

177

177

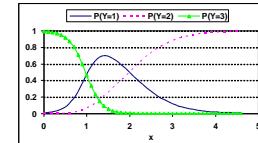
## Logistic regression

- When  $Y$  is categorical,

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

- Alternatively

$$Y \sim \text{Multinomial} \left( \text{softmax}(w_1^T x), \dots, \text{softmax}(w_K^T x) \right)$$



732A99/TDDE01

178

178

## Logistic regression

### Fitting logistic regression

- In binary case,
 
$$\log P(D|w) = \sum_{i=1}^N y_i \log(\text{sigm}(w^T x_i)) + (1 - y_i) \log(1 - \text{sigm}(w^T x_i))$$
  - Can not be maximized analytically, but unique maximizer exists
- To maximize loglikelihood, optimization used
  - Newton's method traditionally used (Iterative Reweighted Least Squares)
  - Steepest descent, Quasi-newton methods...

### Estimation:

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

Decision boundaries of logistic regression are linear

732A99/TDDE01

179

179

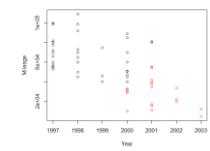
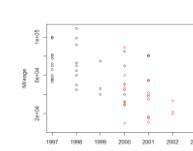
## Logistic regression

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

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

Original data

Classified data



732A99/TDDE01

180

180

## Quadratic discriminant analysis

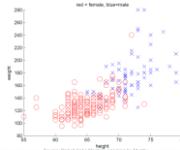
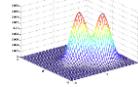
- Generative classifier

- Main assumptions:

$x$  is now random as well as  $y$

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

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



Source: Probabilistic Machine Learning by Murphy

r32A99/TDDE01

181

## Linear discriminant analysis (LDA)

- Difference LDA vs logistic regression??

Coefficients will be estimated differently! (models are different)

- How to estimate coefficients

find MLE.

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

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

Sample mean and sample covariance are MLE!

If class priors are parameters (**proportional priors**),

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

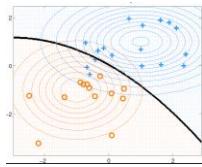
732A99/TDDE01

184

## Quadratic discriminant analysis

- If parameters are estimated, classify:

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



Source: Probabilistic Machine Learning by Murphy

732A99/TDDE01

182

182

## LDA and QDA: code

- Syntax in R, library MASS

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

Prior – class probabilities

Subset – indices, if training data should be used

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

predict(..)

732A99/TDDE01

185

185

## Linear discriminant analysis (LDA)

- Assumption  $\Sigma_i = \Sigma, i = 1, \dots, K$

- Then  $p(y = c_i | \mathbf{x}) = \text{softmax}(\mathbf{w}_i^T \mathbf{x} + w_{0i}) \rightarrow$  exactly the same form as the logistic regression

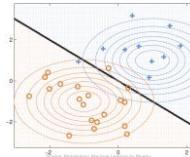
$$-\mathbf{w}_{0i} = -\frac{1}{2} \mathbf{\mu}_i^T \Sigma^{-1} \mathbf{\mu}_i + \log \pi_i$$

$$-\mathbf{w}_i = \Sigma^{-1} \mathbf{\mu}_i$$

- Decision boundaries are linear

– **Discriminant function:**

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



732A99/TDDE01

183

183

## 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
```

732A99/TDDE01

186

186

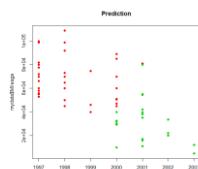
## LDA: output

- Misclassified items

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

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

|    |    |
|----|----|
| 0  | 1  |
| 31 | 6  |
| 7  | 15 |



732A99/TDDE01

187

## Naïve Bayes classifiers Decision trees

### Lecture 2b

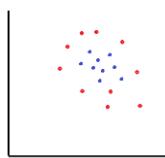
732A99/TDDE01

190

187

## 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



732A99/TDDE01

188

188

## LDA versus Logistic regression

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

732A99/TDDE01

189

189

## Naïve 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



732A99/TDDE01

191

191

## Naïve 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

732A99/TDDE01

192

192

## Naive Bayes classifiers: motivation

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

732A99/TDDE01

193

193

## Naive Bayes classifiers

- Naive Bayes assumption: **conditional independence**

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

- How many parameters now?

$$- P(X_i = x_i | Y = y), i = 1, \dots, p, x_i = \{0, 1\}, y = \{0, 1\} \quad 2 * p$$

- Is Naive Bayes assumption always valid?

$$- P(\text{Space, ship} | \text{SciFi}) = P(\text{Space} | \text{SciFi}) * P(\text{Ship} | \text{SciFi}) ?$$

194

732A99/TDDE01

194

## Naive Bayes classifiers - discrete inputs

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

732A99/TDDE01

195

195

## Naive Bayes classifiers - discrete inputs

### • Example Loan decision

– Classify a person: Home Owner=No, Single=Yes

| Tid | Home Owner | Marital Status | Annual Income | Defaulter Borrower |
|-----|------------|----------------|---------------|--------------------|
| 1   | Yes        | Single         | 125K          | No                 |
| 2   | No         | Married        | 100K          | No                 |
| 3   | Yes        | Single         | 70K           | No                 |
| 4   | No         | Married        | 130K          | 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                |

732A99/TDDE01

196

196

## Naive Bayes – continuous inputs

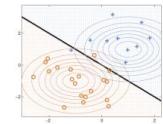
- $X_i$  are continuous

- **Assumption A:**  $x_j | y = C$  are univariate Gaussian

- $p(x_j | y = C_i, \theta) = N(x_j | \mu_{ij}, \sigma_{ij}^2)$

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

- $\Sigma_i = \text{diag}(\sigma_{11}^2, \dots, \sigma_{pp}^2)$



- **Naive bayes is a special case of LDA (given A)**

– → MLE are means and variances (per class)

732A99/TDDE01

197

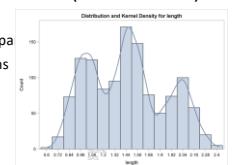
197

## Naive Bayes – continuous inputs

- **Assumption B:**  $p(x_j | y = C)$  are unknown functions of  $x_j$  that can be estimated from data

– Nonparametric density estimation (kernel for ex.)

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



732A99/TDDE01

198

198

## Naive Bayes in R

- naiveBayes in package **e1071**

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

```
library(MASS)
library(e1071)
n.dim(housing)[1]
ind=rep(1:n, housing[,5])
housing$housing[ind,-5]
> table(Yfit,housing$Sat)

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

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

732A99/TDDE01

199

199

## Decision trees

### Idea

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

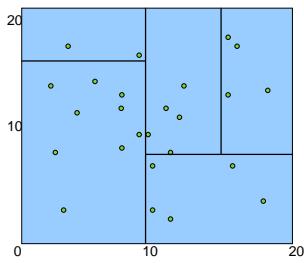
- Regression trees:
  - Target is a continuous variable
- Classification trees
  - Target is a class (qualitative) variable

732A99/TDDE01

200

200

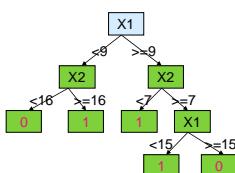
## Classification tree toy example



732A99/TDDE01

201

201



- Root node

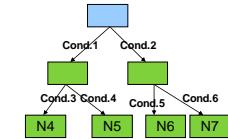
- Nodes

- Leaves (terminal nodes)

- Parent node, child node

- Decision rules

- A value is assigned to the leaves

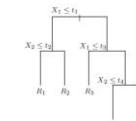
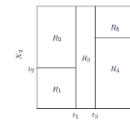


732A99/TDDE01

202

202

## Regression tree toy example



732A99/TDDE01

203

203

## A classification problem

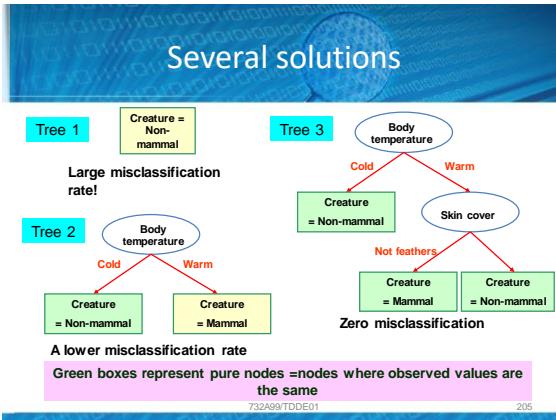
Create a classification tree that would describe the following patterns

| 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     | quills     | no          | no               | no              | yes      | no         | 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  |

732A99/TDDE01

204

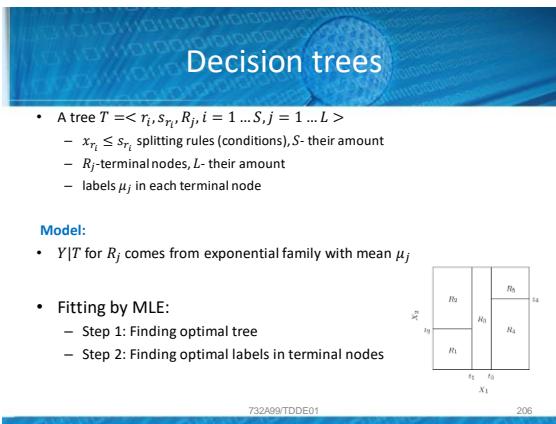
204



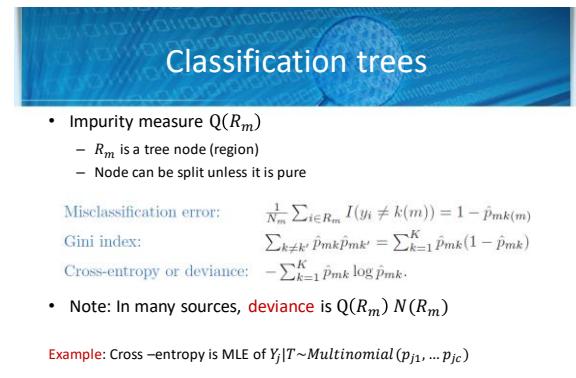
205



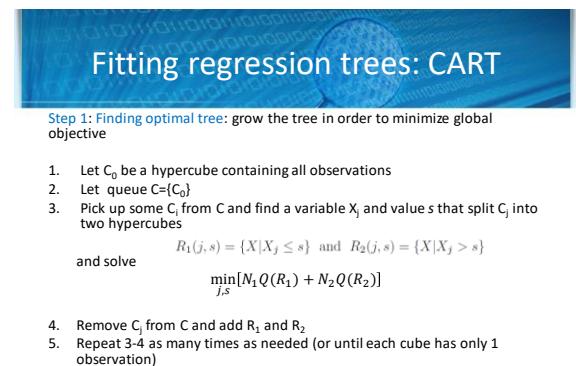
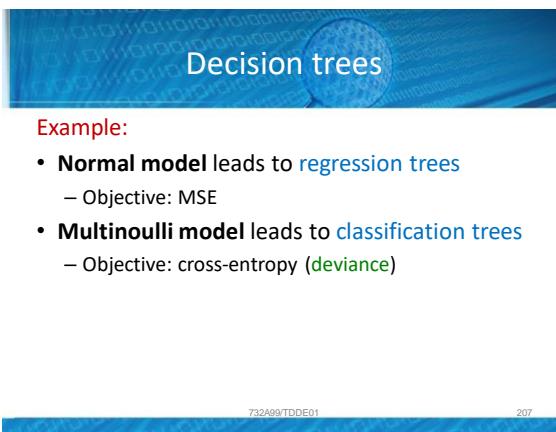
208



206



209



210



## Decision trees in R

- Misclassification results

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

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

217

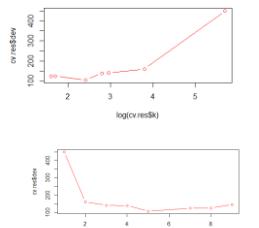
217

## Decision trees in R

- Selecting optimal tree by penalizing

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

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



What is optimal number of leaves?

218

732A99/TDD\_UvL

218

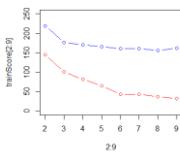
## Decision trees in R

- 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="pred")
 trainScore[i]=deviance(prunedTree)
 testScore[i]=deviance(pred)
}
plot(2:9, trainScore[2:9], type="b", col="red",
ylim=c(0,250))
points(2:9, testScore[2:9], type="b", col="blue")
```



What is optimal number of leaves?

219

732A99/TDD\_UvL

219

## Decision trees in R

- Final tree: 5 leaves

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

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

220

732A99/TDD\_UvL

220

220

## Generalized Linear Models. Uncertainty estimation

### Lecture 2c

221

732A99/TDD\_UvL

## Moving beyond typical distributions

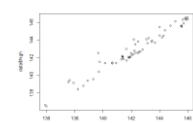
- We know how to model

- Normally distributed targets → linear regression
- Bernoulli and Multinomial targets → logistic regression
- What if target distribution is more complex?

#### Example 1: Daily Stock prices NASDAQ

- Open
- High (within day)

Does it seem that the error is normal here?



#### Example 2: Number of calls to bank

- Y=Number of calls
- X=time

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

222

732A99/TDD\_UvL

222

## Exponential family

- More advanced error distributions are sometimes needed!
  - Many distributions belong to **exponential** family:
    - Normal, Exponential, Gamma, Beta, Chi-squared..
    - Bernoulli, Multinoulli, Poisson...
- $$p(\mathbf{x}|\boldsymbol{\eta}) = h(\mathbf{x})g(\boldsymbol{\eta})e^{(\boldsymbol{\eta}^T u(\mathbf{x}))}$$
- Easy to find MLE and MAP
  - Non-exponential family distributions: uniform, Student t

**Example:** Bernoulli

732A99/TDDE01

223

223

## Generalized linear models

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

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

- If  $f$  is a canonical link, then

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

732A99/TDDE01

224

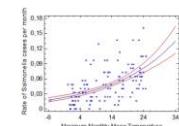
224

## Generalized linear models

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

**Example:** Poisson regression

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



732A99/TDDE01

225

225

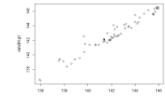
## Generalized linear model: software

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

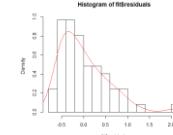
**Example:** Daily Stock prices NASDAQ

- Open
- High (within day)

- Try to fit usual linear regression, study histogram of residuals



Gamma distribution: Wikipedia



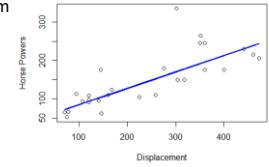
732A99/TDDE01

226

226

## Least absolute deviation regression

- Model  $Y \sim Laplace(w^T X, b)$ 
  - Member of exponential family
- Equivalent to minimizing sum of absolute deviations
- Properties
  - Robust to outliers
  - Sensitive to changes in data
  - Multiple solutions possible
- R: package `L1pack`



732A99/TDDE01

227

227

## 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  $\rightarrow$  find  $w$

732A99/TDDE01

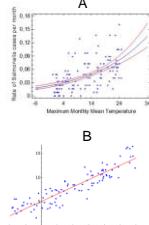
228

228

## 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..." 😊



732A99/TDDE01

229

229

## Probabilistic models

### Probabilistic model

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

- 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

732A99/TDDE01

230

230

## Uncertainty estimation

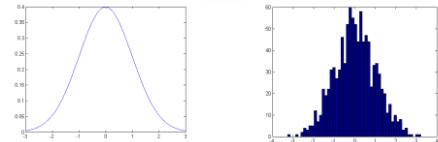
- Given estimator  $\hat{f} = \hat{f}(x, D)$  (or  $\hat{\alpha} = \delta(D)$ ), how to estimate the uncertainty?
- Answer 1:** if the distribution for data  $D$  is given, compute analytically the distribution for the estimator → derive confidence limits
  - Often difficult
  - Example:** In simple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

732A99/TDDE01

231

231

## The bootstrap: general principle



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

- Generate many different  $D_i$  from their distribution
- 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)

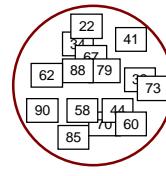
732A99/TDDE01

232

232

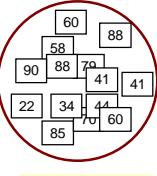
## Nonparametric bootstrap

### Observed data



$$\bar{x}$$

### Resampled data



Sampling with replacement

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

732A99/TDDE01

233

233

## Nonparametric bootstrap

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

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

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

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

732A99/TDDE01

234

234

## Parametric bootstrap

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

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

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

Parametric bootstrap is more precise if the distribution form is correct

235

## Uncertainty estimation

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

236

## Bootstrap confidence intervals

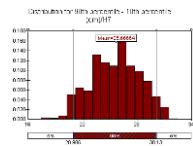
- To estimate  $100(1-\alpha)$  confidence interval for  $w$

### Bootstrap percentile method

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

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

Look at the plot...



237

## Bootstrap: regression context

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

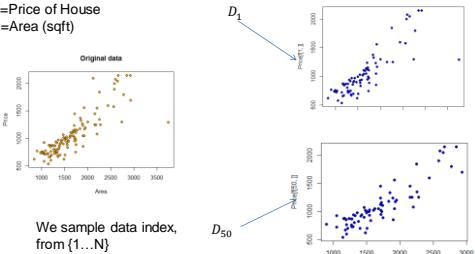
### Nonparametric bootstrap:

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

238

## Uncertainty estimation

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



732A99/TDDE01

239

239

## Bootstrap: regression context

### Parametric bootstrap

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

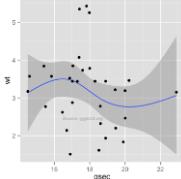
732A99/TDDE01

240

240

## Confidence intervals in regression

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



732A99/TDDE01

241

241

## Confidence intervals in regression

### Estimation

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

732A99/TDDE01

242

242

## Bootstrap: R

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

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

732A99/TDDE01

243

243

## Bootstrap: R

### Nonparametric bootstrap:

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

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

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

732A99/TDDE01

244

244

## Bootstrap: R

### Parametric bootstrap:

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

732A99/TDDE01

245

245

## Bootstrap

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

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

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

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

732A99/TDDE01

246

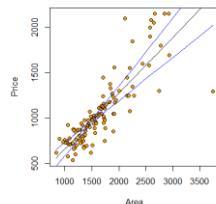
246

## Uncertainty estimation: R

- Bootstrap confidence bands for linear model

```
e=envelope(res) #compute confidence bands
fit=lm(Price~Area, data=data2)
predict= predict(fit)

plot(Area, Price, pch=21, bg="orange")
points(data2$Area,priceP,type="T") #plot fitted line
#plot confidence bands
points(data2$Area,e$point[2], type="T", col="blue")
points(data2$Area,e$point[1], type="T", col="blue")
```



732A99/TDDE01

247

## Lecture 2d

### Latent variable models

250

247

## Prediction bands

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

$$Y \sim \text{Distribution}(x, w)$$

#### Prediction band for parametric bootstrap

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

732A99/TDDE01

248

248

732A99/TDDE01

251

## Overview

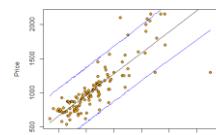
- Principal Component Analysis (PCA)
- Probabilistic PCA
- Independent component analysis (ICA)

## 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?

732A99/TDDE01

249

249

732A99/TDDE01

252

## 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
  - $100 \times 100 \times 5 = 50000$
  - $100 \times 100 + 2 \times 5 + 2 \times 5 = 10020$

3 | 3 | 3 | 3 | 3

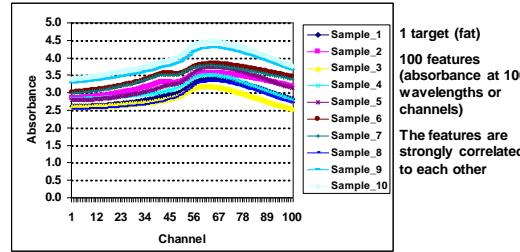
253

732A99/TDDE01

253

Absorbance records for ten samples of chopped meat

Parallel coordinate plot for "FAT"



732A99/TDDE01

256

256

## Principal Component Analysis (PCA)

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

### Applications

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

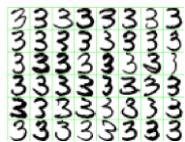
254

732A99/TDDE01

254

## Principal Component Analysis (PCA)

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



255

732A99/TDDE01

255

## Principal components analysis

Idea: Introduce a new coordinate system ( $PC_1, PC_2, \dots$ ) where

- The first principal component ( $PC_1$ ) is the direction that maximizes the variance of the projected data
- The second principal component ( $PC_2$ ) is the direction that maximizes the variance of the projected data after the variation along  $PC_1$  has been removed
- The third principal component ( $PC_3$ ) is the direction that maximizes the variance of the projected data after the variation along  $PC_1$  and  $PC_2$  has been removed
- ...

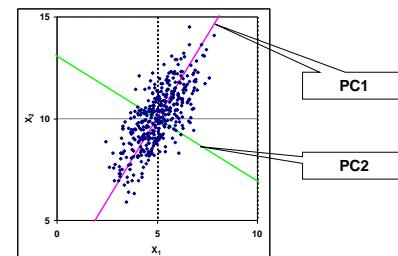
In the new coordinate system, coordinates corresponding to the last principal components are very small → can take away these columns

257

732A99/TDDE01

257

### Principal Component Analysis - two inputs



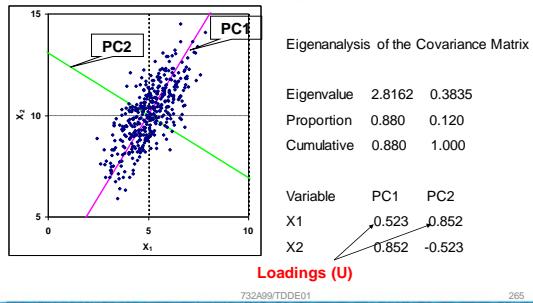
258

732A99/TDDE01

258



## Principal Component Analysis



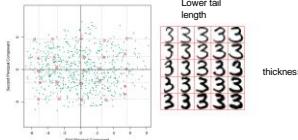
265

## Principal Component Analysis

- Digits: two eigenvectors extracted

$$\mathbf{x} = \boxed{3} + z_1 \cdot \boxed{3} + z_2 \cdot \boxed{3}$$

- Interpretation of eigenvectors



266

## PCA in R

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

```
mydata=read.csv("tecator.csv")
data1=mysite
data1$fat=0
res=prcomp(data1,center=TRUE)
lambda=res$dev^2
#eigenvalues
lambda
#percentage of variation
sprintf("%2.2f",lambda/sum(lambda)*100)
screeplot(res)
```

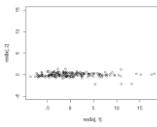
Only 1 component captures the 99% of variation!

267

## PCA in R

- Principal component loadings (U)

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



- Data in (PC1, PC2) - scores (Z)

Do we need second dimension?

732A99/TDDE01

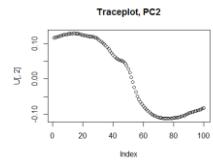
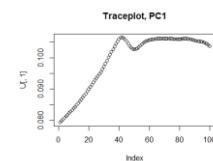
268

268

## PCA in R

- Trace plots

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



Which components contribute to PC1-2?

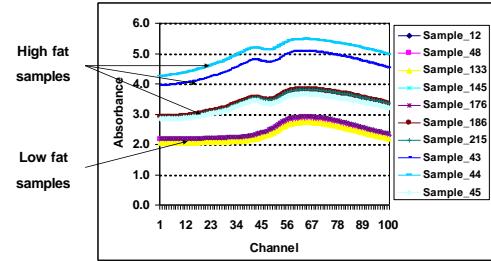
732A99/TDDE01

269

269

## Absorbance records for ten samples of chopped meat

- PCA2 captures the most of remaining variation



732A99/TDDE01

270

270

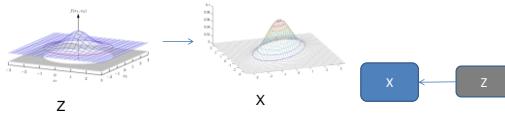
## Probabilistic PCA

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

$$z \sim N(0, I)$$

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

$$z \sim N(0, I), x = \mu + Wz + \epsilon, \epsilon \sim N(0, \sigma^2 I)$$
- Interpretation:** Observed data (X) is obtained by rotation, scaling and translation of standard normal distribution (Z) and adding some noise.



732A99/TDDE01

271

271

## Probabilistic PCA

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

$$x \sim N(\mu, C)$$

$$C = WW^T + \sigma^2 I$$
- Rotation invariance
  - Assume that  $x$  was generated from  $z' = Rz, RR^T = I$ ,  $p(x)$  does not change!
  - $x | z' \sim N(x | Wz' + \mu, \sigma^2 I)$
  - Model will not be able find latent factors uniquely!** ⓘ
    - It does not distinguish  $z$  from  $z'$

732A99/TDDE01

272

272

## Probabilistic PCA

- Estimation of parameters: ML

**Theorem.** ML estimates are given by

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

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

732A99/TDDE01

273

273

## Probabilistic PCA

- Estimation of  $Z$** 
  - Use mean of posterior  
 $\hat{z} = (W_{ML}^T W_{ML} + \sigma_{ML}^2 I)^{-1} W_{ML}^T (x - \mu)$
- Connection to standard PCA**
  - Assume  $R = I, \sigma^2 = 0 \rightarrow$  get standard PCA components scaled by inverse root of eigenvalues
$$Z = XUL^{-\frac{1}{2}}$$

732A99/TDDE01

732A99/TDDE01

274

274

## Advantages of probabilistic PCA

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

732A99/TDDE01

275

275

## Probabilistic PCA in R

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

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

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

732A99/TDDE01

276

276

## Independent component analysis (ICA)

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

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

732499/TDDE01

277

277

## ICA

- Model

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

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

- Assuming noise-free  $x$

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

Subject to  $\|v_i\| = 1$

732499/TDDE01

278

278

## ICA: estimation algorithm

- Estimate  $V$  by maximum likelihood
- Compute  $Z = X'V$

- With prewhitening**

- Convert  $X$  into PCA coordinate system (do not remove dimensions):  $X' = XU$
  - Estimate  $V$  by maximum likelihood in ICA
  - Estimate final scores  $Z = X'V$
- Note: full transformation matrix is  $U_{ICA} = U \cdot V$

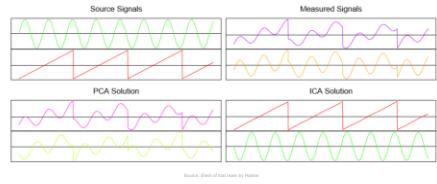
732499/TDDE01

279

279

## ICA

- Example**



732499/TDDE01

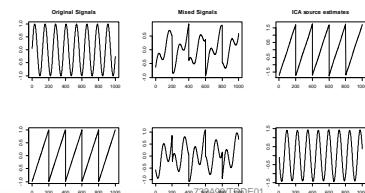
280

280

## Independent component analysis: R

R package: **fastICA**

```
S <- cbind(sin((1:1000)/20), rcp(((1:1000)-100)/100), 5))
A <- matrix(c(0.291, 0.657, -0.5439, 0.5572), 2, 2)
X <- S %*% A #mixing signals
a <- fastICA(X, 2) #now separate them
```



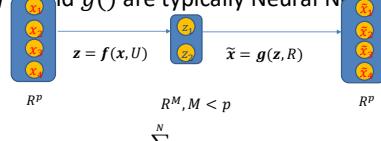
281

281

## Autoencoders (nonlinear PCA)

- Why linear transformations? Take nonlinear instead!

- $f(\cdot)$  and  $g(\cdot)$  are typically Neural Networks



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

...or some other loss function

732499/TDDE01

282

282

## 732A99/TDDE01 Machine Learning Lecture 3a Block 1: Kernel Methods

Jose M. Peña  
IDA, Linköping University, Sweden

1/38  
283

### Histogram Classification

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

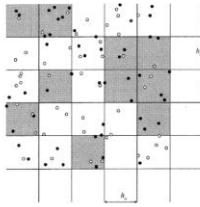


FIGURE 6.1. A cubic histogram rule:  
The decision is 1 in the shaded area.

- In other words,

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

4/38  
284

### Moving Window Classification

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

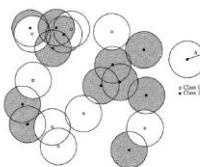


FIGURE 10.1. The moving window rule in  $\mathbb{R}^2$ . The decision is 1 in the shaded area.

- In other words,

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

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

5/38  
285

### Kernel Classification

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

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

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

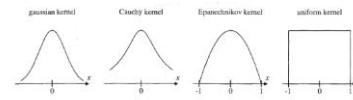


FIGURE 10.3. Various kernels on  $\mathbb{R}$ .

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

6/38

286

### Kernel Classification

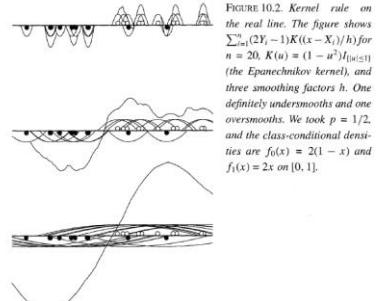


FIGURE 10.2. Kernel rule on the real line. The figure shows  $\sum_n (Y_i - 1) K((x - X_i)/h)$  for  $n = 20$ ,  $K(u) = (1 - u^2) I_{|u| \leq 1}$  (the Epanechnikov kernel), and three smoothing factors  $h$ . One definitely undersmooths and one oversmooths. We took  $p = 1/2$ , and the class-conditional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

7/38

287

### Histogram, Moving Window, and Kernel Regression

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

- In other words,

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

or

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

or

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

8/38  
288

288

## Histogram, Moving Window, and Kernel Density Estimation

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

$$P = \int_R p(x) dx \approx p(x) \text{Volume}(R)$$

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

$$|\{x_n \in R\}| \approx P N$$

and thus

$$p(x) \approx \frac{|\{x_n \in R\}|}{N \text{Volume}(R)}$$

- Then,

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

or

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

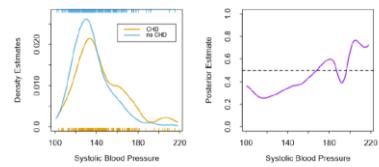
or

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

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

289

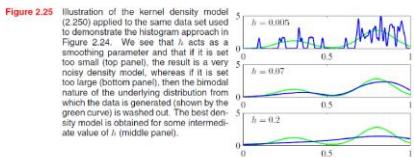
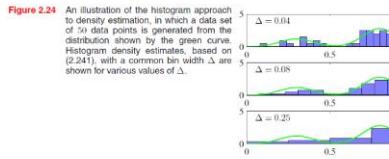
## Histogram, Moving Window, and Kernel Density Estimation



**FIGURE 6.14.** The left panel shows the two separate density estimates for systolic blood pressure in the CHD versus no-CHD groups, using a Gaussian kernel density estimate in each. The right panel shows the estimated posterior probabilities for CHD, using (6.35).

292

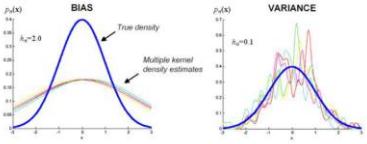
## Histogram, Moving Window, and Kernel Density Estimation



290

## Kernel Selection

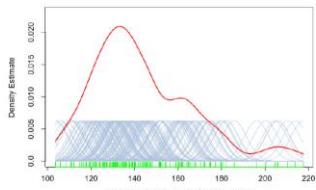
- How to choose the right kernel and width? E.g., by cross-validation.
- What does "right" mean? E.g., minimize loss function.
- Note that the width of the kernel corresponds to a bias-variance trade-off.



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

293

## Histogram, Moving Window, and Kernel Density Estimation



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

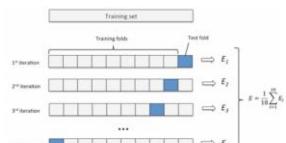
- From kernel density estimation to kernel classification:

- Estimate  $p(x|y=0)$  and  $p(x|y=1)$  using the methods just seen.
- Estimate  $p(y)$  as class proportions.
- Compute  $p(y|x) \propto p(x|y)p(y)$  by Bayes theorem.

291

## Kernel Selection

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

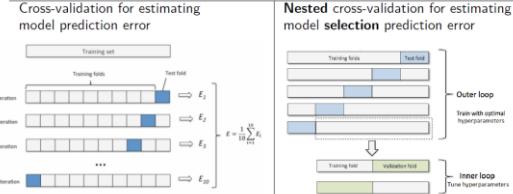


- If the training set contains  $N$  points, note that cross-validation estimates the prediction error when the model is trained on  $N - N/K$  points.
- Note that the model returned is trained on  $N$  points. So, cross-validation overestimates the prediction error of the model returned.
- This seems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

294

## Kernel Selection

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



- 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 ?

15/18

295

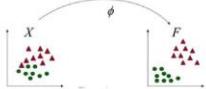
## Kernel Trick

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

- If the matrix

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

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



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

$$\phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}cx_1, \sqrt{2}cx_2, c)$$

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

16/18

296

## Kernel Trick

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

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

- Then,

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

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



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

17/18

297

## Kernel Trick

- Two alternatives for building  $k(\mathbf{x}, \mathbf{x}')$ :

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

### Techniques for Constructing New Kernels.

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

$$k(\mathbf{x}, \mathbf{x}') = ck_1(\mathbf{x}, \mathbf{x}') \quad (6.13)$$

$$k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}') \quad (6.14)$$

$$k(\mathbf{x}, \mathbf{x}') = q(k_1(\mathbf{x}, \mathbf{x}')) \quad (6.15)$$

$$k(\mathbf{x}, \mathbf{x}') = \exp(k_1(\mathbf{x}, \mathbf{x}')) \quad (6.16)$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \quad (6.17)$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}') \quad (6.18)$$

$$k(\mathbf{x}, \mathbf{x}') = k_3(\phi(\mathbf{x}), \phi(\mathbf{x}')) \quad (6.19)$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{A} \mathbf{x}' \quad (6.20)$$

$$k(\mathbf{x}, \mathbf{x}') = k_4(\mathbf{x}_a, \mathbf{x}'_a) + k_5(\mathbf{x}_b, \mathbf{x}'_b) \quad (6.21)$$

$$k(\mathbf{x}, \mathbf{x}') = k_6(\mathbf{x}_a, \mathbf{x}'_a)k_7(\mathbf{x}_b, \mathbf{x}'_b) \quad (6.22)$$

where  $c > 0$  is a constant,  $f(\cdot)$  is any function,  $q(\cdot)$  is a polynomial with nonnegative coefficients,  $\phi(\cdot)$  is a function from  $\mathbf{x}$  to  $\mathbb{R}^M$ ,  $k_1(\cdot, \cdot)$  is a valid kernel in  $\mathbb{R}^M$ ,  $\mathbf{A}$  is a symmetric positive semidefinite matrix,  $\mathbf{x}_a$  and  $\mathbf{x}'_a$  are variables (not necessarily disjoint) with  $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}'_a)$ , and  $k_a$  and  $k_b$  are valid kernel functions over their respective spaces.

18/18

298

## 732A99/TDDE01 Machine Learning Lecture 3b Block 1: Support Vector Machines

Jose M. Peña  
IDA, Linköping University, Sweden

1/18

299

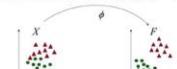
## Support Vector Machines for Classification

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

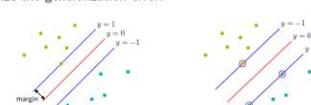
$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b$$

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

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



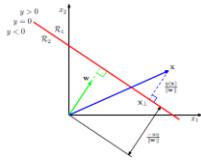
- Aim for the separating hyperplane that maximizes the margin (i.e. the smallest perpendicular distance from any point to the hyperplane) so as to minimize the generalization error.



4/18

300

### Support Vector Machines for Classification



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

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

- Then, the maximum margin separating hyperplane is given by

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

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

5/18

301

### Support Vector Machines for Classification

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

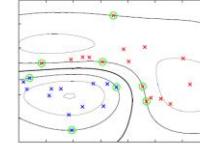
$$a_n(t_n y(\mathbf{x}_n) - 1) = 0$$

- Then,  $a_n > 0$  if and only if  $t_n y(\mathbf{x}_n) = 1$ . The points with  $a_n > 0$  are called support vectors and they lie on the margin boundaries.

- A new point  $\mathbf{x}$  is classified according to the sign of

$$\begin{aligned} y(\mathbf{x}) &= \mathbf{w}^T \phi(\mathbf{x}) + b = \sum_n a_n t_n \phi(\mathbf{x}_n)^T \phi(\mathbf{x}) + b = \sum_n a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b \\ &= \sum_{m \in S} a_m t_m k(\mathbf{x}, \mathbf{x}_m) + b \end{aligned}$$

where  $S$  are the indexes of the support vectors. Sparse solution!



8/18

304

### Support Vector Machines for Classification

- Then, the maximum margin separating hyperplane is given by

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

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

- To minimize the previous expression, we minimize

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

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

- 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  $\mathbf{w}$  and  $b$ , and maximizing with respect to  $a_n$ .

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

$$\begin{aligned} \mathbf{w} &= \sum_n a_n t_n \phi(\mathbf{x}_n) \\ 0 &= \sum_n a_n t_n \end{aligned}$$

6/18

302

### Support Vector Machines for Classification

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

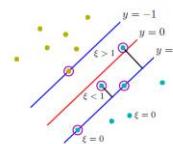
$$1 = t_n y(\mathbf{x}_n) = t_n \left( \sum_{m \in S} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) + b \right)$$

and multiplying both sides by  $t_n$ , we have that

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

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

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



9/18

305

### Support Vector Machines for Classification

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

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

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

- Again, this "easy" to maximize.

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

7/18

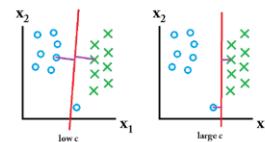
303

### Support Vector Machines for Classification

- The optimal separating hyperplane is given by

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

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



- To minimize the previous expression, we minimize

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

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

10/18

306

## Support Vector Machines for Classification

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

$$\begin{aligned}\mathbf{w} &= \sum_n a_n t_n \phi(\mathbf{x}_n) \\ 0 &= \sum_n a_n t_n \\ a_n &= C - \mu_n\end{aligned}$$

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

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

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

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

$$\begin{aligned}a_n(t_n y(\mathbf{x}_n) - 1 + \xi_n) &= 0 \\ \mu_n \xi_n &= 0\end{aligned}$$

- Then,  $a_n > 0$  if and only if  $t_n y(\mathbf{x}_n) = 1 - \xi_n$  for all  $n$ . The points with  $a_n > 0$  are called support vectors and they lie

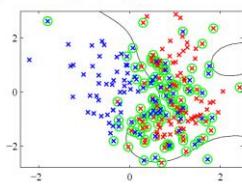
- on the margin if  $a_n < C$ , because then  $\mu_n > 0$  and thus  $\xi_n = 0$ , or
- inside the margin (even on the wrong side of the decision boundary) if  $a_n = C$ , because then  $\mu_n = 0$  and thus  $\xi_n$  is unconstrained.

11/18

307

## Support Vector Machines for Classification

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



- Not covered topics:

- Classifying into more than two classes.
- Returning class posterior probabilities.

12/18

308

## Support Vector Machines for Regression

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

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

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

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

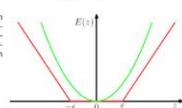
consider minimizing the  $\epsilon$ -insensitive regularized error function

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

where  $C > 0$  controls regularization and

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

**Figure 7.6** Plot of an  $\epsilon$ -insensitive error function (in red) in which the error increases linearly with distance beyond the insensitive region. The corresponding loss function is the quadratic error function (in green).



13/18

309

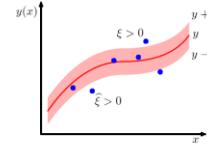
## Support Vector Machines for Regression

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

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

and

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



14/18

310

## Support Vector Machines for Regression

- The optimal regression curve is given by

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

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

- To minimize the previous expression, we minimize

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

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

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

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

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

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

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

15/18

311

## Support Vector Machines for Regression

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

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

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  $n$ :

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

$$\widehat{a}_n(t_n - y(\mathbf{x}_n) + \epsilon + \widehat{\xi}_n) = 0$$

$$\mu_n \xi_n = 0$$

$$\widehat{\mu}_n \widehat{\xi}_n = 0$$

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

16/18

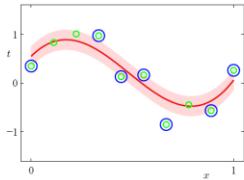
312

## Support Vector Machines for Regression

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

$$y(\mathbf{x}) = \sum_{m \in S} (\alpha_m - \hat{\alpha}_m) k(\mathbf{x}, \mathbf{x}_m) + b$$

where  $S$  are the indexes of the support vectors. Sparse solution!



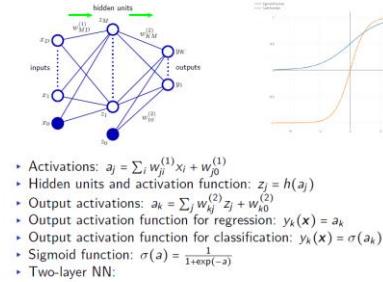
- To find  $b$ , consider any support vector  $\mathbf{x}_n$  with  $0 < \alpha_n < C$ . Then,  $\mu_n > 0$  and thus  $\xi_n = 0$  and thus  $0 = t_n - \epsilon - y(\mathbf{x}_n)$ . Then,

$$b = t_n - \epsilon - \sum_{m \in S} (\alpha_m - \hat{\alpha}_m) k(\mathbf{x}_n, \mathbf{x}_m)$$

17/18

313

## Neural Networks



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

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

5/18

316

## 732A99/TDDE01 Machine Learning Lecture 3c Block 1: Neural Networks

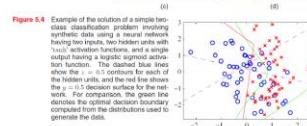
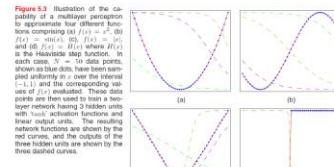
Jose M. Peña  
IDA, Linköping University, Sweden

1/16

314

## Neural Networks

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



6/16

317

## Neural Networks

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

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

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

$$y(\mathbf{x}) = \sum_{m \in S} (\alpha_m - \hat{\alpha}_m) k(\mathbf{x}, \mathbf{x}_m) + b$$

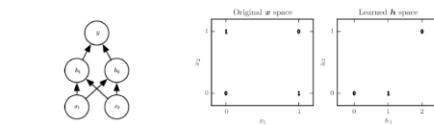
- SVMs imply **data-selected user-defined** basis functions.
- NNs imply a **user-defined** number of **data-selected** basis functions.

4/16

315

## Neural Networks

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



$$w_{11}^{(1)} = w_{12}^{(1)} = w_{21}^{(1)} = w_{22}^{(1)} = 1$$

$$w_{10}^{(1)} = 0, w_{20}^{(1)} = -1$$

$$h_j = z_j = h(a_j) = \max\{0, a_j\}$$

$$w_{11}^{(2)} = 1, w_{12}^{(2)} = -2$$

$$w_{20}^{(2)} = 0$$

$$y = y_k = a_k$$

7/16

318

### Backpropagation Algorithm

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

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

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

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

- Then, the likelihood function is

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

and thus

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

which is equivalent to the sum-of-squares error function for a given  $\sigma$ .

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

8/18

319

### Backpropagation Algorithm

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

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

- Since  $E_n$  depends on  $a_j$  only via  $a_k$ , then

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

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

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

- Putting all together, we have that

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

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

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

- Backpropagation algorithm:

- Forward propagate to compute activations, and hidden and output units.
- Compute  $\delta_k$  for the output units.
- Backpropagate the  $\delta$ 's, i.e. evaluate  $\delta_j$  for the hidden units recursively.
- Compute the required derivatives.

11/18

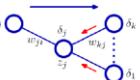
322

### Backpropagation Algorithm

- Backpropagation algorithm:

- Forward propagate to compute activations, and hidden and output units.
- Compute  $\delta_k$  for the output units.
- Backpropagate the  $\delta$ 's, i.e. evaluate  $\delta_j$  for the hidden units recursively.
- Compute the required derivatives.

**Figure 5.7** Illustration of the calculation of  $\delta_j$  for hidden unit  $j$  by backpropagation from the activation from units  $i$  to  $j$ . Blue unit  $j$  sends connections. The blue arrow denotes the direction of information flow during forward propagation, and the red arrows indicate the backward propagation of error information.



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

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

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

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

- This is an example of embarrassingly parallel algorithm.

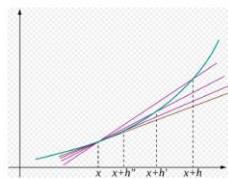
12/18

320

323

### Backpropagation Algorithm

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



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

9/18

### 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:

- Forward propagation, i.e. compute  
 $a_j = \sum_i w_{ji}x_i$  and  $z_j = h(a_j)$  and  $y_k = \sum_j w_{kj}z_j$
- Compute  
 $\delta_k = y_k - t_k$
- Backpropagate, i.e. compute  
 $\delta_j = (1 - z_j^2) \sum_k w_{kj}\delta_k$
- Compute  
 $\frac{\partial E_n}{\partial w_{ji}} = \delta_k z_j$  and  $\frac{\partial E_n}{\partial w_{ji}} = \delta_j x_i$



321

324

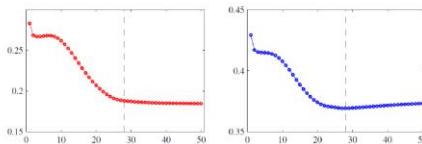
## Backpropagation Algorithm

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

325

14/18

## Regularization



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

- Regularization when learning the parameters: Early stopping the backpropagation algorithm according to the error on some validation data.
  - Regularization when learning the structure:
    - Cross-validation.
    - Penalizing complexity according to
 
$$E(\mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|^2 \text{ or } E(\mathbf{w}) + \frac{\lambda_1}{2} \|\mathbf{w}^{(1)}\|^2 + \frac{\lambda_2}{2} \|\mathbf{w}^{(2)}\|^2$$
- and choose  $\lambda$ , or  $\lambda_1$  and  $\lambda_2$  by cross-validation. Note that the effect of the penalty is simply to add  $\lambda w_{ji}$  and  $\lambda w_{kj}$ , or  $\lambda_1 w_{ji}$  and  $\lambda_2 w_{kj}$  to the appropriate derivatives.

15/18

326

## Limitations of Neural Networks

### Theorem (Universal approximation theorem)

For every continuous function  $f : [a, b]^D \rightarrow \mathbb{R}$  and for every  $\epsilon > 0$ , there 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  $p(\mathbf{x}, t)$ ,

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

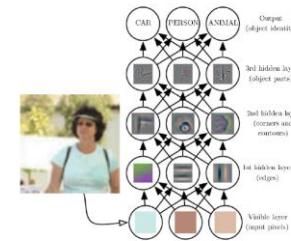
where  $L()$  is the 0/1 loss function.

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

4/18

328

## Deep Neural Networks

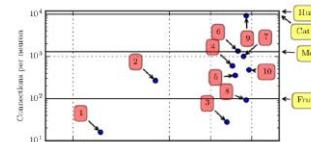


- A deep NN is a function that maps input to output.
- The mapping is formed by composing many simpler functions.
- Each layer provides a new representation of the input, i.e. complex concepts are built from simpler ones.
- The representation is learned automatically from data.

5/18

329

## Deep Neural Networks



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

- Adaptive linear element (Widrow and Hoff, 1960)
- Neocognitron (Fukushima, 1980)
- GPUs for convolutional networks (Ciresan et al., 2010)
- Deep Boltzmann machine (Hinton et al., 2006)
- Unsupervised convolutional network (Ciresan et al., 2010)
- GPU-accelerated multilayer perceptron (Ciresan et al., 2010)
- Distributed autoencoder (Joulin et al., 2012)
- Multi-GPU convolutional network (Gholami et al., 2012)
- COTS HPC unsupervised convolutional network (Ciresan et al., 2013)
- GoogLeNet (Szegedy et al., 2014a)

1/18

327

330

## Deep Neural Networks

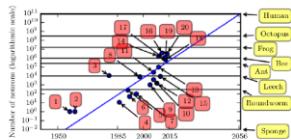
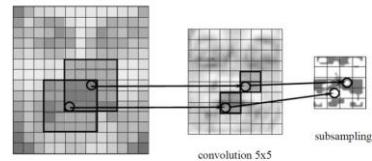


Figure 1.11: Since the introduction of hidden units, artificial neural networks have doubled in size roughly every 2.4 years. Biological neural network sizes from Wikipedia (2015).

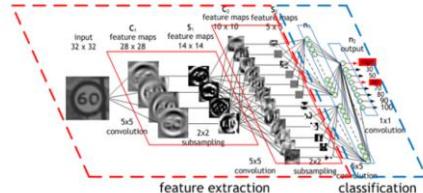
1. Perceptron (Rosenblatt, 1958, 1962)
  2. Adaptive linear elements (Widrow and Hoff, 1960)
  3. Nonnegative (Papert, 1969)
  4. Multilayer perceptron for handwritten digit recognition (Rumelhart et al., 1982)
  5. Recurrent neural network for speech recognition (Rabiner and Pollack, 1993)
  6. Multilayer perceptron for speech recognition (Huang et al., 1991)
  7. Mean field sigmoid belief network (Freud et al., 1990)
  8. Boltzmann machine (Ackley et al., 1985)
  9. Echo state network (Jaeger and Haas, 2004)
  10. Deep belief network (Hinton et al., 2006)
  11. Restricted Boltzmann machine (Hinton et al., 2006)
  12. Deep Boltzmann machine (Salakhutdinov and Hinton, 2009)
  13. GPU-accelerated deep belief network (Tomas et al., 2009)
  14. Unsupervised convolutional networks (Le et al., 2009)
  15. Convolutional neural network for image classification (Krizhevsky et al., 2012)
  16. AlexNet (Krizhevsky et al., 2012)
  17. Distributed autoencoders (Le et al., 2012)
  18. Deep learning (LeCun et al., 2015)
  19. COTS HPC unsupervised convolutional networks (Le et al., 2015)
  20. GoogLeNet (Szegedy et al., 2014)
- 22 layers DNN, but 12 times fewer weights than DNN 19

331

## Convolutional Networks



convolution  $5 \times 5$   
subsampling



11/18

334

## Deep Neural Networks

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

332

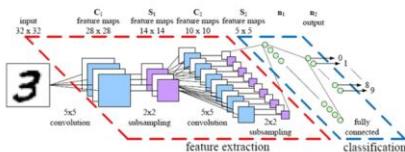
## Convolutional Networks

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

12/18

335

## Convolutional Networks



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

333

## Rectifier Activation Function

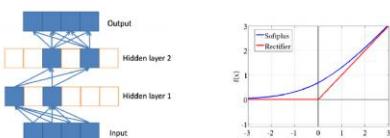


Figure 2: Left: Sparse propagation of activations and gradients in a network of rectifier units. The input selects a subset of active neurons and computation is linear in this subset. Right: Rectifier and softplus activation functions. The second one is a smooth version of the first.

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

13/18

336

### Rectifier Activation Function

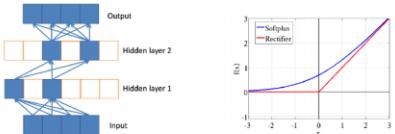


Figure 2: *Left:* Sparse propagation of activations and gradients in a network of rectifier units. The input selects a subset of active neurons and computation is linear in this subset. *Right:* Rectifier and softplus activation functions. The second one is a smooth version of the first.

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

14/58

337

### Layer-Wise Pre-Training

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

15/58

338

# 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                     |                                  |                             | Programming                  |                               |          |                                                                                                |              |                          |
|-----------------------------|----------------------------------|-----------------------------|------------------------------|-------------------------------|----------|------------------------------------------------------------------------------------------------|--------------|--------------------------|
| Creating Vectors            |                                  |                             | For Loop                     |                               |          | While Loop                                                                                     |              |                          |
| c(2, 4, 6)                  | 2 4 6                            | Join elements into a vector | for (variable in sequence){  | Do something                  | }        | while (condition){                                                                             | Do something | }                        |
| 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             | for (i in 1:4){              | j <- i + 10                   | print(j) | while (i < 5){                                                                                 | print(i)     | i <- i + 1               |
| rep(1:2, each=3)            | 1 1 1 2 2 2                      | Repeat elements of a vector |                              |                               |          |                                                                                                |              |                          |
| Vector Functions            |                                  |                             |                              |                               |          |                                                                                                |              |                          |
| <b>sort(x)</b>              | <b>rev(x)</b>                    |                             | If Statements                |                               |          | Functions                                                                                      |              |                          |
| Return x sorted.            | Return x reversed.               |                             | if (condition){              | Do something                  |          | function_name <- function(var){                                                                | Do something |                          |
| <b>table(x)</b>             | <b>unique(x)</b>                 | See counts of values.       | } else {                     | Do something different        | }        | return(new_variable)                                                                           |              |                          |
| Selecting Vector Elements   |                                  |                             |                              |                               |          |                                                                                                |              |                          |
| By Position                 |                                  |                             | Example                      |                               |          | Example                                                                                        |              |                          |
| <b>x[4]</b>                 | The fourth element.              |                             | if (i > 3){                  | print('Yes')                  |          | square <- function(x){                                                                         |              |                          |
| <b>x[-4]</b>                | All but the fourth.              |                             | } else {                     | print('No')                   | }        | squared <- x*x                                                                                 |              |                          |
| <b>x[2:4]</b>               | Elements two to four.            |                             |                              |                               |          | return(squared)                                                                                |              |                          |
| <b>x[-(2:4)]</b>            | All elements except two to four. |                             | Reading and Writing Data     |                               |          |                                                                                                |              |                          |
| <b>x[c(1, 5)]</b>           | Elements one and five.           |                             | Input                        |                               |          | Output                                                                                         |              |                          |
| By Value                    |                                  |                             |                              |                               |          |                                                                                                |              |                          |
| <b>x[x == 10]</b>           | Elements which are equal to 10.  |                             | df <- read.table('file.txt') | write.table(df, 'file.txt')   |          | Description                                                                                    |              |                          |
| <b>x[x &lt; 0]</b>          | All elements less than zero.     |                             |                              |                               |          | Read and write a delimited text file.                                                          |              |                          |
| <b>x[x %in% c(1, 2, 5)]</b> | Elements in the set 1, 2, 5.     |                             | 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. |              |                          |
| Named Vectors               |                                  |                             |                              |                               |          |                                                                                                |              |                          |
| <b>x['apple']</b>           | Element with name 'apple'.       |                             | 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 |
|                             |                                  |                             | a != b                       | Not equal                     | a < b    | Less than                                                                                      | a <= b       | Less than or equal to    |
|                             |                                  |                             |                              |                               |          |                                                                                                |              |                          |
|                             |                                  |                             |                              |                               |          |                                                                                                |              |                          |
|                             |                                  |                             |                              |                               |          |                                                                                                |              |                          |

## 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.

|  |                                          |                                                  |
|--|------------------------------------------|--------------------------------------------------|
|  | <code>m[2, ]</code> - Select a row       | <code>t(m)</code><br>Transpose                   |
|  | <code>m[, 1]</code> - Select a column    | <code>m %*% n</code><br>Matrix Multiplication    |
|  | <code>m[2, 3]</code> - Select an element | <code>solve(m, n)</code><br>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.

| <code>l[[2]]</code>  | <code>l[1]</code>                     | <code>l\$x</code> | <code>l['y']</code>                 |
|----------------------|---------------------------------------|-------------------|-------------------------------------|
| 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 |

### Matrix subsetting

|                       |  |
|-----------------------|--|
| <code>df[, 2]</code>  |  |
| <code>df[2, ]</code>  |  |
| <code>df[2, 2]</code> |  |

### List subsetting

|                    |  |                      |  |
|--------------------|--|----------------------|--|
| <code>df\$x</code> |  | <code>df[[2]]</code> |  |
|--------------------|--|----------------------|--|

Understanding a data frame

`View(df)` See the full data frame.

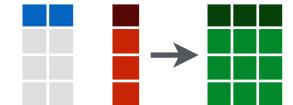
`head(df)` See the first 6 rows.

`nrow(df)`  
Number of rows.

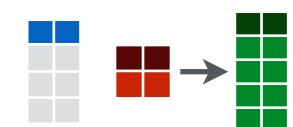
`ncol(df)`  
Number of columns.

`dim(df)`  
Number of columns and rows.

`cbind` - Bind columns.



`rbind` - Bind rows.



## Strings

|                                        |                                       |
|----------------------------------------|---------------------------------------|
| <code>paste(x, y, sep = ' ')</code>    | Join multiple vectors together.       |
| <code>paste(x, collapse = ' ')</code>  | Join elements of a vector together.   |
| <code>grep(pattern, x)</code>          | Find regular expression matches in x. |
| <code>gsub(pattern, replace, x)</code> | Replace matches in x with a string.   |
| <code>toupper(x)</code>                | Convert to uppercase.                 |
| <code>tolower(x)</code>                | Convert to lowercase.                 |
| <code>nchar(x)</code>                  | Number of characters in a string.     |

## Factors

|                                 |                                                                  |
|---------------------------------|------------------------------------------------------------------|
| <code>factor(x)</code>          |                                                                  |
| <code>cut(x, breaks = 4)</code> | Turn a numeric vector into a factor but 'cutting' into sections. |

## Statistics

|                                  |                                            |
|----------------------------------|--------------------------------------------|
| <code>lm(x ~ y, data=df)</code>  | Linear model.                              |
| <code>glm(x ~ y, data=df)</code> | Generalised linear model.                  |
| <code>summary</code>             | Get more detailed information out a model. |
| <code>prop.test</code>           | Test for a difference between proportions. |

## Distributions

|          | Random Variates     | Density Function    | Cumulative Distribution | Quantile            |
|----------|---------------------|---------------------|-------------------------|---------------------|
| Normal   | <code>rnorm</code>  | <code>dnorm</code>  | <code>pnorm</code>      | <code>qnorm</code>  |
| Poisson  | <code>rpois</code>  | <code>dpois</code>  | <code>ppois</code>      | <code>qpois</code>  |
| Binomial | <code>rbinom</code> | <code>dbinom</code> | <code>pbinom</code>     | <code>qbinom</code> |
| Uniform  | <code>runif</code>  | <code>dunif</code>  | <code>unif</code>       | <code>qunif</code>  |

## Plotting

|                         |                        |
|-------------------------|------------------------|
| <code>plot(x)</code>    | Values of x in order.  |
| <code>plot(x, y)</code> | Values of x against y. |
| <code>hist(x)</code>    | 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; all.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>