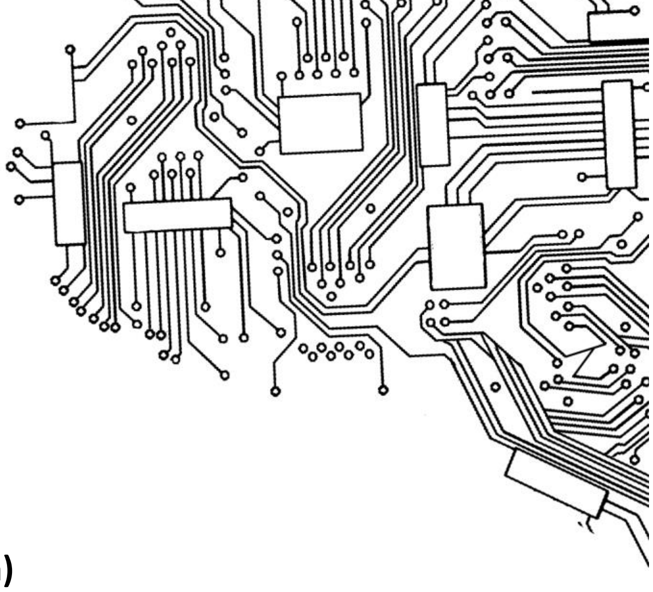
(Cross-) Validation

Simple validation

```
set.seed(23) # to make the example reproducible

?sample

library(ggplot2)

?diamonds

mydiamonds = diamonds[1:500,]

attach(mydiamonds)

trainingdiamonds = sample(x = 500, size = 200) # 200 observations from 500 in the df

mylm = lm(data = mydiamonds, subset = trainingdiamonds, x ~ y + z) # fitting a simple lm

mean ((x - predict(mylm, mydiamonds))[-trainingdiamonds]^2) # mean standard error

library(boot) # for the cv functions
```



LOOCV

```
myglm = glm(data = mydiamonds, x ~ y + z)
myglm.error = cv.glm(data = mydiamonds, myglm)
myglm.error$delta
# delta is the cv estimate or the error rate - raw and adjusted
```

Cross Validation K=5

```
myglm.error2 = cv.glm(data = mydiamonds, myglm, K = 5)
# using K to adjust the group number
myglm.error2$delta
```

Exercise Cross Validation

Create a model with the faithful dataset

?faithful

- **explain waiting with eruptions**
- **get a visual impression to get an idea of the relationship**
- **get the MSE using simple cross validation and 5-fold CV**
- **in the solution I will use a 50/50 split of the dataset for simple CV**
- **compare the results - which one has a lower error rate?**
- **what could be possible problems with the CV approach I outlined**

1. simple CV

```
# simple xy plot
plot(faithful$waiting, faithful$eruptions)

# linear model explaining the waiting time - training data 1-136
mymodel = glm(data = faithful[1:136,], waiting ~ eruptions)

# MSE on the second half of the data (validation set)
mean((faithful$waiting - predict(mymodel, faithful))[137:272]^2)

# Or an alternative way to code it
mean((faithful$waiting[137:272] - predict(mymodel, faithful[137:272,]))^2)
```

2. 5-fold CV

```
library(boot) # for the cv.glm function

# we are going to get a model with the ful dataset

mymodel2 = glm(data=faithful, waiting ~ eruptions)

# cv.glm for 5 fold CV

cv.result = cv.glm(data = faithful, mymodel2, K = 5)

# the error rate is lower than with standard CV

cv.result$delta

# if you perform simple CV with a 50/50 split, you could get bias in your model

# because the observations might be affected by time e.g. first oberservations
are higher than the last ones, etc

# so always be careful how you split your data, best way is to use "sample"
```

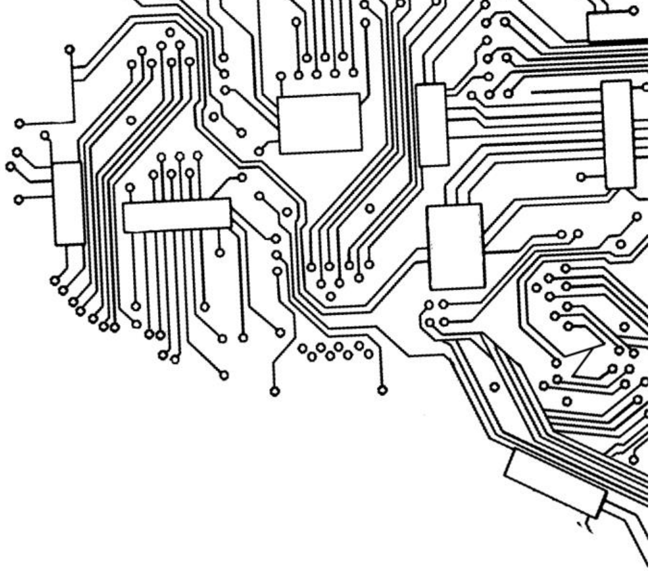
Classification

KNN Classification

```
?mtcars

attach(mtcars)

library(lattice)

# we want to create a model to classify the number of cylinders

# according to weight and mpg

with(mtcars, xyplot(wt ~ mpg, group=cyl, auto.key=T, pch=20, cex=3))

# for KNN we need to get the library class

library(class)

# lets take a look at the knn function we are using

?knn


# train: the data we use to create our model

# test: the data we use to test if the model works

# k: number of neighbors we use for the model
```

Extra arguments:

```
# l: a minimum amount of votes of one class

# use.all: tie handling

# prob: shows the proportion of winning class votes
```



```
# data frame for training

train <- cbind(mpg, wt)


# test data

test <- c(26,2.2)


knn(train, test, cl = cyl, k=2, prob = T)
```

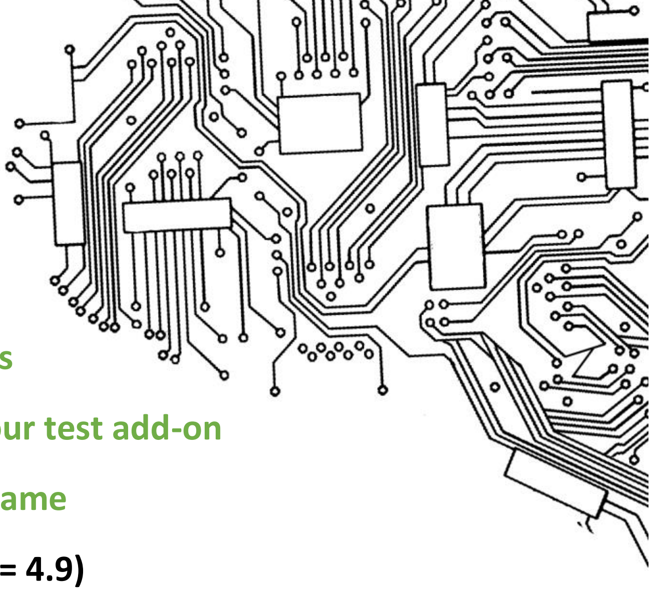
LDA Classification

```
# we need MASS for the lda function

library(MASS)

?lda

# similar to lm and glm

mylda = lda(data=mtcars, cyl ~ wt + mpg)

mylda

# prior probabilities give the proportions of a class in the dataset

# we see the group means for each independent variable and class

# coefficients are calculated to define the areas of each class
```

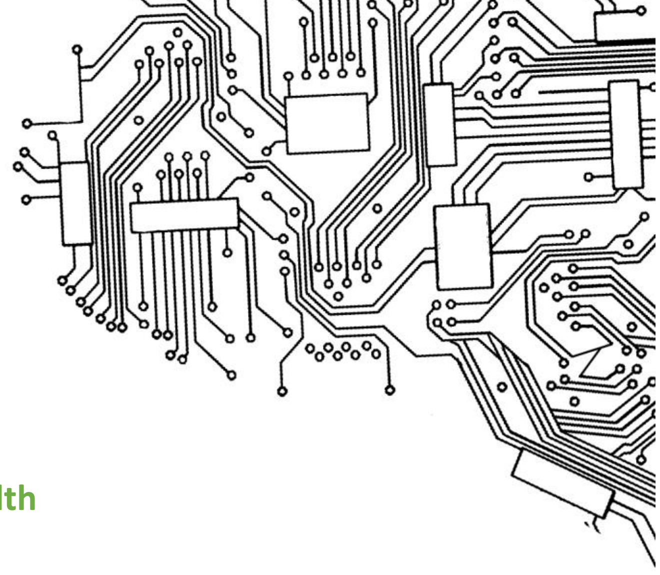
```r
plot(mylda)

# to use the predict function, it is useful to create a data frame with the test vectors

# test data as data frame

wt= c(2.2, 4, 1.1,5)

mpg= c(26, 20, 27, 15)

class=c(4, 6, 4, 8)

test = data.frame(wt, mpg,class)

mylda.prediction = predict(object = mylda, newdata = test[,c(1,2)])$class

# we specify that we want the class as output

mylda.prediction

table(mylda.prediction, test[,3])

# the diagonal shows the correct predictions

# as you can see in the table, there is one error in the result

# observation 2 in the test set was classified as 8

# in the data frame it is class 6

# on the plot we can see that it is quite near to class 8
```

Logistic Regression

```
# in this case we want to model a binary outcome am, with wt, mpg and drat

head(mtcars)

mymtcars=data.frame(am = as.factor(mtcars$am),

                wt = mtcars$wt,

                mpg = mtcars$mpg,

                drat = mtcars$drat)

# since we are performing a logistic regression on a classification, we check if our outcome

# variable is in deed a factor (class)

class(mymtcars$am)

# glm with family = binomial is the classic way of logistic regression in R

mylog = glm(data = mymtcars, am ~  wt + mpg + drat, family = "binomial")

summary(glm(data = mymtcars, am ~  wt + mpg + drat, family = "binomial"))

# in this case I decide to keep all three predictors in the model

# we are going to run the model on the training data itself

testprediction <- predict(mylog, type="response")

testprediction

# prob <= 0.5 means 0 or automatic

# we can get a character vector of the 2 transmission types

predicted.classes = rep( "automatic" ,32)

predicted.classes[testprediction > .5]="manual"

predicted.classes
```

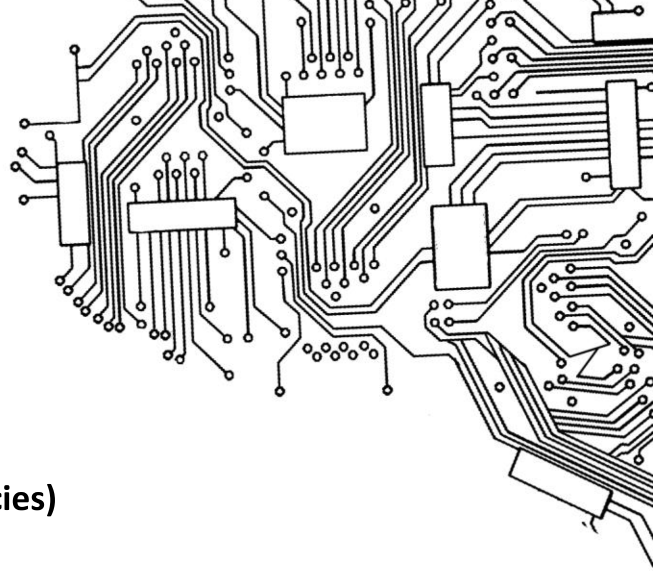


```
table(predicted.classes, mymtcars$am)

# the table tells us that we had 2 misclassifications

# now we see what the model would predict for our test add-on

# for the predict function it is best to use a data frame

addon = data.frame(wt = 4.500, mpg = 30.2 , drat = 4.9)


predict(mylog, addon, type="response")

# type response for probabilities


## the model would predict that a car of 4500 lb has 0 % probability of having a
manual transom
```



Exercise Iris - LDA and KNN

```
# Classify the iris dataset according to Species
# using the predictors Petal.Length and Petal.Width
# perform both LDA and KNN (k of 3)

# test dataframe
Petal.Width = c(0.7, 2.5)
Petal.Length = c(2.4, 7)
Species = c("setosa", "virginica")
test = data.frame(Petal.Width, Petal.Length, Species)

library(lattice)
with(iris, xyplot(Petal.Length ~ Petal.Width, group=Species,
        auto.key=T, pch=20, cex=3))
library(MASS)

mylda = lda(data=iris, Species ~ Petal.Length+Petal.Width)
mylda
plot(mylda)

# to use the predict function, it is useful to create a data frame with the test
vectors
```



```
# test dataframe

Petal.Width = c(0.7, 2.5)

Petal.Length = c(2.4, 7)

Species = c("setosa", "virginica")

test = data.frame(Petal.Width, Petal.Length, Species)


mylda.prediction = predict(object = mylda, newdata = test[,c(1,2)])$class

# we specify that we want the class as output

mylda.prediction

table(mylda.prediction, test[,3])
```

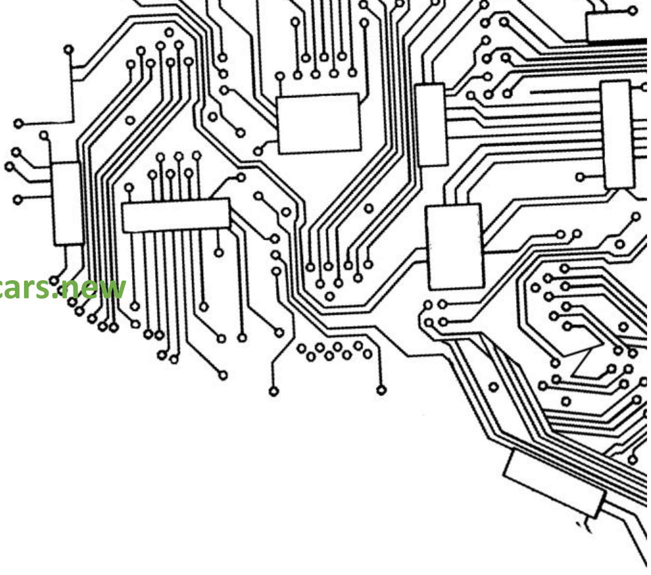
KNN

```
# clear the environment at first to avoid object name confusion

attach(iris)

train <- cbind(Petal.Width, Petal.Length)

library(class)

test = matrix(c(0.7, 2.5, 2.4, 7), nrow=2)

knn(train, test, cl=Species, k=3, prob=T)
```

Tree Based Models

```
# for our example we are using the tree library (one of the main packages for
trees)

library(tree)

# lets work with the standard mtcars dataset
attach(mtcars)
?tree
# lets plot the data to see what we are dealing with
library(lattice)
with(mtcars, xyplot(mpg ~ wt, group=am, auto.key=T, pch=20, cex=3))

# the function is quite similar to lm
mytree = tree(data=mtcars, am ~ wt + mpg)

# as you can see the terminal nodes give us group means
# which is not useful for a Yes vs No question like transmissions

plot(mytree)
text(mytree)
title("Pseudo-Regression Tree MTCARS",  sub="automatic vs manual
Transmission")
```

```
# for this true classification tree I am factorizing am

# you can see it as qualitative now

mytree = tree(data=mtcars, as.factor(am) ~ wt + mpg)


# a summary gives a first overview on the tree

summary(mytree)

plot(mytree)

text(mytree)


title("Classification Tree MTCARS",  sub="automatic vs manual Transmission")


# the terminal nodes appear to be the same, so the whole last split could have been omitted

# however it can be useful because as we can see in the previous tree, the means on the terminal nodes are very different


# lets split the dataset in half and calculate the test error rate


# I am creating a new data frame with am as factor

mtcars.new = data.frame(am.new = as.factor(am) ,wt, mpg)

head(mtcars.new)

class(mtcars.new$am.new)
```



```
# training and test data made out of the inital mtcars.new

train = mtcars.new[1:16,]

test = mtcars.new[17:32,]


# tree made from the training set

mytree.train = tree(am.new ~ wt + mpg, data=train)


# now we run a prediction on the test data

mytree.pred = predict(mytree.train, test, type="class")

# type class for classification


# and here we compare the results predicted vs reality

table(mytree.pred, test$am.new)

treetable = table(mytree.pred, test$am.new)

(sum(diag(treetable)))/16


# in this case the tree would be correct in 87.5 % of the cases
```

Exercise Classification Tree

Example with diamonds data

- **Library: ggplot2, dataset: diamonds**
- **create a tree to classify for color with price and the variable x**
- **use the first 500 rows for your tree**
- **plot and visualize the tree**
- **check the test error by splitting the set of 500 in 2 subgroups**

```
library(tree)

library(ggplot2)

attach(diamonds)

mytree = tree(data=diamonds[1:500,], color ~ price + x)

summary(mytree)

plot(mytree)

text(mytree)

title("Classification Tree Color of Diamonds")


# Lets check the test error rate

diamonds.df = data.frame(color, price, x)

diamonds.new = diamonds.df[1:500,]

head(diamonds.new); class(diamonds.new)

train = diamonds.new[1:250,]

test = diamonds.new[251:500,]
```

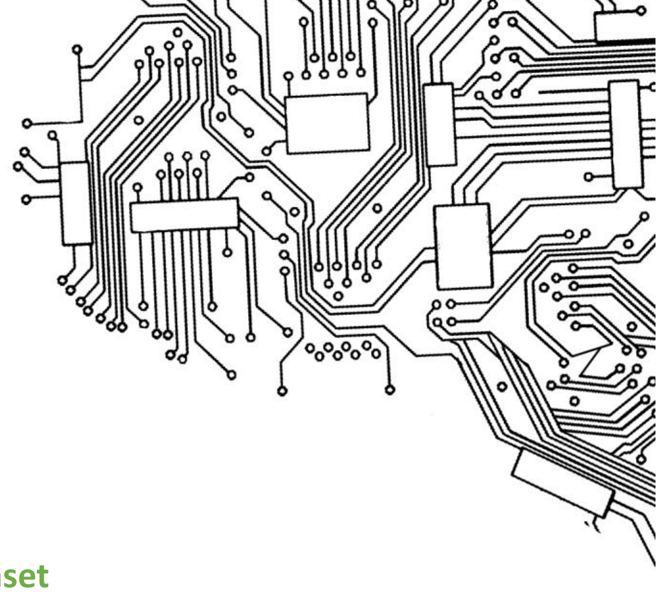
```
mytree.train = tree(color ~ price + x, data=train)

mytree.pred = predict(mytree.train, test, type="class")

# type class for classification

table(mytree.pred, test$color)

(sum(diag(table(mytree.pred, test$color))))/250
```

Random Forests and Bagging

- 2 methods to reduce variance in the model
- idea bagging: use bootstrapping to sample 100s of training sets and
- calculate a tree model for each of the sets - average the models
- idea randomFo: same as bagging but only a limited number of predictors is
- used to calculate a given split

```
library(randomForest)

library(ggplot2)

set.seed(123)

# mtr determins the number of predictor variables to be used

# the full number makes for a bagging approach

bagging = randomForest(formula = color ~ . , data = diamonds[1:500,], mtr=9)

plot(bagging)
```

```
# now it is a random forest with 3 predictors per split

randomFor = randomForest(formula = color ~ . , data = diamonds[1:500,],
mtr=3)

# we can check the importance of the predictors

importance(randomFor)

# and we can visualize it

varImpPlot(randomFor)

# lets test the random forest with a test data frame

test = diamonds[501:800,]

predicted.bagging = predict(newdata=test, bagging, type = "class")

predicted.randomFor = predict(newdata=test, randomFor, type = "class")

table(predicted.bagging, test$color)

table(predicted.randomFor, test$color)


# we can calculate the percentage of correct predictions

sum(diag(table(predicted.bagging, test$color)))/300

sum(diag(table(predicted.randomFor, test$color)))/300


# keep in mind that color has 7 levels - 0.35 better than pure chance of 1/7
```



Clustering

K means clustering

```
# for a simple example we can use the rivers dataset
plot(rivers)
# the function is called kmeans, number of clusters = center
# nstart specifies the number of random sets to start with
kclust = kmeans(rivers, centers = 3, nstart = 30)
kclust
# we can get a visual impression of our clustered data
plot(rivers, col = kclust$cluster)
```

Hierarchical Clustering

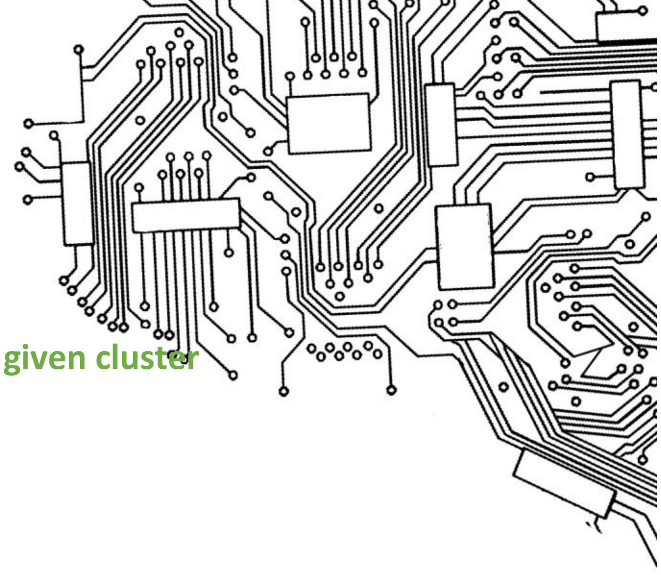
```
# simple example of Euclidean distance
# Square Root of Sum of Squares of Differences in Attributes
a = mtcars[1,]
b = mtcars[11,]
dist(rbind(a,b))

# lets get the distance matrix of the first 16 obs at first
dm = dist(as.matrix(mtcars[1:16,]))
```

```
# use the hclust function for hierachical clustering

hcluster = hclust(dm) # standard method is complete linkage

# dendrogram

plot(hcluster)
```

Exercise K Means Clustering

- data for the exercise: extract the first 3 numeric columns from the iris dataset
- head(iris)
- check if the extraction worked (hint: head, summary, nrow, class, ...)
- perform K means clustering on the dataset (experiment with the number of K)
- create a vector with all observation numbers of a specific cluster (hint: which, on cluster 3)
- visualize your results in a 3d plot (hint: library rgl, plot3d)

```
# at first lets get the dataset to work with

clusterdata = data.frame(iris$Sepal.Length, iris$Sepal.Width, iris$Petal.Length)

# checking if the data frame has the attributes we want

head(clusterdata); summary(clusterdata); nrow(clusterdata); class(clusterdata)

# I am performing 3 kmeans analyses with K of 3, 5, 8

clusterk3 = kmeans(clusterdata, centers = 3, nstart = 35)

clusterk5 = kmeans(clusterdata, centers = 5, nstart = 35)

clusterk8 = kmeans(clusterdata, centers = 8, nstart = 35)
```



```
# Lets take a look at the 3 objects

clusterk3; clusterk5; clusterk8

# extracting a vector with all observation IDs in a given cluster

cluster3vector = which(clusterk3$cluster == 3)

cluster3vector


library(rgl) # for an easy 3d scatterplot function


# using the plot3d function to get a 3 dimensional scatterplot

plot3d(clusterdata, size = 6, col = clusterk3$cluster,

    xlab = "", ylab = "", zlab = "", sub = "3 Clusters")


plot3d(clusterdata, size = 6, col = clusterk5$cluster,

    xlab = "", ylab = "", zlab = "", sub = "5 Clusters")


plot3d(clusterdata, size = 6, col = clusterk8$cluster,

    xlab = "", ylab = "", zlab = "", sub = "8 Clusters")



# which plots shows be most overlapping?
```