# H2 Single Layer MLP

*Bankbintje*

*22 februari 2016*

## Data

### Load Data & Packages

Required packages worden geladen. Aanname is dat deze al voorkomen.

```
library(neuralnet)
```

```
## Loading required package: grid
## Loading required package: MASS
```

```
library(nnet)
```

```
## Warning: package 'nnet' was built under R version 3.1.3
```

```
library(plyr)
```

Install package "MixAll" en laad de dataset "DebTrivedi". Check # rijen en kolommen. Beschrijving kolommen:

- ofp (number of physician office visits)
- hosp (number of hospital stays)
- health (self-perceived health status)
- numchron (number of chronic conditions)
- gender
- school (number of years of education)
- privins (private insurance indicator)

```
##install.packages("MixAll")
data("DebTrivedi", package="MixAll")
data<-DebTrivedi
nrow(data)
```

```
## [1] 4406
```

```
ncol(data)
```

```
## [1] 19
```

Verwijder data met NA's, check # rijen en kolommen.

```r
data<-na.omit(data)
nrow(data)
```

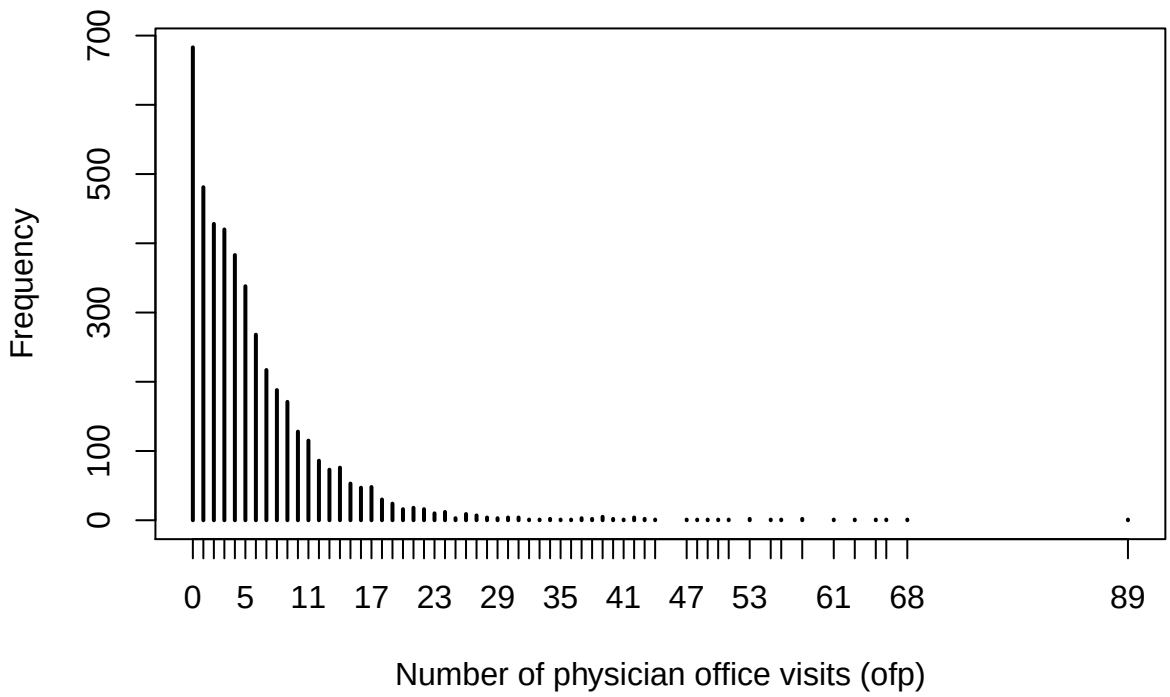
```
## [1] 4406
```

```r
ncol(data)
```

```
## [1] 19
```

### Explore Data

```
plot(table(data$ofp), xlab= "Number of physician office visits (ofp)", ylab="Frequency")
```

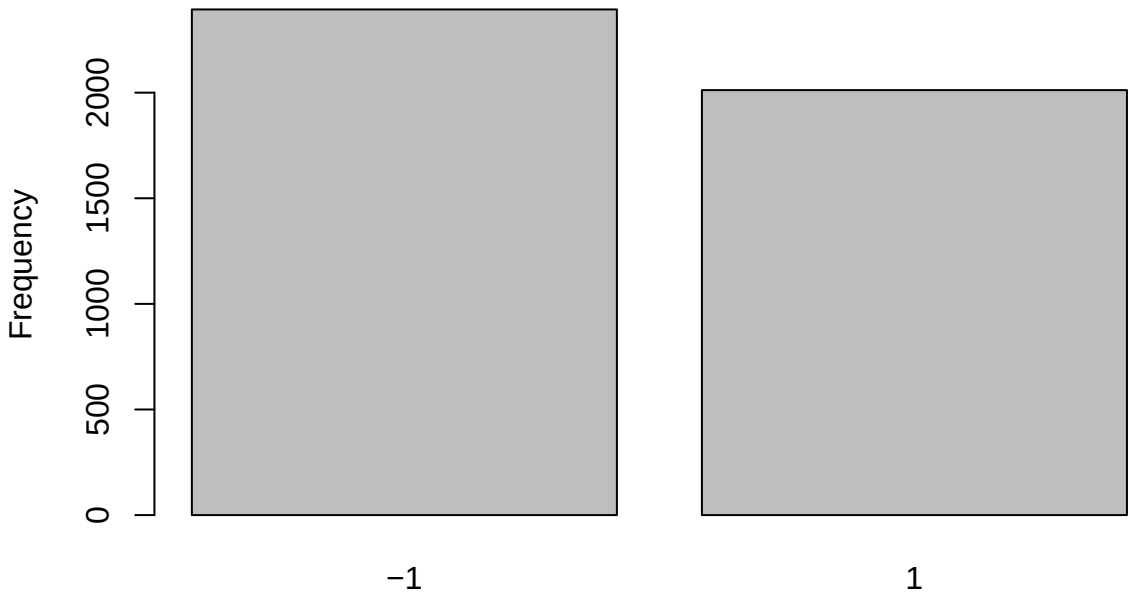



Doel is om NN te gebruiken om te voorspellen of een individu hoger of lager dan gemiddeld aantal "office visits" heeft. Daarom een variabele toevoegen die waarde -1 heeft als het aantal ofp's lager is dan de mediaan en 1 als deze hoger is dan de mediaan.

```
data$Class<- ifelse(DebTrivedi$ofp>=median(DebTrivedi$ofp),-1,1)
```

Check de verdeling

```
barplot(table(data$Class), ylab = "Frequency")
```



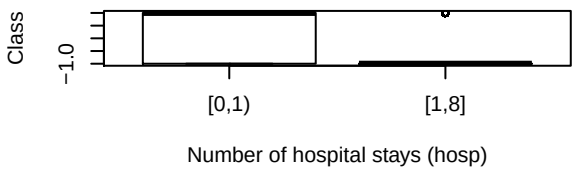

Gebruik cfac function om relatie te tonen tussen “Class” en de covariates 1. hosp 2. health 3. numchron 4. gender 5. school 6. privins

uit https://cran.r-project.org/web/packages/pscl/vignettes/countreg.pdf

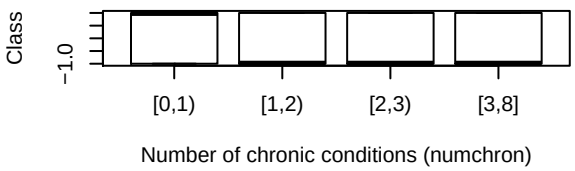
```
cfac <- function (x, breaks = NULL){
if(is.null(breaks)) breaks <- unique(quantile(x, 0:10/10))
x <- cut(x,breaks,include.lowest=TRUE, right=FALSE)
levels(x) <- paste(breaks[-length(breaks)], ifelse(diff(breaks)>1,
c(paste("-", breaks[-c(1, length(breaks))] -1, sep = ""), "+"), ""),
sep = "") + return(x)}
```

Plot de data

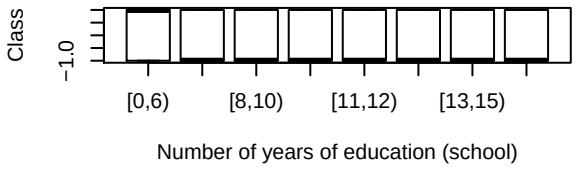
```
par (mfrow = c(3,2))
plot (Class ~ cfac(hosp), data = data, xlab = "Number of hospital stays (hosp)")
plot (Class ~ cfac(numchron), data = data, xlab = "Number of chronic conditions (numchron)")
plot (Class ~ cfac(school), data = data, xlab = "Number of years of education (school)")
plot (Class ~ health, data = data)
plot (Class ~ gender, data = data)
plot (Class ~ privins, data = data, xlab = "Private Insurance (privins)")
```



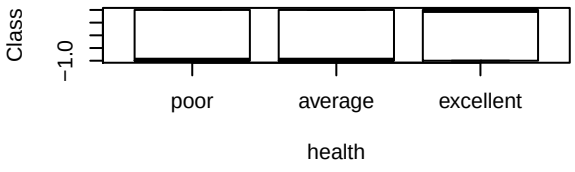

Number of hospital stays (hosp)



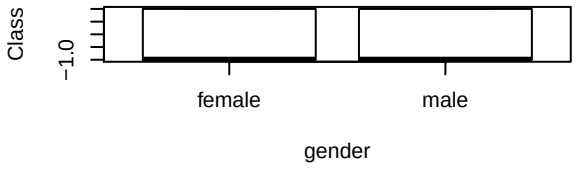

Number of chronic conditions (numchron)



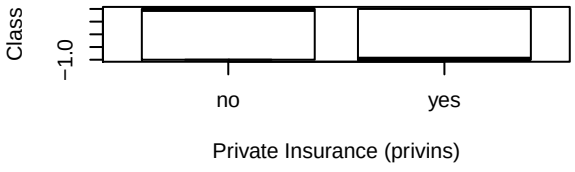

Number of years of education (school)




health




gender




Private Insurance (privins)

## Prepare Data

Converteer gender, privins en health naar numerieke variabelen.

```
levels(data$gender)<- c("-1","1")
data$gender<- as.numeric(as.character(data$gender))

levels(data$privins)<- c("-1","1")
data$privins<- as.numeric(as.character(data$privins))

levels(data$health)<- c("0","1","2")
data$health<- as.numeric(as.character(data$health))
```

Converteer overige variabelen naar numeriek

```
data$hosp <- as.numeric(as.character(data$hosp))
data$numchron <- as.numeric(as.character(data$numchron))
data$school <- as.numeric(as.character(data$school))
```

Selecteer alleen relevante variabelen

```
keeps<-c("Class", "hosp", "health", "numchron", "gender", "school", "privins")
data<-data[keeps]
head(data)
```

```
##   Class hosp health numchron gender school privins
## 1    -1    1      1        2      1      6       1
## 2     1    0      1        2     -1     10       1
## 3    -1    3      0        4     -1     10      -1
## 4    -1    1      0        2      1      3       1
## 5     1    0      1        2     -1      6       1
## 6    -1    0      0        5     -1      7      -1
```

Centreer data: subtract the mean of all data points from each individual data point.

```
head(scale(data, center = TRUE, scale = FALSE))
```

```
##     Class       hosp      health  numchron     gender    school    privins
## 1 -0.9133  0.7040399  0.04788924 0.4580118  1.1929187 -4.290286  0.4471176
## 2  1.0867 -0.2959601  0.04788924 0.4580118 -0.8070813 -0.290286  0.4471176
## 3 -0.9133  2.7040399 -0.95211076 2.4580118 -0.8070813 -0.290286 -1.5528824
## 4 -0.9133  0.7040399 -0.95211076 0.4580118  1.1929187 -7.290286  0.4471176
## 5  1.0867 -0.2959601  0.04788924 0.4580118 -0.8070813 -4.290286  0.4471176
## 6 -0.9133 -0.2959601 -0.95211076 3.4580118 -0.8070813 -3.290286 -1.5528824
```

Standardiseer (center & scale) de data met scale: subtract the mean of all data points from each individual data point, then divide those points by the standard deviation of all points.

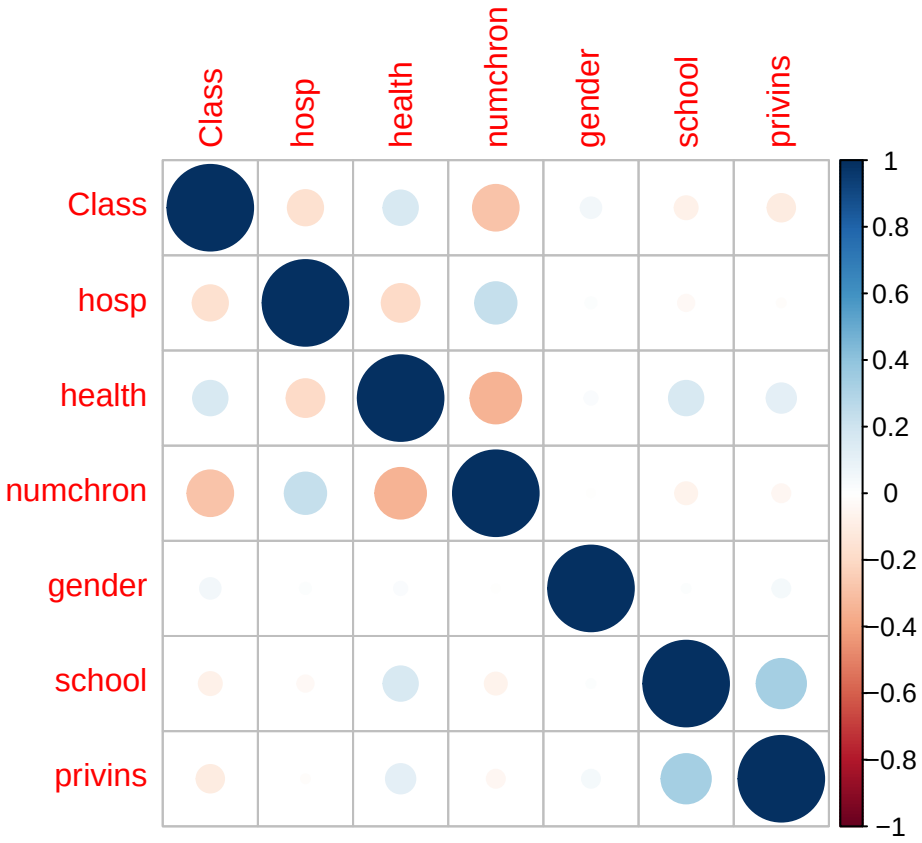
```
data<-scale(data)
head(data)
```

```
##        Class       hosp     health  numchron     gender      school
## 1 -0.9166481  0.9432503  0.1067271 0.3393606  1.2156191 -1.14752323
## 2  1.0906836 -0.3965179  0.1067271 0.3393606 -0.8224394 -0.07764282
## 3 -0.9166481  3.6227866 -2.1218967 1.8212464 -0.8224394 -0.07764282
## 4 -0.9166481  0.9432503 -2.1218967 0.3393606  1.2156191 -1.94993354
## 5  1.0906836 -0.3965179  0.1067271 0.3393606 -0.8224394 -1.14752323
## 6 -0.9166481 -0.3965179 -2.1218967 2.5621893 -0.8224394 -0.88005313
##      privins
## 1  0.5365279
## 2  0.5365279
## 3 -1.8634130
## 4  0.5365279
## 5  0.5365279
## 6 -1.8634130
```

### Correlation Plots

Maak een standaard corrplot van de dataset

```
library(corrplot)
corrplot(cor(data))
```



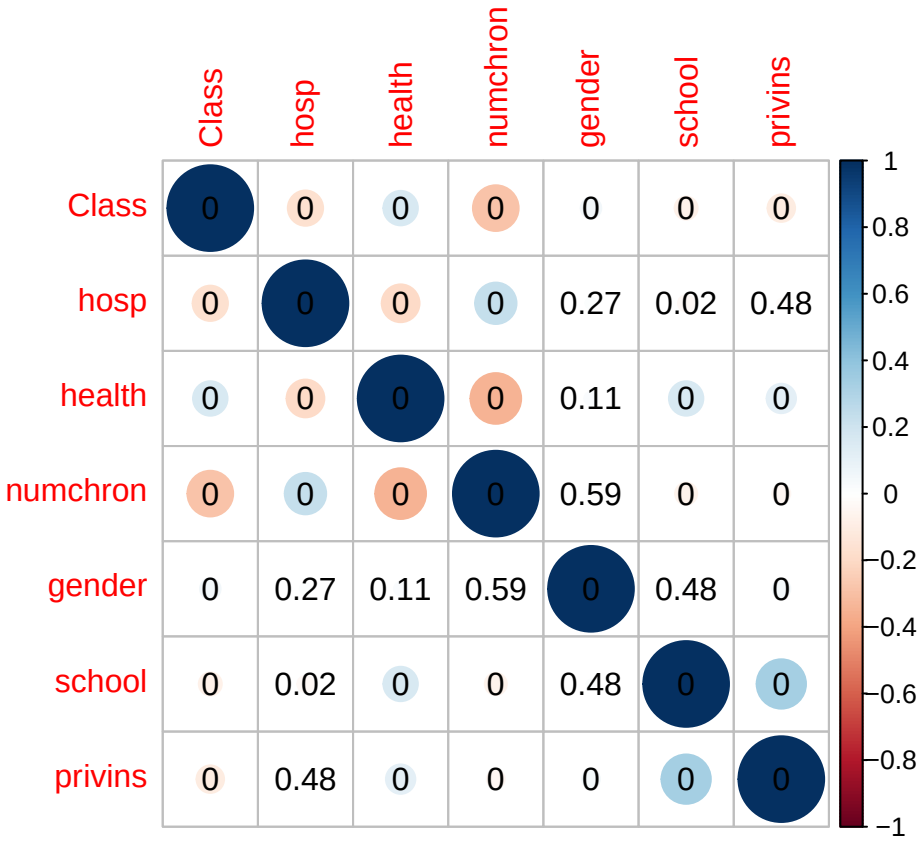

Functie voor het bepalen van correlatie & significantie

```
cor.mtest <- function(mat, conf.level = 0.95) {
    mat <- as.matrix(mat)
    n <- ncol(mat)
    p.mat <- lowCI.mat <- uppCI.mat <- matrix(NA, n, n)
    diag(p.mat) <- 0
    diag(lowCI.mat) <- diag(uppCI.mat) <- 1
    for (i in 1:(n - 1)) {
        for (j in (i + 1):n) {
            tmp <- cor.test(mat[, i], mat[, j], conf.level = conf.level)
            p.mat[i, j] <- p.mat[j, i] <- tmp$p.value
            lowCI.mat[i, j] <- lowCI.mat[j, i] <- tmp$conf.int[1]
            uppCI.mat[i, j] <- uppCI.mat[j, i] <- tmp$conf.int[2]
        }
    }
    return(list(p.mat, lowCI.mat, uppCI.mat))
}
```

Verschillende manieren om correlatie te plotten. Zie: https://cran.r-project.org/web/packages/corrplot/vignettes/corrplot-intro.html

```
res<-cor.mtest(data)
## add p-values on no significant coefficient
corrplot(cor(data), p.mat = res[[1]], sig.level = 0.1,insig="p-value")
```


```
## add all p-values
corrplot(cor(data), p.mat = res[[1]], sig.level = -1,insig="p-value")
```


Class
hosp
health
numchron
gender
school
privins
Class 0 0 0 0 0 0 0
hosp 0 0 0 0 0.27 0.02 0.48
health 0 0 0 0 0.11 0 0
numchron 0 0 0 0 0.59 0 0
gender 0 0.27 0.11 0.59 0 0.48 0
school 0 0.02 0 0 0.48 0 0
privins 0 0.48 0 0 0 0 0
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1

# Neural Network

## Preparations

Schrijf de formule eerst als variabele:

```
f<-Class ~ hosp + health + numchron + gender + school + privins
# seed
set.seed(103)
n=nrow(data)
# selecteer trainingset
train <- sample(1:n, 4000, FALSE)
```

## Estimate the model

Maak een MLP met een hidden layer

```
fit <- neuralnet(f, data = data[train,], hidden = 1, algorithm = "rprop+",
                 err.fct = "sse", act.fct = "logistic",
                 linear.output = FALSE)
```

- hidden = nbr of hidden layers
- algorithm = “rprop+” = resilient backpropagation with backtracking
- algorithm = “backprop” = traditional backpropagation, needs learning rate
- learningrate = 0.01
- err.fct = error function: sum squared errors (“sse”“) or cross-entropy (”ce“)
- act.fct = type of activation function
- linear.output = if set to “TRUE” the node’s output is not transformed by the specified activation function ; it is in essence linear

## Results

```
print(fit)
```

```
## Call: neuralnet(formula = f, data = data[train, ], hidden = 1, algorithm = "rprop+",     err.fct = "s
##
## 1 repetition was calculated.
##
##         Error Reached Threshold Steps
## 1 1854.659153    0.008971185555 15376
```

The model converged after 15.376 steps with an error of 1874.

https://groups.google.com/forum/#!topic/rropen/qS7Fki9pj8k

```
plotnn <- function (x, rep = NULL, x.entry = NULL, x.out = NULL, radius = 0.15,
  arrow.length = 0.2, intercept = TRUE, intercept.factor = 0.4,
  information = TRUE, information.pos = 0.1, col.entry.synapse = "black",
  col.entry = "black", col.hidden = "black", col.hidden.synapse = "black",
  col.out = "black", col.out.synapse = "black", col.intercept = "blue",
  fontsize = 12, dimension = 6, show.weights = TRUE, file = NULL,
  ...)
{
  net <- x
  if (is.null(net$weights))
    stop("weights were not calculated")
  if (!is.null(file) && !is.character(file))
    stop("'file' must be a string")
  if (is.null(rep)) {
    for (i in 1:length(net$weights)) {
      if (!is.null(file))
        file.rep <- paste(file, ".", i, sep = "")
      else file.rep <- NULL
      # dev.new()
      plot.nn(net, rep = i, x.entry, x.out, radius, arrow.length,
        intercept, intercept.factor, information, information.pos,
        col.entry.synapse, col.entry, col.hidden, col.hidden.synapse,
        col.out, col.out.synapse, col.intercept, fontsize,
        dimension, show.weights, file.rep, ...)
    }
  }
  else {
    if (is.character(file) && file.exists(file))
      stop(sprintf("%s already exists", sQuote(file)))
    result.matrix <- t(net$result.matrix)
    if (rep == "best")
      rep <- as.integer(which.min(result.matrix[, "error"]))
    if (rep > length(net$weights))
      stop("'rep' does not exist")
    weights <- net$weights[[rep]]
    if (is.null(x.entry))
      x.entry <- 0.5 - (arrow.length/2) * length(weights)
```

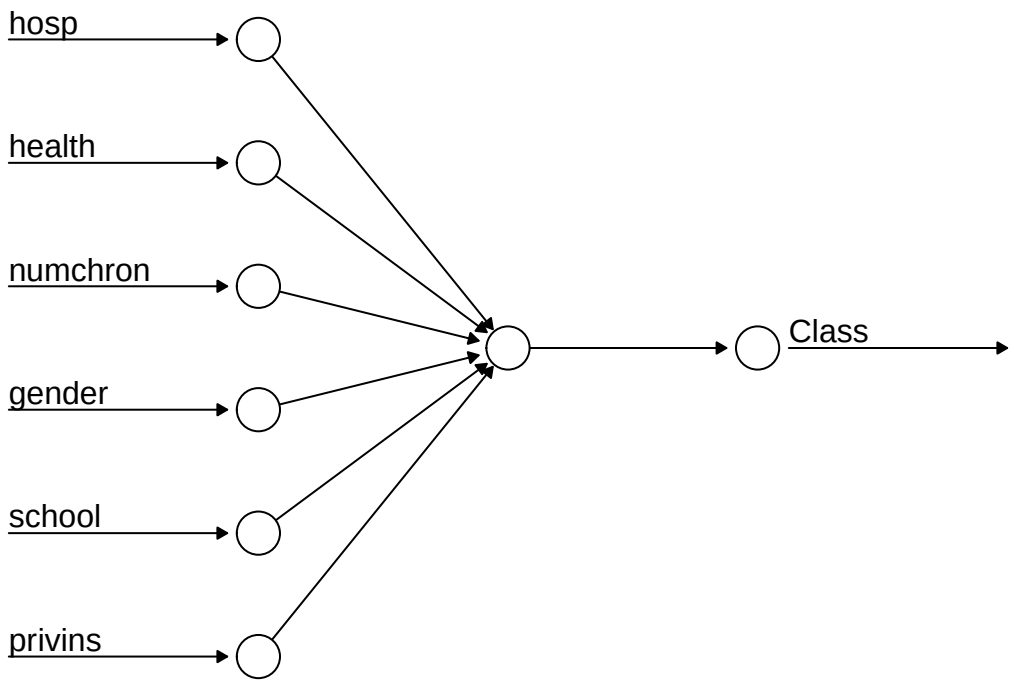
```
  if (is.null(x.out))
    x.out <- 0.5 + (arrow.length/2) * length(weights)
  width <- max(x.out - x.entry + 0.2, 0.8) * 8
  radius <- radius/dimension
  entry.label <- net$model.list$variables
  out.label <- net$model.list$response
  neuron.count <- array(0, length(weights) + 1)
  neuron.count[1] <- nrow(weights[[1]]) - 1
  neuron.count[2] <- ncol(weights[[1]])
  x.position <- array(0, length(weights) + 1)
  x.position[1] <- x.entry
  x.position[length(weights) + 1] <- x.out
  if (length(weights) > 1)
    for (i in 2:length(weights)) {
      neuron.count[i + 1] <- ncol(weights[[i]])
      x.position[i] <- x.entry + (i - 1) * (x.out -
        x.entry)/length(weights)
    }
  y.step <- 1/(neuron.count + 1)
  y.position <- array(0, length(weights) + 1)
  y.intercept <- 1 - 2 * radius
  information.pos <- min(min(y.step) - 0.1, 0.2)
  if (length(entry.label) != neuron.count[1]) {
    if (length(entry.label) < neuron.count[1]) {
      tmp <- NULL
      for (i in 1:(neuron.count[1] - length(entry.label))) {
        tmp <- c(tmp, "no name")
      }
      entry.label <- c(entry.label, tmp)
    }
  }
  if (length(out.label) != neuron.count[length(neuron.count)]) {
    if (length(out.label) < neuron.count[length(neuron.count)]) {
      tmp <- NULL
      for (i in 1:(neuron.count[length(neuron.count)] -
        length(out.label))) {
        tmp <- c(tmp, "no name")
      }
      out.label <- c(out.label, tmp)
    }
  }
  grid.newpage()
  for (k in 1:length(weights)) {
    for (i in 1:neuron.count[k]) {
      y.position[k] <- y.position[k] + y.step[k]
      y.tmp <- 0
      for (j in 1:neuron.count[k + 1]) {
        y.tmp <- y.tmp + y.step[k + 1]
        result <- calculate.delta(c(x.position[k],
          x.position[k + 1]), c(y.position[k], y.tmp),
          radius)
        x <- c(x.position[k], x.position[k + 1] -
          result[1])
```

```
        y <- c(y.position[k], y.tmp + result[2])
        grid.lines(x = x, y = y, arrow = arrow(length = unit(0.15,
          "cm"), type = "closed"), gp = gpar(fill = col.hidden.synapse,
          col = col.hidden.synapse, ...))
        if (show.weights)
          draw.text(label = weights[[k]][neuron.count[k] -
            i + 2, neuron.count[k + 1] - j + 1], x = c(x.position[k],
            x.position[k + 1]), y = c(y.position[k],
            y.tmp), xy.null = 1.25 * result, color = col.hidden.synapse,
            fontsize = fontsize - 2, ...)
      }
      if (k == 1) {
        grid.lines(x = c((x.position[1] - arrow.length),
          x.position[1] - radius), y = y.position[k],
          arrow = arrow(length = unit(0.15, "cm"),
            type = "closed"), gp = gpar(fill = col.entry.synapse,
            col = col.entry.synapse, ...))
        draw.text(label = entry.label[(neuron.count[1] +
          1) - i], x = c((x.position - arrow.length),
          x.position[1] - radius), y = c(y.position[k],
          y.position[k]), xy.null = c(0, 0), color = col.entry.synapse,
          fontsize = fontsize, ...)
        grid.circle(x = x.position[k], y = y.position[k],
          r = radius, gp = gpar(fill = "white", col = col.entry,
            ...))
      }
      else {
        grid.circle(x = x.position[k], y = y.position[k],
          r = radius, gp = gpar(fill = "white", col = col.hidden,
            ...))
      }
    }
  }
  out <- length(neuron.count)
  for (i in 1:neuron.count[out]) {
    y.position[out] <- y.position[out] + y.step[out]
    grid.lines(x = c(x.position[out] + radius, x.position[out] +
      arrow.length), y = y.position[out], arrow = arrow(length = unit(0.15,
      "cm"), type = "closed"), gp = gpar(fill = col.out.synapse,
      col = col.out.synapse, ...))
    draw.text(label = out.label[(neuron.count[out] +
      1) - i], x = c((x.position[out] + radius), x.position[out] +
      arrow.length), y = c(y.position[out], y.position[out]),
      xy.null = c(0, 0), color = col.out.synapse,
      fontsize = fontsize, ...)
    grid.circle(x = x.position[out], y = y.position[out],
      r = radius, gp = gpar(fill = "white", col = col.out,
        ...))
  }
  if (intercept) {
    for (k in 1:length(weights)) {
      y.tmp <- 0
      x.intercept <- (x.position[k + 1] - x.position[k]) *
```

```
          intercept.factor + x.position[k]
        for (i in 1:neuron.count[k + 1]) {
          y.tmp <- y.tmp + y.step[k + 1]
          result <- calculate.delta(c(x.intercept, x.position[k +
            1]), c(y.intercept, y.tmp), radius)
          x <- c(x.intercept, x.position[k + 1] - result[1])
          y <- c(y.intercept, y.tmp + result[2])
          grid.lines(x = x, y = y, arrow = arrow(length = unit(0.15,
            "cm"), type = "closed"), gp = gpar(fill = col.intercept,
            col = col.intercept, ...))
          xy.null <- cbind(x.position[k + 1] - x.intercept -
            2 * result[1], -(y.tmp - y.intercept + 2 *
            result[2]))
          if (show.weights)
            draw.text(label = weights[[k]][1, neuron.count[k +
              1] - i + 1], x = c(x.intercept, x.position[k +
              1]), y = c(y.intercept, y.tmp), xy.null = xy.null,
              color = col.intercept, alignment = c("right",
                "bottom"), fontsize = fontsize - 2,
              ...)
        }
        grid.circle(x = x.intercept, y = y.intercept,
          r = radius, gp = gpar(fill = "white", col = col.intercept,
            ...))
        grid.text(1, x = x.intercept, y = y.intercept,
          gp = gpar(col = col.intercept, ...))
      }
    }
    if (information)
      grid.text(paste("Error: ", round(result.matrix[rep,
        "error"], 6), "   Steps: ", result.matrix[rep,
        "steps"], sep = ""), x = 0.5, y = information.pos,
        just = "bottom", gp = gpar(fontsize = fontsize +
          2, ...))
    if (!is.null(file)) {
      weight.plot <- recordPlot()
      save(weight.plot, file = file)
    }
  }
}
```

Simple plot, intercept & weights are turned off.

```
##par (mfrow = c(1,1))
plotnn(fit, intercept = FALSE, show.weights = FALSE)
```




To see what `fit` actually contains, run:

```
attributes(fit)
```

```
## $names
##  [1] "call"                "response"            "covariate"
##  [4] "model.list"          "err.fct"             "act.fct"
##  [7] "linear.output"       "data"                "net.result"
## [10] "weights"             "startweights"        "generalized.weights"
## [13] "result.matrix"
##
## $class
## [1] "nn"
```

To check details of a particular attribute, use the `$` operator. For example, check the summary of the fitted network contained in `result.matrix`:

```
fit$result.matrix
```

```
##                                    1
## error                 1854.659153417289
## reached.threshold        0.008971185555
## steps                15376.000000000000
## Intercept.to.1layhid1   -1.476505618519
## hosp.to.1layhid1         0.936995772797
```

```
## health.to.1layhid1       -0.308589104362
## numchron.to.1layhid1      1.065004008360
## gender.to.1layhid1       -0.190692443760
## school.to.1layhid1        0.159883623634
## privins.to.1layhid1       0.363787899579
## Intercept.to.Class        1.648630276189
## 1layhid.1.to.Class      -40.196940113888
```

The value goven for `hosp.to.1layhid1` is the calculated optimum weight of the synapse between `hosp` and the hidden neuron.

### Predicting new cases

Method `compute` from the neuralnet package computes the output of all neurons given a trained neural network using (the same!) covariate vectors.

```
pred<-compute(fit, data[-train, 2:7])
```

View the first few predictions from $net.result

```
## show attributes in pred
attributes(pred)
```

```
## $names
## [1] "neurons"    "net.result"
```

```
## get top results
head(pred$net.result)
```

```
##                     [,1]
## 1  0.000001149730617861
## 15 0.000004387003430401
## 16 0.666976073629486299
## 46 0.000000001263003749
## 76 0.000000469604180798
## 82 0.000217696667194658
```

These numbers give the probability of an individual belonging to the below median or above median. Let's convert them back to the same -1, +1 scale as used in `Class`.

```
r2 <- ifelse(pred$net.result<=0.5,-1,1)
head(r2)
```

```
##    [,1]
## 1    -1
## 15   -1
## 16    1
## 46   -1
## 76   -1
## 82   -1
```

## Check Results

Build a confusion matrix:

```
table(sign(r2), sign(data[-train,1]), dnn=c("Predicted", "Observed"))
```

```
##          Observed
## Predicted  -1   1
##        -1 205 157
##        1   11  33
```

Of the 406 observations 205 were correctlt classified as belonging to group -1, and 33 were correctly classified as belonging to group +1. The error rate is calculated measuring the misclassified observations as proportion of the total:

```
error_rate = (1- sum(sign(r2)==sign(data[-train,1]))/length(data[-train,1]))
round(error_rate, 2)
```

```
## [1] 0.41
```

Overall 41% of individuals were misclassified. This implies a prediction accuray of around 59%.

Nog beter: gebruik caret

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.1.3
```

```
## Loading required package: lattice
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 3.1.3
```

```
confusionMatrix(data=r2,reference = sign(data[-train,1]) )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  -1   1
##         -1 205 157
##         1   11  33
##
##                Accuracy : 0.5862069
##                  95% CI : (0.536584, 0.6345639)
##     No Information Rate : 0.5320197
##     P-Value [Acc > NIR] : 0.01602206
##
##                   Kappa : 0.1287108
##  Mcnemar's Test P-Value : < 0.00000000000000222
##
##             Sensitivity : 0.9490741
```

```
##             Specificity : 0.1736842
##          Pos Pred Value : 0.5662983
##          Neg Pred Value : 0.7500000
##              Prevalence : 0.5320197
##          Detection Rate : 0.5049261
##    Detection Prevalence : 0.8916256
##       Balanced Accuracy : 0.5613791
##
##        'Positive' Class : -1
##
```

# Excercises

## Question 1

Re-build the model, but this time using six hidden nodes Maak een MLP met zes hidden layers

```
set.seed(103)
fit.q1 <- neuralnet(f, data = data[train,], hidden = 6, algorithm = "rprop+",
                    err.fct = "sse", act.fct = "logistic",
                    linear.output = FALSE)
```

### Q1 Results

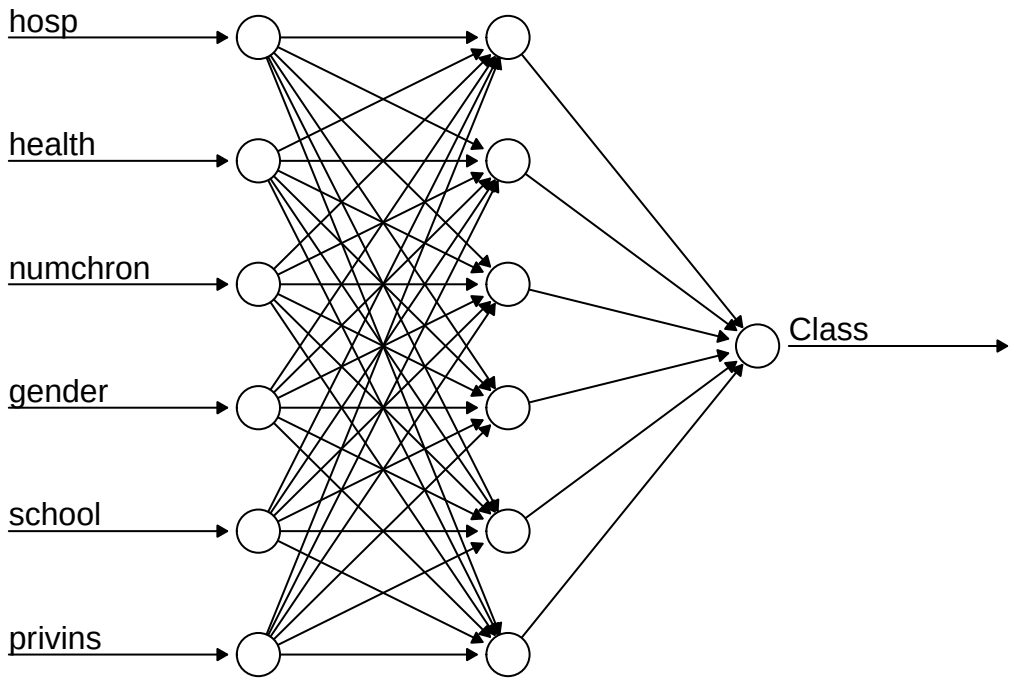
```
print(fit.q1)
```

```
## Call: neuralnet(formula = f, data = data[train, ], hidden = 6, algorithm = "rprop+",     err.fct = "
##
## 1 repetition was calculated.
##
##          Error Reached Threshold Steps
## 1 1828.890459    0.009396198984  3698
```

The model converged after 3.698 steps with an error of 1829.

Simple plot, intercept & weights are turned off.

```
plotnn(fit.q1, intercept = FALSE, show.weights = FALSE)
```




To check details of a particular attribute, use the `$` operator. For example, check the summary of the fitted network contained in `result.matrix`:

```
fit.q1$result.matrix
```

```
##                                    1
## error                 1828.890458758729
## reached.threshold        0.009396198984
## steps                 3698.000000000000
## Intercept.to.1layhid1   -0.423447092047
## hosp.to.1layhid1        25.652913369638
## health.to.1layhid1     -29.078245640452
## numchron.to.1layhid1    35.197525071508
## gender.to.1layhid1      -8.269092230283
## school.to.1layhid1     -11.301181730922
## privins.to.1layhid1     13.546659283759
## Intercept.to.1layhid2    6.935431125606
## hosp.to.1layhid2        -4.496416358485
## health.to.1layhid2      -8.027431561327
## numchron.to.1layhid2    -0.446288483916
## gender.to.1layhid2       4.023964410207
## school.to.1layhid2      -1.325467311502
## privins.to.1layhid2      8.635792296373
## Intercept.to.1layhid3   -0.255934305279
## hosp.to.1layhid3        15.742935399865
## health.to.1layhid3       3.988391663041
```

```
## numchron.to.1layhid3    39.842999489774
## gender.to.1layhid3       2.926658697943
## school.to.1layhid3      16.803364678757
## privins.to.1layhid3      9.332931392087
## Intercept.to.1layhid4   -0.115168770514
## hosp.to.1layhid4       -33.404504874158
## health.to.1layhid4     -16.018131378684
## numchron.to.1layhid4   -63.848494692455
## gender.to.1layhid4       0.275023894060
## school.to.1layhid4     -15.231794136035
## privins.to.1layhid4    -30.010214935539
## Intercept.to.1layhid5    2.897128715511
## hosp.to.1layhid5         0.751106103924
## health.to.1layhid5      -5.131686199332
## numchron.to.1layhid5     2.627657202005
## gender.to.1layhid5      -0.752776696962
## school.to.1layhid5       0.512525226472
## privins.to.1layhid5     -0.864357341599
## Intercept.to.1layhid6   -8.017062403861
## hosp.to.1layhid6        40.598224383093
## health.to.1layhid6      12.401076005280
## numchron.to.1layhid6    27.942416386952
## gender.to.1layhid6      -2.669361683444
## school.to.1layhid6      27.402140833438
## privins.to.1layhid6     29.260771923642
## Intercept.to.Class      -0.985088675130
## 1layhid.1.to.Class    -252.166172075793
## 1layhid.2.to.Class      -2.479937906277
## 1layhid.3.to.Class    -245.656856629309
## 1layhid.4.to.Class       3.940218969832
## 1layhid.5.to.Class      -3.441144063418
## 1layhid.6.to.Class      -2.957627410226
```

### Q1 Predicting new cases

Method `compute` from the neuralnet package computes the output of all neurons given a trained neural network using (the same!) covariate vectors.

```
pred.q1<-compute(fit.q1, data[-train, 2:7])
```

These numbers give the probability of an individual belonging to the below median or above median. Let's convert them back to the same -1, +1 scale as used in `Class`.

```
r2.q1 <- ifelse(pred.q1$net.result<=0.5,-1,1)
head(r2.q1)
```

```
##    [,1]
## 1    -1
## 15   -1
## 16    1
## 46   -1
## 76   -1
## 82   -1
```

### Check Results Q1

Build a confusion matrix:

```
table(sign(r2.q1), sign(data[-train,1]), dnn=c("Predicted", "Observed"))
```

```
##          Observed
## Predicted  -1   1
##        -1 203 158
##        1   13  32
```

Of the 406 observations 201 were correctlt classified as belonging to group -1, and 35 were correctly classified as belonging to group +1. The error rate is calculated measuring the misclassified observations as proportion of the total:

```
error_rate.q1 = (1- sum(sign(r2.q1)==sign(data[-train,1]))/length(data[-train,1]))
round(error_rate.q1, 2)
```

```
## [1] 0.42
```

Overall 42% of individuals were misclassified. This implies a prediction accuray of around 58%. Nog beter: gebruik caret

```
library(caret)
confusionMatrix(data=r2.q1,reference = sign(data[-train,1]) )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  -1   1
##         -1 203 158
##         1   13  32
##
##                Accuracy : 0.5788177
##                  95% CI : (0.5291297, 0.6273485)
##     No Information Rate : 0.5320197
##     P-Value [Acc > NIR] : 0.03262661
##
##                   Kappa : 0.1134466
##  Mcnemar's Test P-Value : < 0.000000000000000222
##
##             Sensitivity : 0.9398148
##             Specificity : 0.1684211
##          Pos Pred Value : 0.5623269
##          Neg Pred Value : 0.7111111
##              Prevalence : 0.5320197
##          Detection Rate : 0.5000000
##    Detection Prevalence : 0.8891626
##       Balanced Accuracy : 0.5541179
##
##        'Positive' Class : -1
##
```

## Question 2

Re-estimate the model build in question 1, but using resilient backpropagation without backtracking.

Maak een MLP met zes hidden layers

```
set.seed(103)
fit.q2 <- neuralnet(f, data = data[train,], hidden = 6, algorithm = "rprop-",
                    err.fct = "sse", act.fct = "logistic",
                    linear.output = FALSE)
```

### Q2 Results

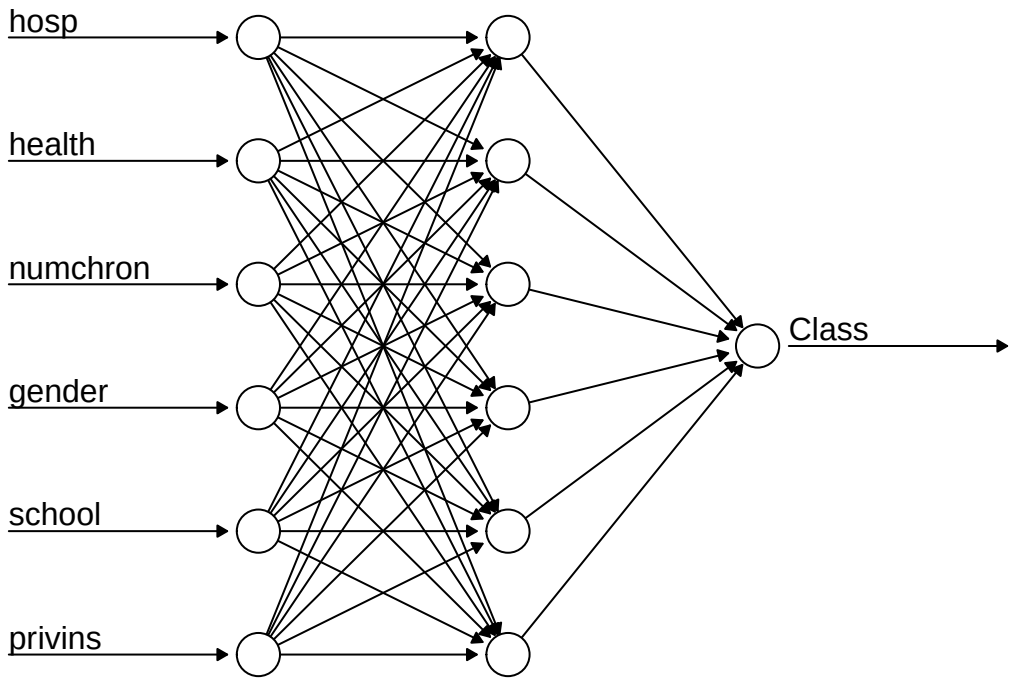
```
print(fit.q2)
```

```
## Call: neuralnet(formula = f, data = data[train, ], hidden = 6, algorithm = "rprop-",     err.fct = "
##
## 1 repetition was calculated.
##
##         Error Reached Threshold Steps
## 1 1823.284806    0.009739367719 28568
```

The model converged after 28.568 steps with an error of 1823.

Simple plot, intercept & weights are turned off.

```
plotnn(fit.q2, intercept = FALSE, show.weights = FALSE)
```




To check details of a particular attribute, use the `$` operator. For example, check the summary of the fitted network contained in `result.matrix`:

```
fit.q2$result.matrix
```

```
##                                      1
## error                   1823.284806030894
## reached.threshold          0.009739367719
## steps                  28568.000000000000
## Intercept.to.1layhid1      3.057835545610
## hosp.to.1layhid1          -0.535743490727
## health.to.1layhid1        -8.515857321455
## numchron.to.1layhid1       3.876165905862
## gender.to.1layhid1        -0.696248051676
## school.to.1layhid1        -1.176874225339
## privins.to.1layhid1        1.253137569104
## Intercept.to.1layhid2     24.340652059980
## hosp.to.1layhid2          -2.015005763762
## health.to.1layhid2       -44.343409191050
## numchron.to.1layhid2      54.584835965681
## gender.to.1layhid2        33.406802665794
## school.to.1layhid2       -19.211713352079
## privins.to.1layhid2       19.840455575198
## Intercept.to.1layhid3      5.697902890758
## hosp.to.1layhid3          14.606761120051
## health.to.1layhid3        29.469395377339
```

```
## numchron.to.1layhid3    145.673555107605
## gender.to.1layhid3       11.094037573479
## school.to.1layhid3       47.007263234492
## privins.to.1layhid3      34.915257246065
## Intercept.to.1layhid4     2.800096043635
## hosp.to.1layhid4        -65.725138726742
## health.to.1layhid4      -25.945546434082
## numchron.to.1layhid4   -104.742776612260
## gender.to.1layhid4       23.426772226935
## school.to.1layhid4       24.763469511725
## privins.to.1layhid4     -72.365713717550
## Intercept.to.1layhid5     2.170784122503
## hosp.to.1layhid5          4.332640394469
## health.to.1layhid5       -9.379252769875
## numchron.to.1layhid5     -1.728445851798
## gender.to.1layhid5       -0.363541361119
## school.to.1layhid5        0.714046916596
## privins.to.1layhid5      -0.597306920004
## Intercept.to.1layhid6    -8.563690192907
## hosp.to.1layhid6          1.230028435242
## health.to.1layhid6        2.319895454026
## numchron.to.1layhid6     -0.597510391351
## gender.to.1layhid6       -0.750306871965
## school.to.1layhid6        2.757249994580
## privins.to.1layhid6       6.917432429675
## Intercept.to.Class        2.716843198117
## 1layhid.1.to.Class       -7.488769394344
## 1layhid.2.to.Class       -3.793820235659
## 1layhid.3.to.Class       -4.716519973559
## 1layhid.4.to.Class        9.303512444689
## 1layhid.5.to.Class      -12.190836530598
## 1layhid.6.to.Class      -12.244292191868
```

### Q2 Predicting new cases

Method `compute` from the neuralnet package computes the output of all neurons given a trained neural network using (the same!) covariate vectors.

```
pred.q2<-compute(fit.q2, data[-train, 2:7])
```

These numbers give the probability of an individual belonging to the below median or above median. Let's convert them back to the same -1, +1 scale as used in `Class`.

```
r2.q2 <- ifelse(pred.q2$net.result<=0.5,-1,1)
head(r2.q2)
```

```
##    [,1]
## 1    -1
## 15   -1
## 16    1
## 46   -1
## 76   -1
## 82   -1
```

### Check Results Q2

Build a confusion matrix:

```
table(sign(r2.q2), sign(data[-train,1]), dnn=c("Predicted", "Observed"))
```

```
##          Observed
## Predicted  -1   1
##        -1 201 155
##        1   15  35
```

Of the 406 observations 201 were correctly classified as belonging to group -1, and 35 were correctly classified as belonging to group +1. The error rate is calculated measuring the misclassified observations as proportion of the total:

```
error_rate.q2 = (1- sum(sign(r2.q2)==sign(data[-train,1]))/length(data[-train,1]))
round(error_rate.q2, 2)
```

```
## [1] 0.42
```

Overall 42% of individuals were misclassified. This implies a prediction accuray of around 58%. Nog beter: gebruik caret

```
library(caret)
confusionMatrix(data=r2.q2,reference = sign(data[-train,1]) )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  -1   1
##         -1 201 155
##         1   15  35
##
##                Accuracy : 0.5812808
##                  95% CI : (0.5316132, 0.6297549)
##     No Information Rate : 0.5320197
##     P-Value [Acc > NIR] : 0.02597266
##
##                   Kappa : 0.1200918
##  Mcnemar's Test P-Value : < 0.000000000000000222
##
##             Sensitivity : 0.9305556
##             Specificity : 0.1842105
##          Pos Pred Value : 0.5646067
##          Neg Pred Value : 0.7000000
##              Prevalence : 0.5320197
##          Detection Rate : 0.4950739
##    Detection Prevalence : 0.8768473
##       Balanced Accuracy : 0.5573830
##
##        'Positive' Class : -1
##
```

## Question 3

Suppose a domain expert informed you only `hosp`, `health` and `numchron` were relevant attributes. Build a model with 2 hidden nodes using resilient backpropagation without backtracking.

Maak nieuwe dataset met alleen de relevante variabelen.

```
keeps.q3<-c("Class", "hosp", "health", "numchron")
data.q3<-data[,keeps.q3]
head(data.q3)
```

```
##           Class          hosp       health     numchron
## 1 -0.9166480631  0.9432503020  0.106727104 0.3393605955
## 2  1.0906836298 -0.3965178575  0.106727104 0.3393605955
## 3 -0.9166480631  3.6227866210 -2.121896688 1.8212464071
## 4 -0.9166480631  0.9432503020 -2.121896688 0.3393605955
## 5  1.0906836298 -0.3965178575  0.106727104 0.3393605955
## 6 -0.9166480631 -0.3965178575 -2.121896688 2.5621893129
```

Maak een nieuwe formule:

```
f.q3<-Class ~ hosp + health + numchron
```

Maak een MLP met twee hidden layers

```
set.seed(103)
fit.q3 <- neuralnet(f.q3, data = data.q3[train,], hidden = 2,
                    algorithm = "rprop-", err.fct = "sse",
                    act.fct = "logistic", linear.output = FALSE)
```

### Q3 Results

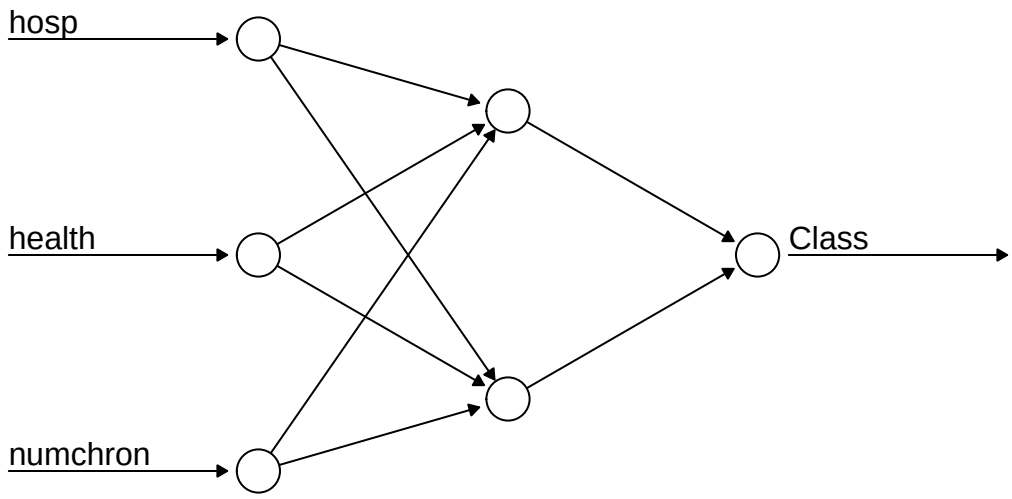
```
print(fit.q3)
```

```
## Call: neuralnet(formula = f.q3, data = data.q3[train, ], hidden = 2,     algorithm = "rprop-", err.f
##
## 1 repetition was calculated.
##
##         Error Reached Threshold Steps
## 1 1878.360131    0.008920075304  1119
```

The model converged after 1.119 steps with an error of 1878.

Simple plot, intercept & weights are turned off.

```
plotnn(fit.q3, intercept = FALSE, show.weights = FALSE)
```




To check details of a particular attribute, use the `$` operator. For example, check the summary of the fitted network contained in `result.matrix`:

```
fit.q3$result.matrix
```

```
##                                    1
## error                 1878.360130962280
## reached.threshold        0.008920075304
## steps                 1119.000000000000
## Intercept.to.1layhid1    1.151758601210
## hosp.to.1layhid1         1.757383420677
## health.to.1layhid1      -8.903994031155
## numchron.to.1layhid1     1.265204391022
## Intercept.to.1layhid2    0.958555555696
## hosp.to.1layhid2         5.457862673942
## health.to.1layhid2       1.047973107247
## numchron.to.1layhid2     6.904970816420
## Intercept.to.Class       0.863040590969
## 1layhid.1.to.Class      -8.396997863382
## 1layhid.2.to.Class     -12.446359336159
```

**Q3 Predicting new cases**

Method `compute` from the neuralnet package computes the output of all neurons given a trained neural network using (the same!) covariate vectors.

```
pred.q3<-compute(fit.q3, data.q3[-train, 2:4])
```

These numbers give the probability of an individual belonging to the below median or above median. Let's convert them back to the same -1, +1 scale as used in `Class`.

```
r2.q3 <- ifelse(pred.q3$net.result<=0.5,-1,1)
head(r2.q3)
```

```
##    [,1]
## 1    -1
## 15   -1
## 16   -1
## 46   -1
## 76   -1
## 82   -1
```

### Check Results Q3

Build a confusion matrix:

```
table(sign(r2.q3), sign(data.q3[-train,1]), dnn=c("Predicted", "Observed"))
```

```
##          Observed
## Predicted  -1   1
##        -1 214 178
##        1    2  12
```

Of the 406 observations 214 were correctly classified as belonging to group -1, and 12 were correctly classified as belonging to group +1. The error rate is calculated measuring the misclassified observations as proportion of the total:

```
error_rate.q3 = (1- sum(sign(r2.q3)==sign(data.q3[-train,1]))/length(data.q3[-train,1]))
round(error_rate.q3, 2)
```

```
## [1] 0.44
```

Overall 44% of individuals were misclassified. This implies a prediction accuray of around 56%. Nog beter: gebruik caret

```
library(caret)
confusionMatrix(data=r2.q3,reference = sign(data.q3[-train,1]) )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  -1   1
##         -1 214 178
##         1    2  12
##
```

```
##                Accuracy : 0.5566502
##                  95% CI : (0.5068319, 0.6056369)
##     No Information Rate : 0.5320197
##     P-Value [Acc > NIR] : 0.172391
##
##                   Kappa : 0.0570809
##  Mcnemar's Test P-Value : < 0.0000000000000002
##
##             Sensitivity : 0.99074074
##             Specificity : 0.06315789
##          Pos Pred Value : 0.54591837
##          Neg Pred Value : 0.85714286
##              Prevalence : 0.53201970
##          Detection Rate : 0.52709360
##    Detection Prevalence : 0.96551724
##       Balanced Accuracy : 0.52694932
##
##        'Positive' Class : -1
##
```