Machine Learning Exam January 2020

Assignment 1

```r
# import data
glass <- read.csv("glass.csv")

# preparing data
glass$Class <- as.factor(glass$Class)
glass <- glass[,-1]

#splitting data
n=dim(glass)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.4))
train=glass[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.3))
valid=glass[id2,]
id3=setdiff(id1,id2)
test=glass[id3,]

# combine train & valid data
trval=rbind(train,valid)
```

```r
# fitting the logistic regression model
log_reg <- glm(Class~., data=trval, family=binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
# predict test data
# type = response for numerical values between 0 and 1
pred_test <- predict(log_reg, newdata = test, type = "response")

# determine the threshold value
pred_test=as.numeric(pred_test>0.5)

# confusion matrix
cm <- table(test$Class, pred_test)
```

The prediction of the model is good with only 8 misclassifications out of 65.

```r
knitr::kable(cm)
```

	0	1
0	18	5
1	3	39

The fitted probabilistic model is given by the formula:

$$p(y = Class|w, x) = \frac{1}{1 + exp(-w_0 - \sum_1^9 w_i x_i)}$$

Where:

$$-w_0 - \sum_1^9 w_i x_i = 12081.97 - 3915.57RI - 50.77Na - 33.88Mg - 94.59Al - 66.46Si - 40K - 40.48Ca - 52.4Ba + 88.57Fe$$

The decision boundary is:

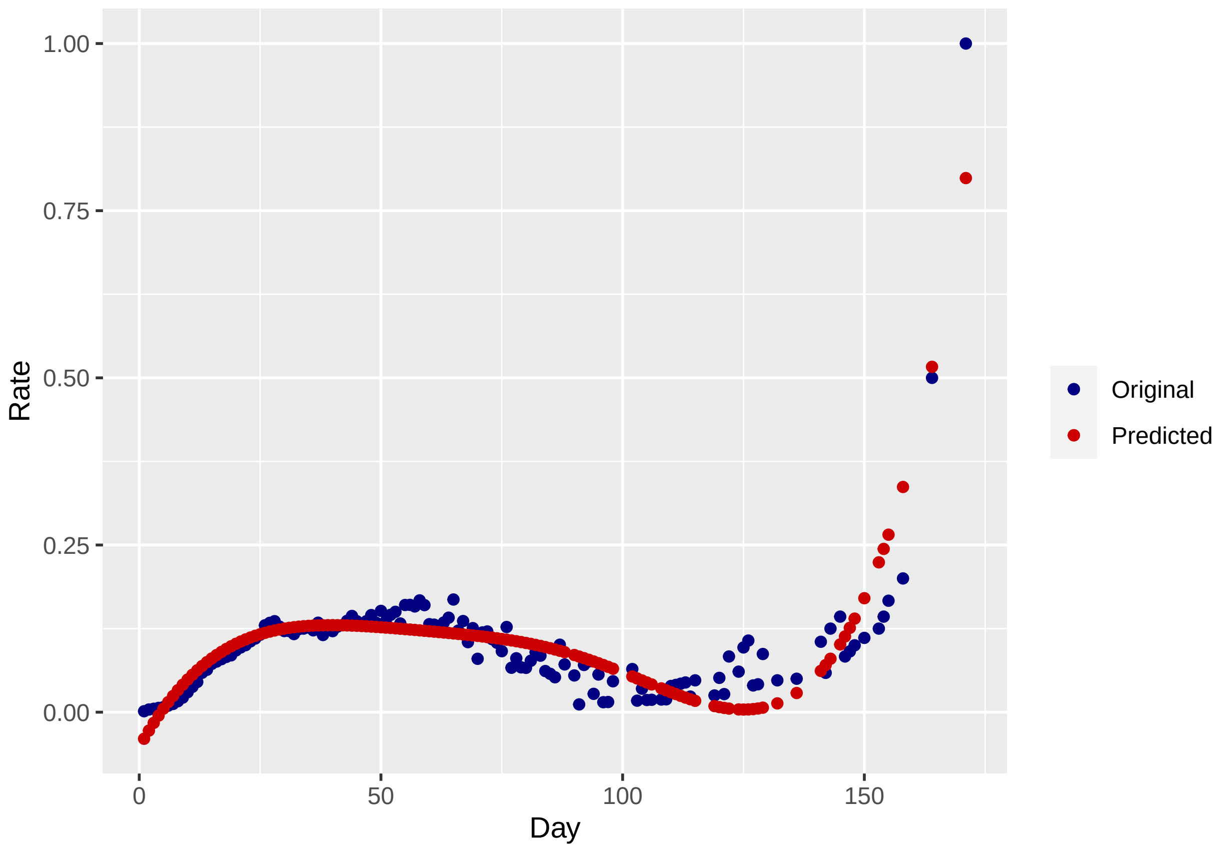
$$12081.97 - 3915.57RI - 50.77Na - 33.88Mg - 94.59Al - 66.46Si - 40K - 40.48Ca - 52.4Ba + 88.57Fe$$

Assignment 2

```
# import data
mortality_rate <- read.csv2("mortality_rate.csv")

# preparing data
X  <- mortality_rate$Day
data <-data.frame(X1=X,
                  X2=X^2,
                  X3=X^3,
                  X4=X^4,
                  X5=ifelse(X-75>0, X-75,0)^4,
                  Y=mortality_rate$Rate)
```

```
#fitting the linear regression model
lin_reg <- lm(Y~., data = data)
pred <- predict(lin_reg)
```

```
# preparing data for plot
df_plot <- cbind(mortality_rate,pred)

# plot original & predicted data
ggplot() +
  geom_point(data = df_plot, aes(x=Day,y=Rate, color = "navy")) +
  geom_point(data = df_plot, aes(x=Day,y=pred, color = "red3")) +
  theme(legend.position="right") +
  scale_color_manual(values=c("navy","red3"),
                     name = "",
                     labels = c("Original","Predicted"))
```



From the above plot the model seems to be underfitted. The reasons that this might happen is because of the either the 5-spline order or the value of the knot.

```
summary(lin_reg)
```

```
##
## Call:
## lm(formula = Y ~ ., data = data)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.136799 -0.016559 -0.002466  0.016241  0.201196
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.262e-02  1.923e-02  -2.737 0.007063 **
## X1           1.319e-02  2.251e-03   5.859 3.62e-08 ***
## X2          -3.420e-04  7.719e-05  -4.431 1.98e-05 ***
## X3           3.804e-06  9.839e-07   3.866 0.000174 ***
## X4          -1.607e-08  4.179e-09  -3.844 0.000188 ***
## X5           3.898e-08  6.604e-09   5.903 2.92e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.03909 on 130 degrees of freedom
## Multiple R-squared:  0.8452, Adjusted R-squared:  0.8393
## F-statistic:   142 on 5 and 130 DF,  p-value: < 2.2e-16
```

The third and fourth degree terms of the model were necessary with the significant level of 0.001.

```
df <- function(input_data){

  X <- as.matrix(input_data[,-6])
  I <- diag(ncol(X))
  hat_matrix <- X %*% solve(t(X) %*% X) %*% t(X)

  degrees_of_freedom <- sum(diag(hat_matrix))

  return(degrees_of_freedom)
}

df(data)
```

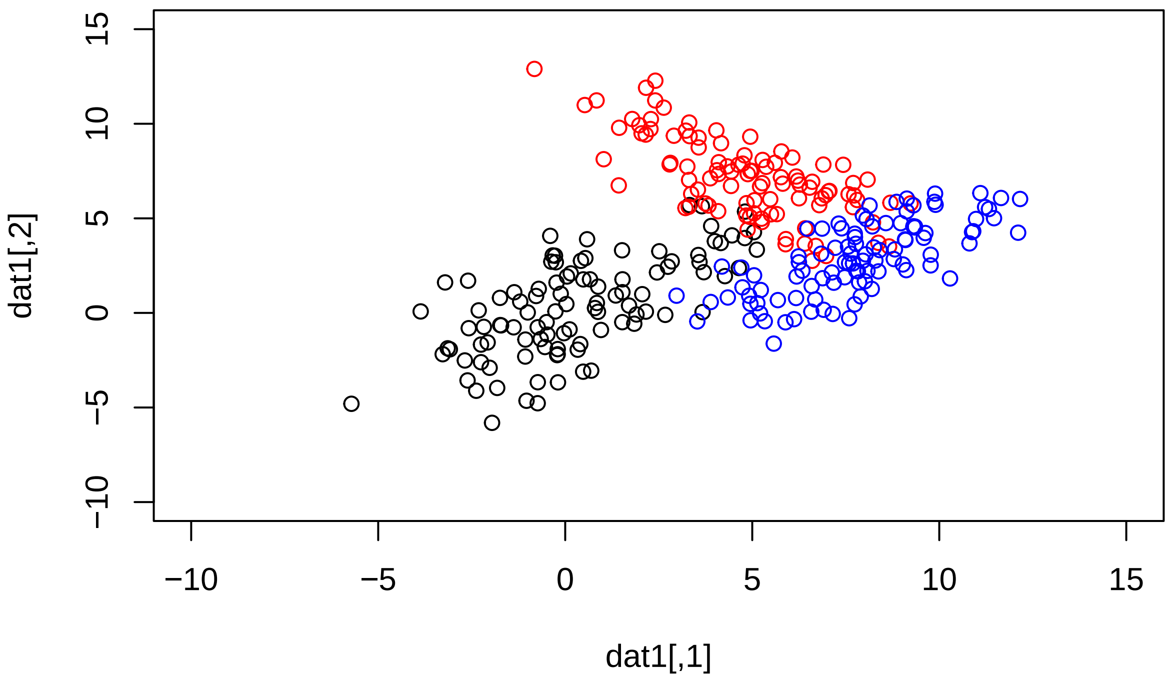
```
## [1] 5
```

The degrees of freedom are 6 (5 + the intercept), which is the amount of the parameters of the fitted linear regression model. Therefore the model is parametric, because whatever data we take the degrees of freedom will always be 6 and thus the complexity of the data will remain the same.

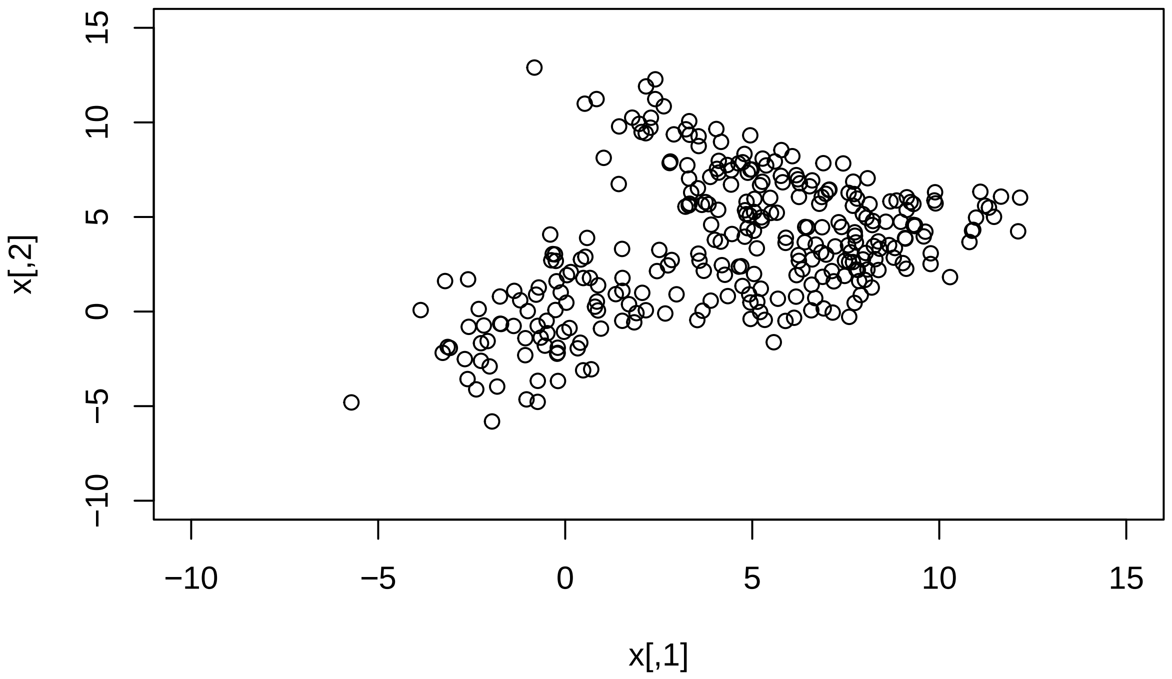
Assignment 4

Mixture Gaussian Models

```
library(mvtnorm)

set.seed(1234567890)

max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log likelihood between two consecutive EM iterations
N=300 # number of training points
D=2 # number of dimensions
x <- matrix(nrow=N, ncol=D) # training data

# Producing the training data
mu1<-c(0,0)
Sigma1 <- matrix(c(5,3,3,5),D,D)
dat1<-rmvnorm(n = 100, mu1, Sigma1)
mu2<-c(5,7)
Sigma2 <- matrix(c(5,-3,-3,5),D,D)
dat2<-rmvnorm(n = 100, mu2, Sigma2)
mu3<-c(8,3)
Sigma3 <- matrix(c(3,2,2,3),D,D)
dat3<-rmvnorm(n = 100, mu3, Sigma3)
plot(dat1,xlim=c(-10,15),ylim=c(-10,15))
points(dat2,col="red")
points(dat3,col="blue")
```



```
x[1:100,]<-dat1
x[101:200,]<-dat2
x[201:300,]<-dat3
plot(x,xlim=c(-10,15),ylim=c(-10,15))
```



```
K=3 # number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional means
Sigma <- array(dim=c(D,D,K)) # conditional covariances
llik <- vector(length = max_it) # log likelihood of the EM iterations
```

```
# Random initialization of the parameters
pi <- runif(K,0,1)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0,5)
  Sigma[,,k]<-c(1,0,0,1)
}
pi
```

```
## [1] 0.53980633 0.37254500 0.08764867
```

```
mu
```

```
##           [,1]      [,2]
## [1,] 0.5506079 0.9707686
## [2,] 4.4511697 4.1405008
## [3,] 4.2580319 2.7822372
```

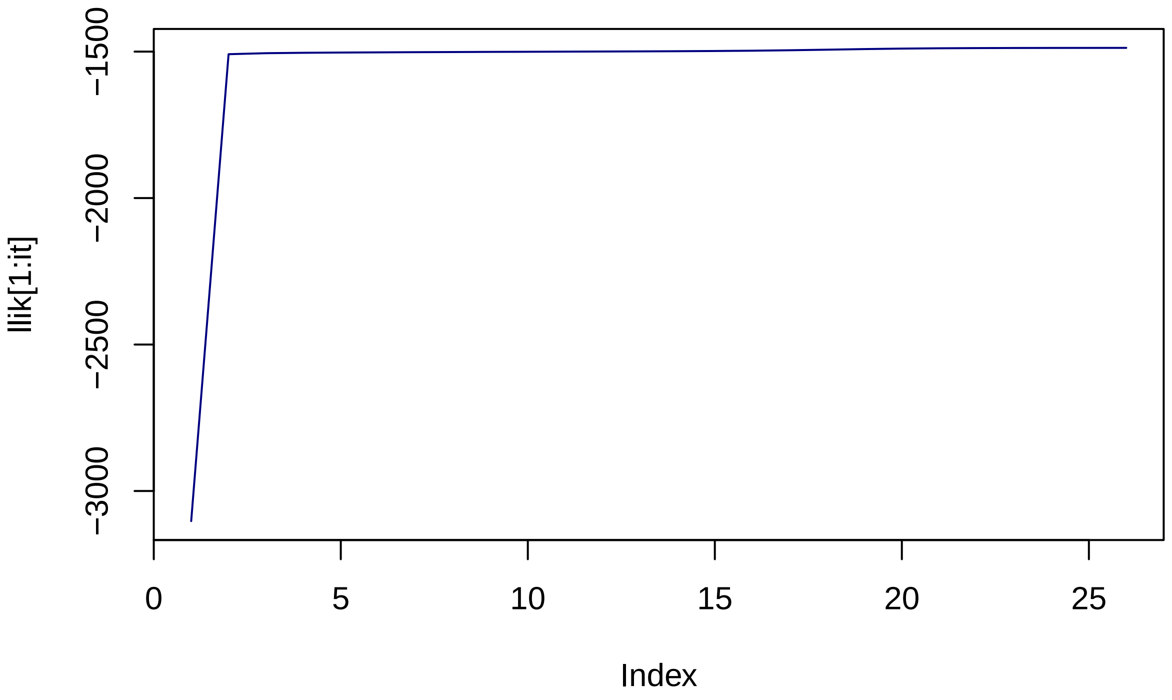
```
Sigma
```

```
## , , 1
##
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
##
## , , 2
##
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
##
## , , 3
##
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

```
for(it in 1:max_it) {
  # E-step: Computation of the fractional component assignments
  llik[it] <- 0
  for(n in 1:N) {
    for(k in 1:K) {
      z[n,k] <- pi[k]*dmvnorm(x[n,],mu[k,],Sigma[,,k])
    }

    #Log likelihood computation.
    llik[it] <- llik[it] + log(sum(z[n,]))

    z[n,] <- z[n,]/sum(z[n,])
  }

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the lok likelihood has not changed significantly
  if (it > 1) {
    if(abs(llik[it] - llik[it-1]) < min_change) {
      break
    }
  }

  #M-step: ML parameter estimation from the data and fractional component assignments
  for(k in 1:K) {
    pi[k] <- sum(z[,k]) / N
    for(d in 1:D) {
      mu[k, d] <- sum(x[, d] * z[, k]) / sum(z[,k])
    }
    for(d in 1:D) {
      for(d2 in 1:D) {
        Sigma[d,d2,k]<-sum((x[, d]-mu[k,d]) * (x[, d2]-mu[k,d2]) * z[, k]) / sum(z[,k])
      }
    }
  }
}
```

```
## iteration:  1 log likelihood:  -3102.839
## iteration:  2 log likelihood:  -1508.843
## iteration:  3 log likelihood:  -1505.373
## iteration:  4 log likelihood:  -1504.041
## iteration:  5 log likelihood:  -1503.199
## iteration:  6 log likelihood:  -1502.533
## iteration:  7 log likelihood:  -1501.954
## iteration:  8 log likelihood:  -1501.434
## iteration:  9 log likelihood:  -1500.958
## iteration:  10 log likelihood:  -1500.512
## iteration:  11 log likelihood:  -1500.082
## iteration:  12 log likelihood:  -1499.649
## iteration:  13 log likelihood:  -1499.182
## iteration:  14 log likelihood:  -1498.632
## iteration:  15 log likelihood:  -1497.914
## iteration:  16 log likelihood:  -1496.887
## iteration:  17 log likelihood:  -1495.387
## iteration:  18 log likelihood:  -1493.416
## iteration:  19 log likelihood:  -1491.349
## iteration:  20 log likelihood:  -1489.687
## iteration:  21 log likelihood:  -1488.612
## iteration:  22 log likelihood:  -1487.994
## iteration:  23 log likelihood:  -1487.651
## iteration:  24 log likelihood:  -1487.458
## iteration:  25 log likelihood:  -1487.348
## iteration:  26 log likelihood:  -1487.284
```

```
plot(llik[1:it], type="l", col = "navy")
```



The number of parameters in a MM model is the number of mixing coefficients minus 1 (because they all have to sum up to 1), plus the number of elements in the mean vector for each component, plus the number of entries in the covariance matrix for each component (since this matrix is symmetric, we only count the upper diagonal part of it). That is, K-1 mixing coefficients, plus K*D mean elements, plus K*D*(D+1)/2 covariance elements.

```
BIC<-llik[it] - log(N) * 0.5 * ((K-1)+K*D+K*D*(D+1)/2)
BIC
```

```
## [1] -1535.766
```

```
# Producing the validation data
dat1<-rmvnorm(n = 1000, mu1, Sigma1)
dat2<-rmvnorm(n = 1000, mu2, Sigma2)
dat3<-rmvnorm(n = 1000, mu3, Sigma3)
v <- matrix(nrow=3000, ncol=D) # validation data
v[1:1000,]<-dat1
v[1001:2000,]<-dat2
v[2001:3000,]<-dat3
z <- matrix(nrow=3000, ncol=K)

vllik <- 0
for(n in 1:3000) {
  for(k in 1:K) {
    z[n,k] <- pi[k]*dmvnorm(v[n,],mu[k,],Sigma[,,k])
  }

  #Log likelihood computation.
  vllik <- vllik + log(sum(z[n,]))
}

pi
```

```
## [1] 0.2993551 0.3616209 0.3390240
```

```
mu
```

```
##             [,1]       [,2]
## [1,] -0.1096711 -0.0234151
## [2,]  4.9736528  6.8964965
## [3,]  7.5481552  2.8716299
```

```
Sigma
```

```
## , , 1
##
##          [,1]     [,2]
## [1,] 3.940797 2.706445
## [2,] 2.706445 5.796523
##
## , , 2
##
##           [,1]      [,2]
## [1,]  4.513017 -3.317680
## [2,] -3.317680  5.154163
##
## , , 3
##
##          [,1]     [,2]
## [1,] 4.638160 2.828176
## [2,] 2.828176 3.861886
```

```
llik[it]
```

```
## [1] -1487.284
```

```
BIC
```

```
## [1] -1535.766
```

```
vllik
```

```
## [1] -14903.88
```

```
K
```

```
## [1] 3
```

BIC for K=2: -1547.347
BIC for K=3: -1535.766
BIC for K=4: -1547.366
So, K=3 wins.

Validation log lik for K=2: -15225.83
Validation log lik for K=3: -14903.88
Validation log lik for K=4: -14982.4
So, K=3 wins.

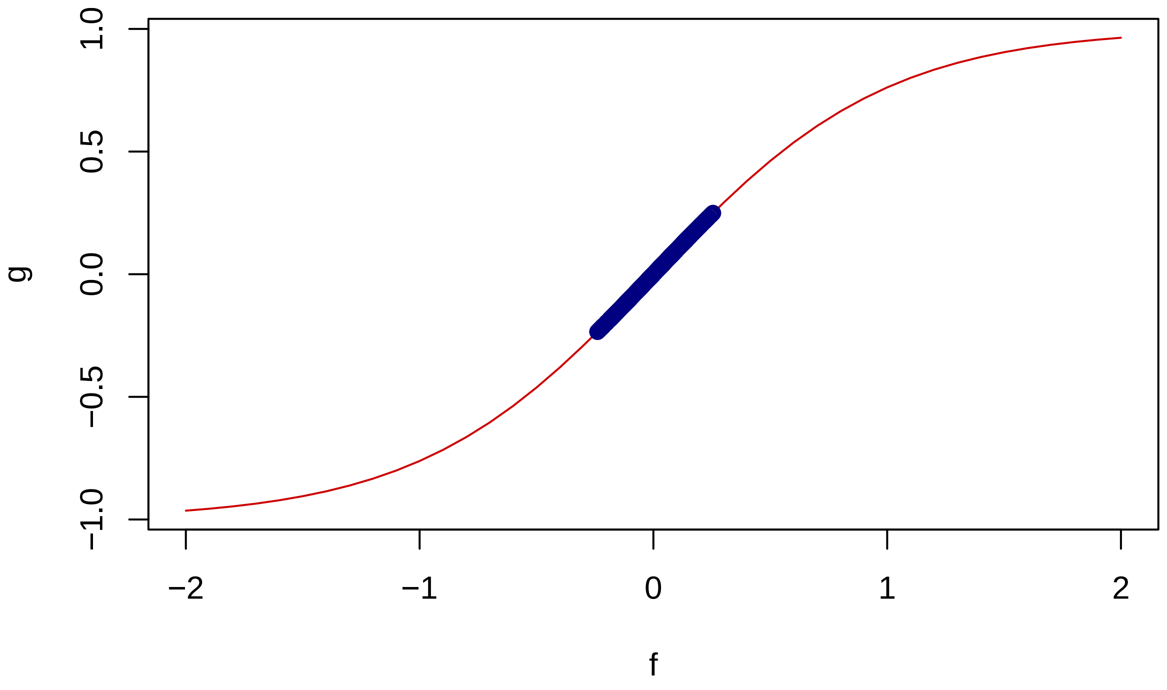
It does NOT make sense to use the log lik on the training data to select among models because the higher the number of components the higher the log lik (since the models are nested). However, using the log lik on some validation makes sense to select among different number of components, since it measures the generalization ability of the models on some previously unseen data. This implicitly penalizes models with many components because they overfit to the training data. The previous results confirm this.

Neural Network

```
library(neuralnet)

set.seed(1234567890)
x1 <- runif(1000, -1, 1)
x2 <- runif(1000, -1, 1)
tr <- data.frame(x1,x2, y=x1 + x2)

nn<-neuralnet(formula = y ~ x1 + x2, data = tr, hidden = c(1), act.fct = "tanh")

plot(nn)
```

$Y = x_1 + x_2$ implies that $y = 7.75(\frac{x1}{7.75} + \frac{x2}{7.75}) = 7.75(0.13x1 + 0.13x2)$. Therefore, by choosing the weights as the NN does, the hidden unit takes the value resulting from passing $0.13x1 + 0.13x2$ through tanh. However, $0.13x1 + 0.13x2$ is small for the training data at hand and, thus, tanh behaves linearly (run the code below to appreciate this). Therefore, the hidden unit equals $\frac{x1}{7.75} + \frac{x2}{7.75}$ and, now, it only remains to weighten this with 7.75 to produce the output $x1 + x2$.

```
f <- seq(-2,2,.1)
g <- tanh(f)
plot(f,g,type="l", col = "red3")
v <- tanh(0.13*tr$x1+0.13*tr$x2)
points(0.13*tr$x1+0.13*tr$x2,v, col = "navy") # The hidden unit only takes values in the linear part of
```


g
1.0
0.5
0.0
−0.5
−1.0
−2
−1
0
1
2
f