ml_exam

random_sampler

16 January 2019

Assignment 1

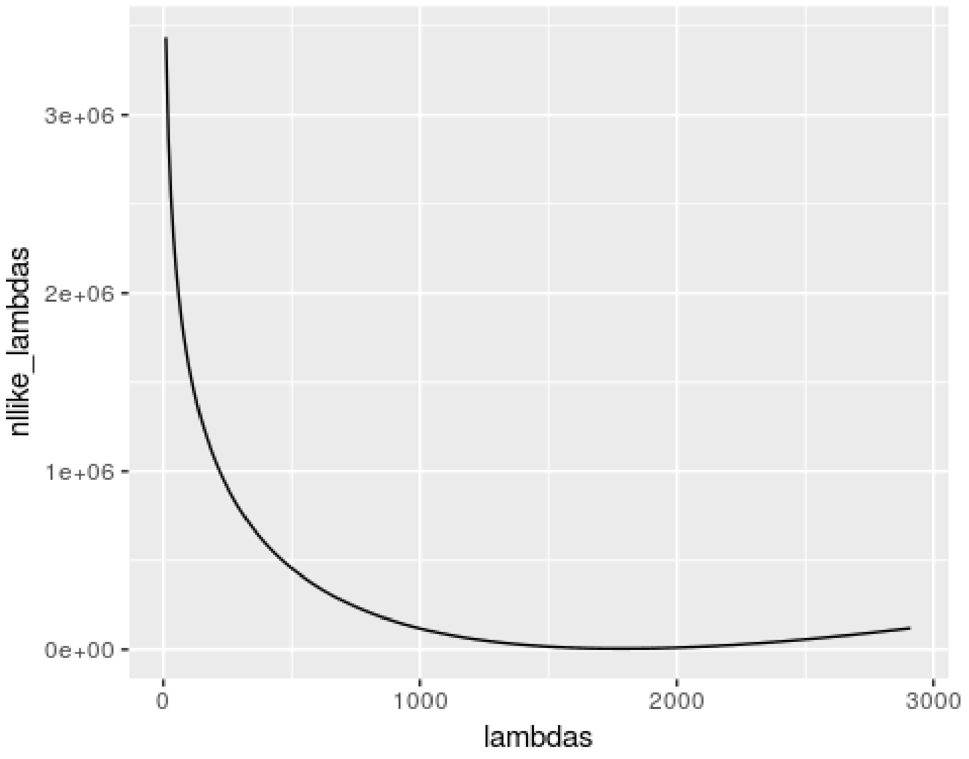
Task 1.1

By visual inspection the best lambda is given by 1750, which is the one that minimizes the negative loglikelihood. If we where to inspect the values, the optimal value, given the sequence of lambdas given on the grid, is 1780, which is pretty close to 1750.

2p

```
# TASK 1.1

# Loading the data set.
data = read.csv("given_files/Influenza.csv")

# Poisson probability function.
prob_y = function(y, lambda)
{
  py = -lambda + (y * log(lambda)) - sum(log(1:y))
  return(py)
}

# Negative log likelihood.
neg_llike = function(Y, lambda)
{
  llike = 0

  for (y in Y)
  {
    llike = llike + prob_y(y, lambda)
  }

  return(- llike)
}


# Plotting the lambda grids.
lambdas = seq(10, 2910, 10)
nllike_lambdas = c()

for (lambda in lambdas)
```

```
{
  nllike_lambdas = c(nllike_lambdas, neg_llike(data$Mortality, lambda))
}

# Plotting the stuff.
p = ggplot2::ggplot() +
    ggplot2::geom_line(ggplot2::aes(x=lambdas, y=nllike_lambdas))

print(p)
```



```
# Selecting the lambda that maximizes the
# loglikelihood or that minimizes the negative
# loglikelihood.
min_id = which.min(nllike_lambdas)
best_lambda = lambdas[min_id]
print("The lambda that maximizes the loglikelihood is:")

## [1] "The lambda that maximizes the loglikelihood is:"

print(best_lambda)

## [1] 1780
```

Task 1.2

The best regularization parameter for the LASSO regression was selected via cross validation. This parameter λ is equal to 630.4167 which is given by the cv.glmnet object. As for the test MSE, it’s value is 630.4167. This value was obtained by training the model with the optimal regularization parameter given the CV procedure and then trained on the train data. With this model the response values were predicted and then the mse was calculated.

The MSE even though it’s still a measure of distance that penalize values that are further distant from the real ones. Implies that the response variable Y , in this case moratlity, by means of optimizing the likelihood of it, follows a normal distribution. In this case, it’s stated that the response variable follows a poisson distribution which contradicts this assumption. There is also the fact, that the response variable is not a continuous variable, but a discrete one. Since one cant have a person who is half dead (decimals), just integers for the response (since it’s a dead body count). Which is another reason of why the MSE might not be the best measurement of performance for this particular model.

2.5p

```
library(glmnet)

## Loading required package: Matrix

## Loading required package: foreach

## Loaded glmnet 2.0-16

# Scaling all the variables as stated in the task.
y = data$Mortality
data_scaled = as.data.frame(scale(data))
data_scaled$Mortality = y

# Dividing the data set into a train and test set.
set.seed(12345)
n = nrow(data_scaled)
id = sample(1:n, floor(n * 0.5))
train = data_scaled[id, ]
test = data_scaled[-id, ]

# Creating the CV for LASSO.
model = cv.glmnet(x=as.matrix(train[, -3]),
               y=train$Mortality,
               alpha=0,
               family="poisson")
```

```
# Getting the penalization parameter.
min_lambda = model$lambda.min
print("Minimum penalization parameter lambda:")

## [1] "Minimum penalization parameter lambda:"

print(min_lambda)

## [1] 630.4167

# Retraining the model with the optimal values.
best_model = glmnet(x=as.matrix(train[, -3]),
                    y=train$Mortality,
                    alpha=0,
                    family="poisson",
                    lambda=min_lambda)

# Calculating the MSE on the test set.
y_hat = predict(best_model, newx=as.matrix(test[, -3]), type="response")

mse = mean((test$Mortality - y_hat) ^ 2)

print("The MSE on the test set is:")

## [1] "The MSE on the test set is:"

print(mse)

## [1] 11621.01
```

The optimal LASSO coefficients are presented below. Since all variables have been scaled, one can compare (ceteris paribus), which variables have the biggest impact on the response variable. The variables in decreasing order, who had the biggest impact are:

- Influenza_lag2
- Temperature.deficit
- Influenza_lag1
- Influenza
- Temp_lag1
- Week
- Temp_lag2
- Year

From these, the ones who had an impact bigger than 0.01 are the top 4. Influenza_lag2, Temperature.deficit, Influenza_lag1 and Influenza. These variables seem to be right, specially if its considered that the reported cases of influenza does not imply a case of mortality in the present but in the future. Thats why the lag variables have so much impact on the model.

```
print(best_model$beta)

## 8 x 1 sparse Matrix of class "dgCMatrix"
##                              s0
## Year               -0.002070813
## Week               -0.008726099
## Influenza           0.012477733
## Temperature.deficit  0.015018005
## Influenza_lag1      0.013025498
## Temp_lag1           0.009459390
## Influenza_lag2      0.015415834
## Temp_lag2           0.007970495
```

As for the intercept of the model α it's value is 7.48177 and the value of $\exp(\alpha)$ is of 1775.393. Which is pretty close to the optimal λ from step 1. These quantities are similar and they should be. It's worth remembering that the parameter λ in a poisson distribuion is an average rate, which in this case, is the average rate of mortality (or the average of the column mortality in the data frame). The mean of mortality in this data is about 1783.765 which is pretty close. It's also needed to remember that at the end, this is a generalized linear model, which implies a linear model that uses the log link function and thus the exponential of α which is the intercept, must resemble the intercept of a linear regresion given OLS. And that this intercept should follow closely the mean value of the response variable.

```
print("Mean of mortality")

## [1] "Mean of mortality"

print(mean(data$Mortality))

## [1] 1783.765

print("Alpha")

## [1] "Alpha"

print(best_model$a0)

##       s0
## 7.481777

print("Exp of alpha")

## [1] "Exp of alpha"

print(exp(best_model$a0))
```

```
##      s0
## 1775.393
```

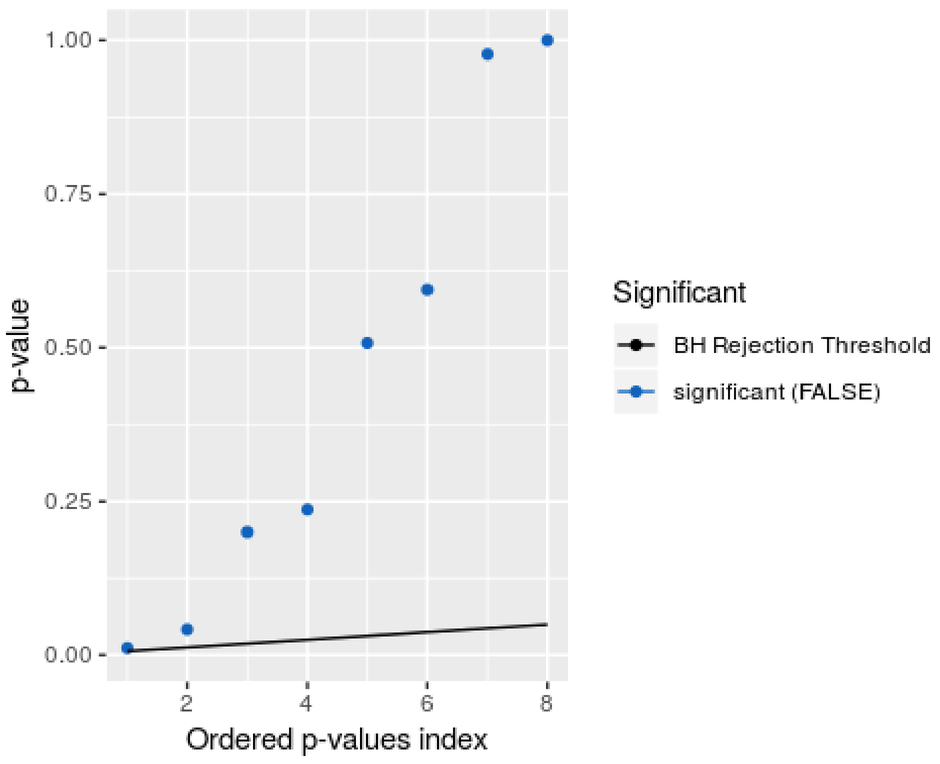
Task 1.3

From the plot and table presented below, it’s clear that there is no sufficient statistical evidence to reject the null hypothesis for any of the tests made. This means that there is no significant differences between the mean of the variables between 1995 and 1996 compared with it self (each test compares the mean of variable X from 1995 to the mean of the same variable on 1996). In this case, it’s no so important to use the BH method since there is almost 12 (11.55) times the number of rows (observations) than the number of columns (features). The BH method is very usefull when dealing with data which have a lot more variables than number of observations, such as genetic data.

1.5p

```
# Creating variable that is going to store the p-values.
p_values = c()

# Use only the data from years 1995 and 1996.
filtered_data = data[data$Year == 1995 | data$Year == 1996, ]
test_data = filtered_data
test_data$Year = NULL
ncols = ncol(test_data)

# Calculating all the p-values given a t-test.
for (i in colnames(test_data))
{
  if (i != "Year")
  {
    test = t.test(test_data[, i] ~ filtered_data[, "Year"])
    p_values = c(p_values, test$p.value)
  }
}

# Ordering the p-values.
ordered_idxs = base::order(p_values)
ordered_pvalues = p_values[ordered_idxs]

# Defining the BJ rejection threshold.
bh_rejection = function(j, alpha=0.15, M=8)
{
  return(alpha * j / M)
}

# Populating the threshold.
fpr = 0.05
threshold = sapply(1:8, bh_rejection, fpr)
```

```
# Merging everything into a dataframe.
results = data.frame(list(index=1:8,
                     ordered_pvalues=ordered_pvalues,
                     threshold=threshold,
                     significant=threshold > ordered_pvalues))

p = ggplot2::ggplot(results) +
   ggplot2::geom_point(ggplot2::aes(x=index, y=ordered_pvalues,
colour="significant (FALSE)")) +
   ggplot2::geom_line(ggplot2::aes(x=index, y=threshold, colour="BH
Rejection Threshold")) +
   ggplot2::scale_colour_manual(values=c("#000000", "#1162bc",
"#f83d69")) +
   ggplot2::labs(x="Ordered p-values index", y="p-value",
colour="Significant")

print(p)
```



```
print(results)

##   index ordered_pvalues threshold significant
## 1     1      0.01152369   0.00625       FALSE
## 2     2      0.04204082   0.01250       FALSE
## 3     3      0.20015678   0.01875       FALSE
## 4     4      0.23698767   0.02500       FALSE
## 5     5      0.50765830   0.03125       FALSE
## 6    6     0.59450045   0.03750      FALSE
## 7    7     0.97732617   0.04375      FALSE
## 8    8     1.00000000   0.05000      FALSE
```

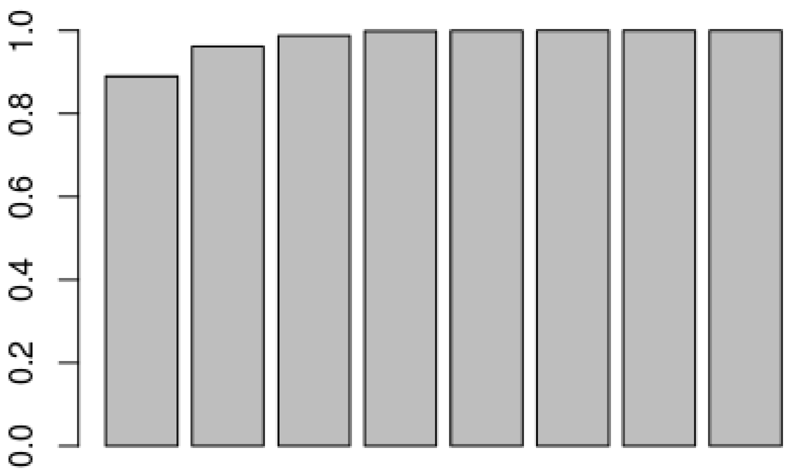
Task 1.4

From the list and plot shown below, we can see that the first two principal components account for more than the 90% of the variation, exactly, it accounts for 97.72% of the total variation of the data.

A plot showing the dependance of the cross-validation error on log(λ) is provided below as well. The complexity of the model does not increase when λ increases but it also do not decrease. We can see that the minimum of non zero variables for each λ is 8, which means that all features were selected for each of the λ parameters. Thus, the complexity of the model nor increases nor decreases, it stays the same.

1p

```
# Creating the dataset for PCA.
pca_data = data
pca_data$Mortality = NULL

# Doing pca stuff.
pca = prcomp(pca_data)

# Showing 90% of the variation.
variation = cumsum(pca$sdev^2) / sum(pca$sdev^2)
print("Cumulative percentage of the variation explained per new latent
variable")

## [1] "Cumulative percentage of the variation explained per new latent
variable"

print(variation)

## [1] 0.8897091 0.9621606 0.9883643 0.9986085 0.9995397 0.9998217
0.9999363
## [8] 1.0000000

barplot(variation)
```



```
# Doing the LASSO CV stuff.
model = cv.glmnet(x=pca$x,
                  y=data$Mortality,
                  alpha=0,
                  family="poisson",
                  lambda=seq(0, 50, 0.1))

print("Minimum number of features selected on the cross validation
procedure")

## [1] "Minimum number of features selected on the cross validation
procedure"

print(min(model$nzero))

## [1] 8
```

The probability model for the LASSO model is presented below:

$$Mortality \sim Poisson(\lambda = \exp(X\beta))$$

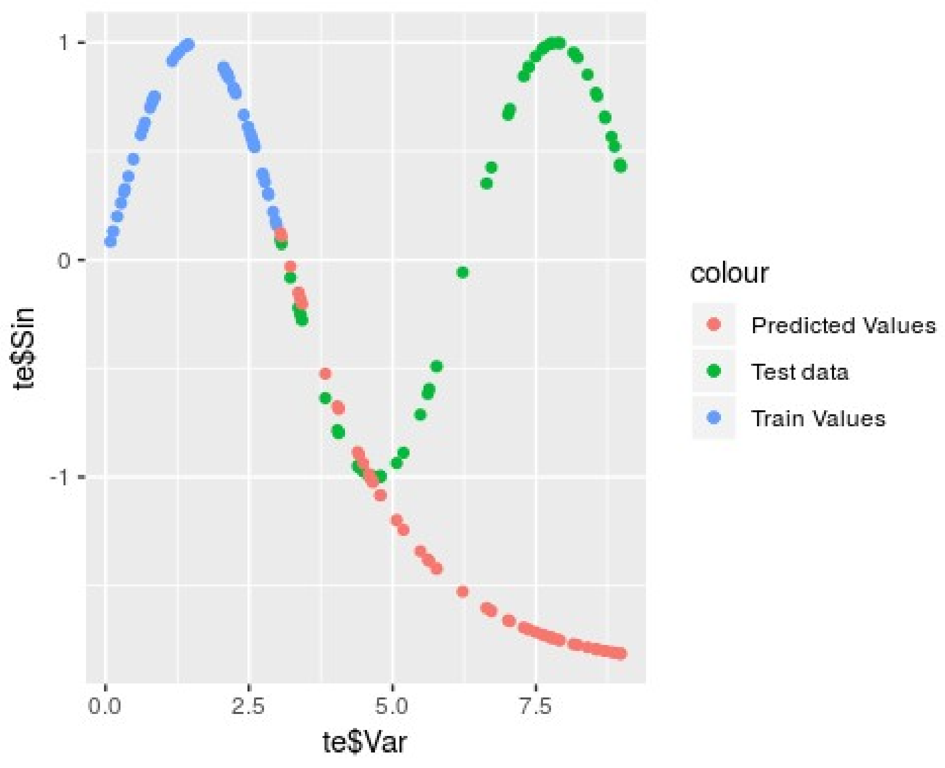
Where X are the matrix that contains all the variables and β are the regression coefficients. # Assignment 2

Neural Networks

Task 2.1

The code and the plot are shown below.

```r
library(neuralnet)

# Doing as told by the exam instructions.
set.seed(1234567890)

# Never use <- please :).
Var = runif(50, 0, 3)
tr = data.frame(Var, Sin=sin(Var))
Var = runif(50, 3, 9)
te = data.frame(Var, Sin=sin(Var))

# Random initialization of the weights in the interval [-1, 1].
winit = runif(10, -1, 1)

# Training the neural network on the train set.
nn = neuralnet(Sin ~ Var, data=tr, hidden=c(3), startweights=winit)

# Getting the predictions.
yhat_test = compute(nn, te$Var)$net.result

# Plotting the plotty mc plot face.
p = ggplot2::ggplot() +
    ggplot2::geom_point(ggplot2::aes(x=te$Var, y=te$Sin, color="Test data"))
+
    ggplot2::geom_point(ggplot2::aes(x=tr$Var, y=tr$Sin, color="Train
Values")) +
    ggplot2::geom_point(ggplot2::aes(x=te$Var, y=yhat_test, color="Predicted
Values"))
print(p)
```



Task 2.2

The weights of the neural network are presented below. The fact that the weights that conforms the output neuron are all negatives and the fact that it grows towards two, means that the output of the hidden layer is being saturated. This means, that the default activation function, which is a sigmoid, is being driven to only ouput a value of 1 thanks to the fact that X is increasing and that the maximum of that function is 1. This means that by multiplying one by each of the output neuron weights, at the end, we are getting a prediction that is no more than the sum of the weights. To be more precise, the asymptotical value at which the neural network tends as X increases is -2.156195796 which is the sum of the weights of the output neuron.

```
print(nn$startweights)

## [[1]]
## [[1]][[1]]
##               [,1]         [,2]          [,3]
## [1,] 0.46286520641 0.2440019781 -0.4238076480
## [2,] 0.08593486762 0.4512787731 -0.6915816143
##
## [[1]][[2]]
##               [,1]
## [1,] -0.7623883728
## [2,] -0.6410439857
## [3,] -0.5902480748
## [4,] -0.1625153627
```

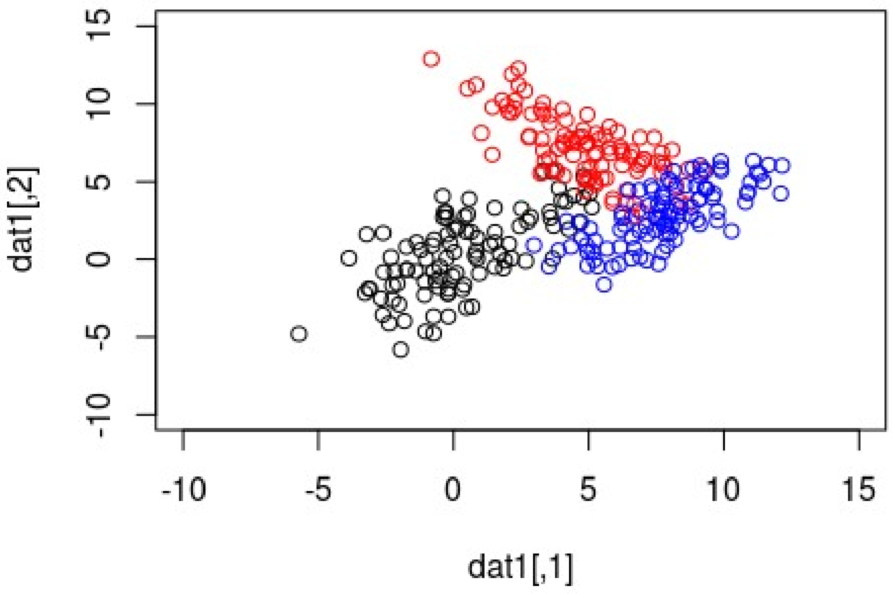
Mixture Models

Task 2.3

The implementation of the EM algorithm can be seen below.

```
library(mvtnorm)

# Doing as told by the exam instructions.
set.seed(1234567890)

# Max number of EM iterations.
max_it = 100
# min change in log likelihood between two consecutive EM iterations.
min_change = 0.1

# Number of training points.
N = 300

# Number of dimensions.
D = 2

# Training data.
x = matrix(nrow=N, ncol=D)

# Sampling the training data.
# First component.
mu1 = c(0, 0)
Sigma1 = matrix(c(5,3,3,5), D, D)
dat1 = rmvnorm(n=100, mu1, Sigma1)

# Second component.
mu2 = c(5, 7)
Sigma2 = matrix(c(5,-3,-3,5), D, D)
dat2 = rmvnorm(n=100, mu2, Sigma2)

# Third component.
mu3 = c(8, 3)
Sigma3 = matrix(c(3,2,2,3), D, D)
dat3 = rmvnorm(n=100, mu3, Sigma3)

# Populating the data set.
x[1:100, ] = dat1
x[101:200, ] = dat2
```

```
x[201:300, ] = dat3

true_pi = c(1/3, 1/3, 1/3)

# Plot of the data.
plot(dat1,xlim=c(-10,15),ylim=c(-10,15))
points(dat2,col="red")
points(dat3,col="blue")
```



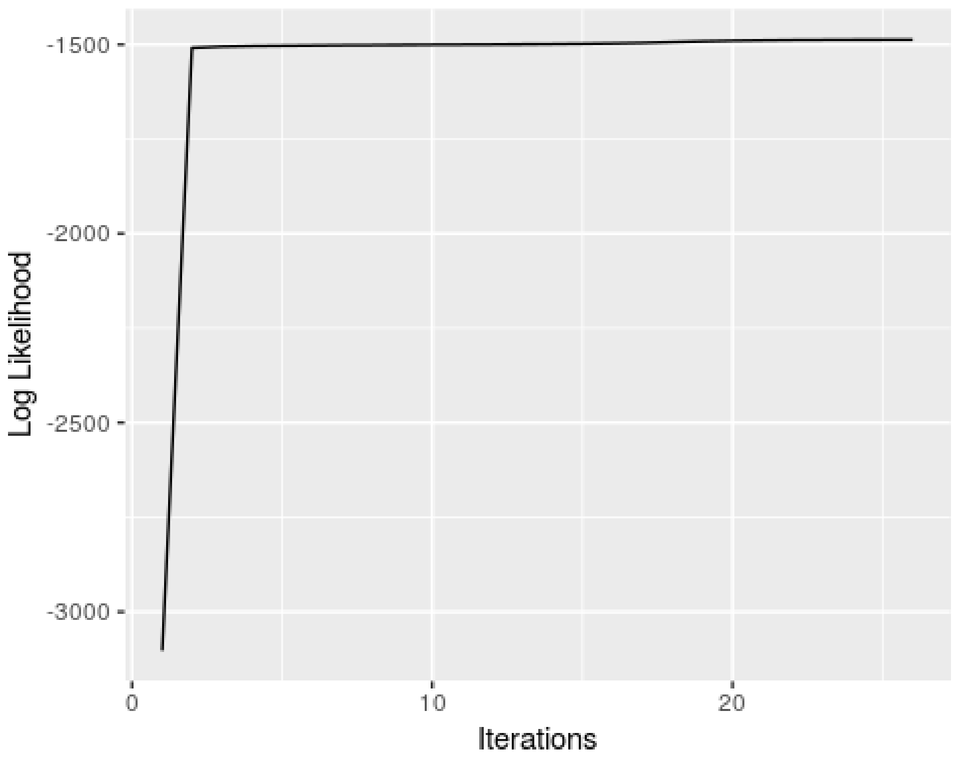
```
# Initialization of the parameters of the model.
# Number of guessed components.
K = 3

# Fractional component assignments. AKA latent variable.
z = matrix(nrow=N, ncol=K)

# Mixing coefficients.
pi = vector(length = K)

# Conditional means.
mu = matrix(nrow=K, ncol=D)

# Conditional variances.
Sigma = array(dim=c(D, D, K))

# Variable that stores the log likelihood.
```

```
llik = c()

# 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)
}

get_me_the_F = function(x, mu, Sigma)
{
  mvn_f = matrix(nrow=nrow(x), ncol=3)

  for (i in 1:nrow(x))
  {
    for (j in 1:3)
    {
      mvn_f[i, j] = dmvnorm(x[i, ], mu[j, ], Sigma[ , , j])
    }
  }

  return(mvn_f)
}

#######################################
# IMPLEMENTATION OF THE EM ALGORITHM #
#######################################

for (i in 1:max_it)
{
  ##########
  # E STEP #
  ##########

  # Multivariate normal density calculation.
  mvn_f = get_me_the_F(x, mu, Sigma)

  # Calculating the posterior.
  numerator_pos = mvn_f * matrix(pi, nrow=nrow(x), ncol=3, byrow=TRUE)
  denominator_pos = rowSums(numerator_pos)
  posterior = numerator_pos / denominator_pos

  ###############################
  # LOOKING IF WE NEED TO STOP #
  ###############################
  llik_i = sum(log(denominator_pos))
  llik = c(llik, llik_i)
  if (i >= 2)
  {
    if (llik[i] - llik[i - 1] < min_change)
    {
      break
    }
  }

  #########
  # M STEP#
  #########

  # Updating pi.
  numerator_pi = colSums(posterior)
  pi = numerator_pi / N

  # Updating mu.
  mu = (t(posterior) %*% x) / numerator_pi

  # Updating each sigma.
  for (k in 1:K)
  {
    shifted_x =(x - matrix(mu[k, ], nrow=N, ncol=2, byrow=TRUE))
    Sigma[ , , k] = (t(shifted_x) %*% (shifted_x * posterior[, k])) /
numerator_pi[k]
  }


}
```

Task 2.4

Below we can see that the likelihood increases with the number of iterations. We can also see that the parameters are some what close, even though they are not perfectly the same. This might be due to randomness and local optima. Although its pretty clear that the parameters tend to have the same values as the true variables.

```
p = ggplot2::ggplot() +
    ggplot2::geom_line(ggplot2::aes(x=1:length(llik), y=llik)) +
    ggplot2::labs(x="Iterations", y="Log Likelihood")

print(p)
```



```r
# Showing that the true values are close to the estimated.
print("TRUE PI VALUES")

## [1] "TRUE PI VALUES"

print(c(1/3, 1/3, 1/3))

## [1] 0.3333333333 0.3333333333 0.3333333333

print("ESTIMATED PI VALUES")

## [1] "ESTIMATED PI VALUES"

print(pi)

## [1] 0.2993550974 0.3616208618 0.3390240407

print("TRUE MU VALUES")

## [1] "TRUE MU VALUES"

print(mu1)

## [1] 0 0

print(mu2)

## [1] 5 7

print(mu3)
```

```
## [1] 8 3

print("ESTIMATED MU VALUES")

## [1] "ESTIMATED MU VALUES"

print(mu)

##              [,1]          [,2]
## [1,] -0.1096711342 -0.02341510203
## [2,]  4.9736527626  6.89649646076
## [3,]  7.5481551652  2.87162986843

print("TRUE SIGMA VALUES")

## [1] "TRUE SIGMA VALUES"

print(Sigma1)

##      [,1] [,2]
## [1,]    5    3
## [2,]    3    5

print(Sigma2)

##      [,1] [,2]
## [1,]    5   -3
## [2,]   -3    5

print(Sigma3)

##      [,1] [,2]
## [1,]    3    2
## [2,]    2    3

print("ESTIMATED SIGMA VALUEs")

## [1] "ESTIMATED SIGMA VALUEs"

print(Sigma)

## , , 1
##
##             [,1]        [,2]
## [1,] 3.940797379 2.706444705
## [2,] 2.706444705 5.796523323
##
## , , 2
##
##              [,1]         [,2]
## [1,]  4.513016518 -3.317680382
## [2,] -3.317680382  5.154162760
##
## , , 3
```

```
##
##             [,1]        [,2]
## [1,] 4.638159570 2.828175975
## [2,] 2.828175975 3.861885534
```