January 2022

Assignment 1

Reading Data

```
data <- read.csv("olive.csv")
```

Task 1

```
# scaled data for fatty acid variables
df1 <- data.frame(scale(data[, -c(1:3)]))

# calculating the covariance matrix
cov_matrix <- data.frame(cov(df1))

# calculating eigenvalues
eigenval <- eigen(cov_matrix)

counter <- which(cumsum(eigenval$values/sum(eigenval$values) * 100) >=
    90)

prob_var <- as.numeric(sprintf("%2.3f", eigenval$values/sum(eigenval$values) *
    100))
variation <- sum(prob_var[1:4])

paste("The first", counter[1], "principal components explain", variation,
    " of the variation in the data.")
```

```
## [1] "The first 4 principal components explain 91.205  of the variation in the data."
```

Scaling the data allows the PCA to explain the relevant variation in the data, as otherwise it would overestimate the importance of features with a high variance.

Task 2

```
# implementing PCA
pca <- princomp(df1)

# get a visual of the scores of all principal component scores.
components <- data.frame(pca$scores)

# get the first three principal components scores
three_components <- data.frame(components[, c(1:3)])

df2 <- cbind(data$Region, three_components)
colnames(df2) <- c("Region", "Comp1", "Comp2", "Comp3")
```

```
n = dim(df2)[1]
set.seed(12345)
id = sample(1:n, floor(n * 0.5))
# 50% train data
train2 = df2[id, ]
# 50% test data
test2 = df2[-id, ]
```

```
library("nnet")
# implementing the logistic regression model
model1 <- multinom(Region ~ ., data = train2)
```

```
## # weights:  15 (8 variable)
## initial  value 314.203115
## iter  10 value 88.100326
## iter  20 value 52.996752
## final  value 52.987997
## converged
```

```
misclass <- function(actual_val, predicted_val) {
    conf_mat <- table(actual_val, predicted_val)
    n <- length(actual_val)
    error <- 1 - sum((diag(conf_mat))/n)
    return(error)
}

train2_predictions <- predict(model1, train2)
test2_predictions <- predict(model1, test2)

train2_error <- misclass(train2_predictions, train2$Region)
test2_error <- misclass(test2_predictions, test2$Region)

df_error <- data.frame(`Train error` = train2_error, `Test error` = test2_error)
row.names(df_error) <- c("Misclassification Rates")
knitr::kable(df_error)
```

	Train.error	Test.error
Misclassification Rates	0.0594406	0.0734266

The training misclassification rate is 0.05944056 and the test misclassification rate is 0.07342657. So for the test data only about 7% of data is misclassifide which is a relatively low value, considering that only the first three principal components were used as features in the model.

```
print(model1)
```

```
## Call:
## multinom(formula = Region ~ ., data = train2)
##
## Coefficients:
##   (Intercept)     Comp1    Comp2     Comp3
## 2   -3.218617 -2.992163 3.187718 -4.000276
## 3   -5.948747 -5.677377 2.827254 -3.045757
##
## Residual Deviance: 105.976
## AIC: 121.976
```

The decision boundary of the first class equals:

$$-3.218617 - 2.992163Comp1 + 3.187718Comp2 - 4.000276Comp3$$

The decision boundary of the second class equals:

$$-5.948747 - 5.677377Comp1 + 2.827254Comp2 - 3.045757Comp3$$

The decision boundary of the third class equals:

$$-2.73013 - 2.685214Comp1 - 0.360464Comp2 + 0.954519Comp3$$

Task 3

```
# creating target value
regions <- data.frame(ifelse(data$Region == 1, "South", "Other"))
colnames(regions) <- c("Region")

# preparing data
df3 <- cbind(regions, data[, -c(1, 2, 3)])
df3$Region <- as.factor(df3$Region)

n <- dim(df3)
set.seed(12345)
id3 <- sample(1:n, floor(n * 0.5))
```

```
## Warning in 1:n: numerical expression has 2 elements: only the first used
```

```
# 50% train data
train3 <- df3[id3, ]
# 50% test data
test3 <- df3[-id3, ]
```

```
library("glmnet")
```

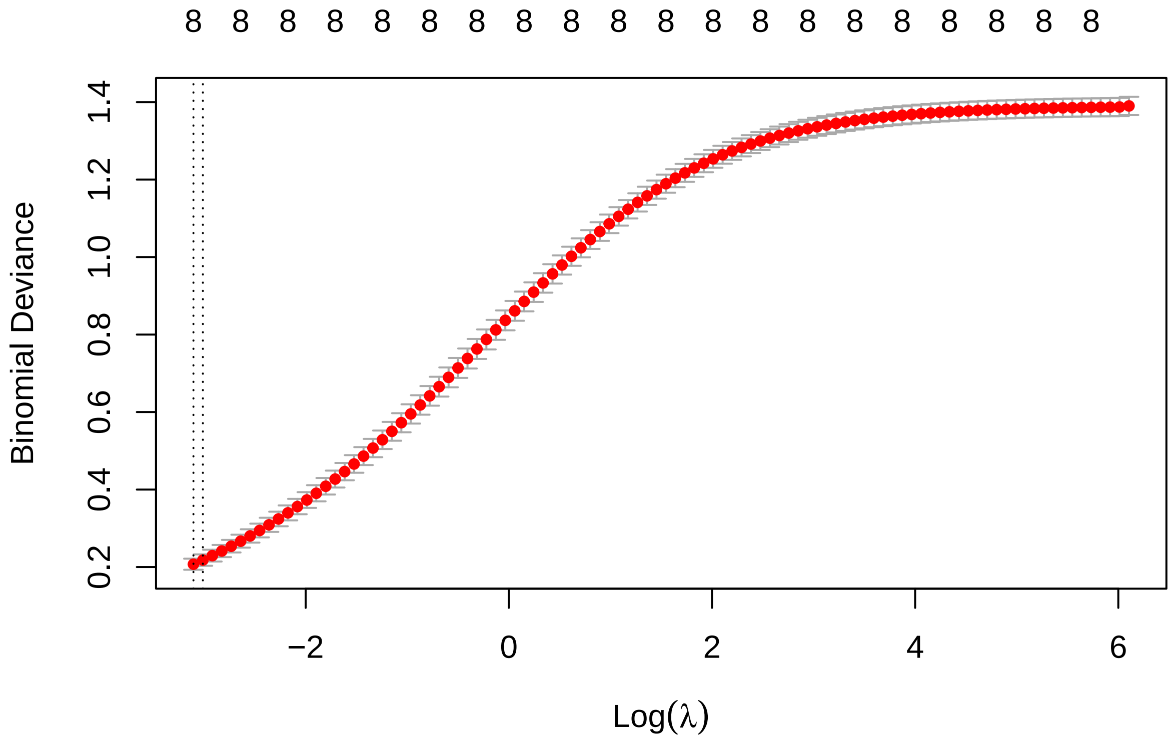
```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-3
```

```
Y <- train3$Region
X <- train3[, -c(1)]

# implementing ridge regression model
ridge <- glmnet(as.matrix(X), as.matrix(Y), alpha = 0, family = "binomial")

# cross validation
cv_ridge <- cv.glmnet(as.matrix(X), as.matrix(Y), alpha = 0, family = "binomial")

cat("The optimal penalty factor value is", cv_ridge$lambda.min)
```

```
## The optimal penalty factor value is 0.04481765
```

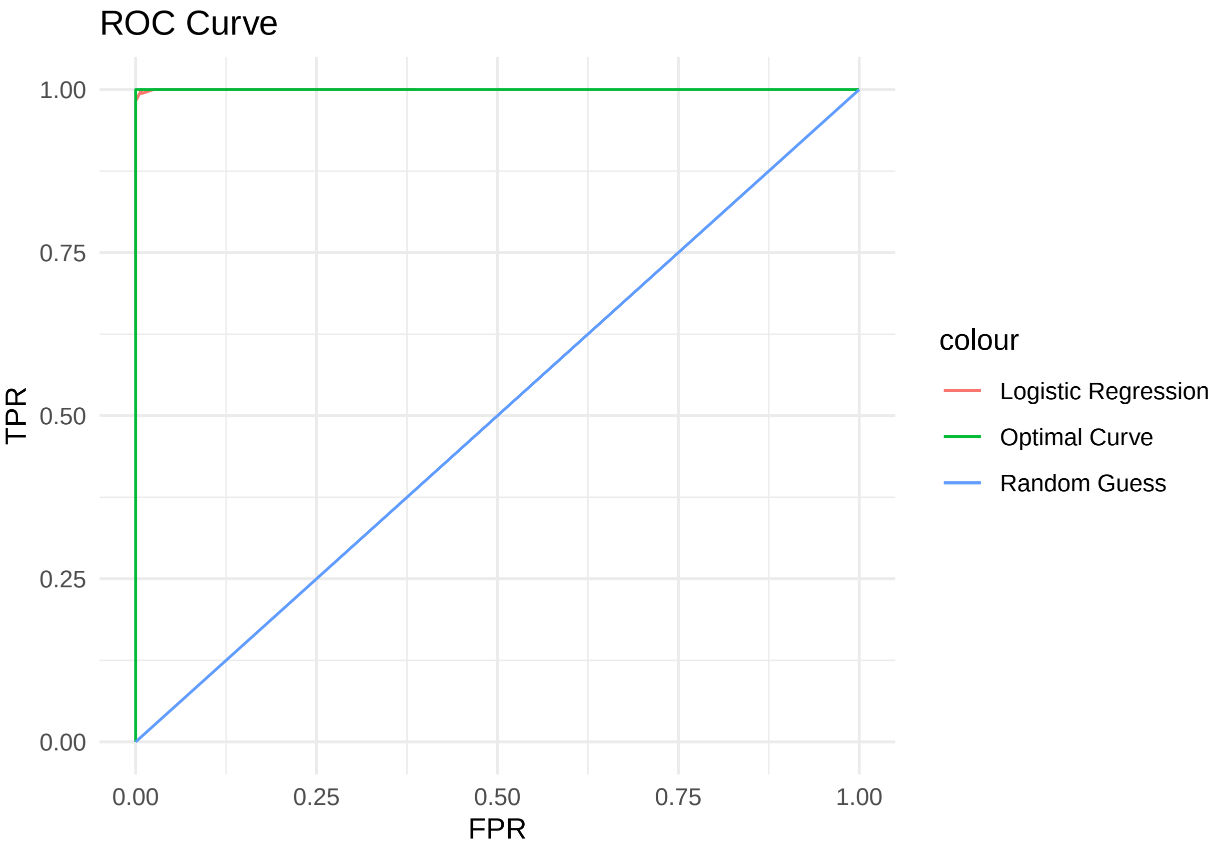
```
plot(cv_ridge)
```



From the plot one can see that the model seems to start getting bad results very quickly for increasing values of lambda. Therefore, it can be concluded that the unpenalised model has a good trade-off between bias and variance, because introducing a high penalty factor lambda would decrease the variance and increase the bias and the error would therefore be less for these high values of lambda, if the variance of the unpenalised model was too high (which is called overfit).

```
TPR <- c()
FPR <- c()

# penalised predictions on test data
test_predict <- predict(ridge, as.matrix(subset(test3, select = -Region)),
    type = "response", s = cv_ridge$lambda.min)

# calculating TPR & FPR
for (i in seq(0.05, 0.95, 0.05)) {
    p <- as.factor(ifelse(test_predict > i, "South", "Other"))
    cm <- table(test3$Region, p)
    TPR <- append(TPR, cm[2, 2]/(cm[1, 2] + cm[2, 2]))
    FPR <- append(FPR, cm[2, 1]/(cm[1, 1] + cm[2, 1]))
}
```

```
plot_data <- data.frame(TPR = TPR, FPR = FPR)

library("ggplot2")

ggplot() + geom_line(data = plot_data, mapping = aes(FPR, TPR, color = "Logistic Regression")) +
    geom_line(data = data.frame(FPR = c(0, 0, 1), TPR = c(0, 1, 1)), mapping = aes(FPR,
        TPR, color = "Optimal Curve")) + geom_line(data = data.frame(FPR = c(0,
    1), TPR = c(0, 1)), mapping = aes(FPR, TPR, color = "Random Guess")) +
    theme_minimal() + labs(title = "ROC Curve")
```



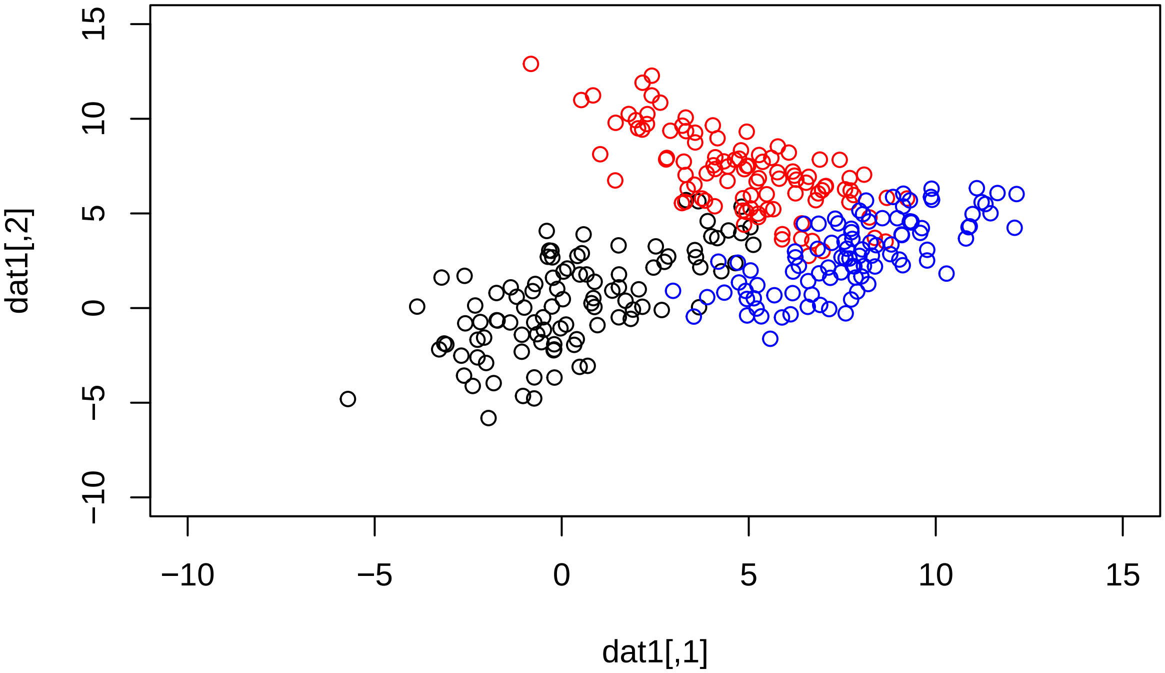
From the plot one can see that the Logistic Regression performs very well on the test data. The area under the curve is almost 1 which is the optimal value a model could theoretically get. A model that would only output random guesses of the two classes for the prediction would have a value for the area under the curve of 0.5, so the proposed model performs way better. In this case, the ROC curve with true positive rate (TPR) and false positive rate (FPR) was used, because the two target classes are appear about equally in the data (otherwise another metric than FPR and TPR would be more insightful).

Assignment 2

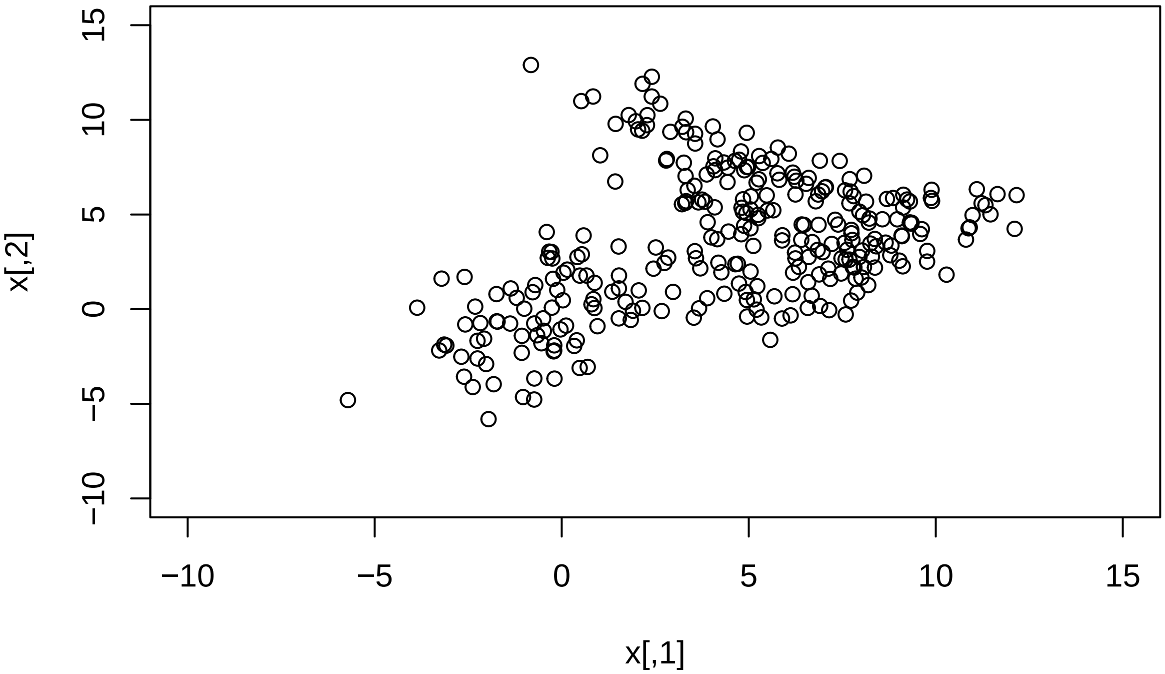
Gaussian Mixture Models (Semi-Supervised GMM)

```
# install.packages('mvtnorm')
library(mvtnorm)
set.seed(1234567890)
min_change <- 0.001  # min parameter change between two consecutive 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))
```



```r
K = 3  # number of classes
w <- matrix(nrow = N, ncol = K)  # fractional class assignments
pi <- vector(length = K)  # mixing coefficients
mu <- matrix(nrow = K, ncol = D)  # class conditional means
Sigma <- array(dim = c(D, D, K))  # class conditional covariances
```

```r
# Calculate pi, mu, and Sigma according to 10.3 page 205 (assuming
# the we know the true class of the first ten components of each
# class)

pi <- c(10, 10, 10)  # We know the first 10 components of each class
pi <- pi/sum(pi)  # to get percentages

pi
```

```
## [1] 0.3333333 0.3333333 0.3333333
```

```r
# Mean of the known components for each class
mu[1, ] <- colMeans(x[1:10, ])
mu[2, ] <- colMeans(x[101:110, ])
mu[3, ] <- colMeans(x[201:210, ])

mu
```

```
##           [,1]     [,2]
## [1,] 0.8053535 0.860496
## [2,] 4.1857881 7.206610
## [3,] 7.6752711 2.489227
```

```r
# Variance of the known components for each class

Sigma[, , 1] <- var(x[1:10, ])  #which is the same as (t(x[1:10,] - mu[1,]) %*% (x[1:10,] - mu[1,])) /
Sigma[, , 2] <- var(x[101:110, ])
Sigma[, , 3] <- var(x[201:210, ])
```

```
# set max_it to some value
max_it <- 10000

# Create vector to store log-likelihoods which is needed to compare
# the minimum change in log-likelihoods.
llik <- rep(NA, length = max_it)

for (it in 1:max_it) {

    # E-step: Computation of the fractional component
    llik[it] <- 0

    for (n in 1:N) {

        # According to the instructions, the algorithm is not run for
        # all value of N data points but only for the unkown ones.
        # This does not make a difference though, because in the
        # implementation below every known data point is set to its
        # original known value (i.e. 1 for the correct class and 0
        # for the incorrect ones). In other words, these iterations
        # do not make a difference and it is the same as if they were
        # set only one # initial time.

        for (k in 1:K) {
            w[n, k] <- pi[k] * dmvnorm(x[n, ], mu[k, ], Sigma[, , k])
        }

        # If data point is known, set to 1 for correct class and 0
        # for all others.
        if (n %in% 1:10) {
            w[n, 1] <- 1
            w[n, 2] <- 0
            w[n, 3] <- 0
        }

        if (n %in% 101:110) {
            w[n, 1] <- 0
            w[n, 2] <- 1
            w[n, 3] <- 0
        }

        if (n %in% 201:210) {
            w[n, 1] <- 0
            w[n, 2] <- 0
            w[n, 3] <- 1
        }

        # Log likelihood computation according to (10.2b)
        llik[it] <- llik[it] + log(sum(w[n, ]))

        # does not change value of known data points because they sum
        # up to one by definition
        w[n, ] <- w[n, ]/sum(w[n, ])
    }
    cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
    flush.console()

    # Stop if the log 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 each class calculate pi according to
    # (10.10b)
    for (k in 1:K) {
        pi[k] <- sum(w[, k])/N
        # for each dimension calculate
        for (d in 1:D) {
            # mu according to (10.10c)
            mu[k, d] <- sum(x[, d] * w[, k])/sum(w[, k])
        }
        # for each dimension calculate
        for (d in 1:D) {
            # for each dimension calculate
            for (d2 in 1:D) {
                # according to (10.10d)
                Sigma[d, d2, k] <- sum((x[, d] - mu[k, d]) * (x[, d2] -
                  mu[k, d2]) * w[, k])/sum(w[, k])
            }
        }
    }
}
```

```
## iteration:  1 log likelihood:  -1376.92
## iteration:  2 log likelihood:  -1346.474
## iteration:  3 log likelihood:  -1342.466
## iteration:  4 log likelihood:  -1341.037
## iteration:  5 log likelihood:  -1340.37
## iteration:  6 log likelihood:  -1339.997
## iteration:  7 log likelihood:  -1339.761
## iteration:  8 log likelihood:  -1339.602
## iteration:  9 log likelihood:  -1339.488
## iteration:  10 log likelihood:  -1339.405
## iteration:  11 log likelihood:  -1339.343
## iteration:  12 log likelihood:  -1339.296
## iteration:  13 log likelihood:  -1339.26
## iteration:  14 log likelihood:  -1339.232
## iteration:  15 log likelihood:  -1339.21
## iteration:  16 log likelihood:  -1339.194
## iteration:  17 log likelihood:  -1339.18
## iteration:  18 log likelihood:  -1339.17
## iteration:  19 log likelihood:  -1339.162
## iteration:  20 log likelihood:  -1339.155
## iteration:  21 log likelihood:  -1339.15
## iteration:  22 log likelihood:  -1339.146
## iteration:  23 log likelihood:  -1339.143
## iteration:  24 log likelihood:  -1339.14
## iteration:  25 log likelihood:  -1339.138
## iteration:  26 log likelihood:  -1339.136
## iteration:  27 log likelihood:  -1339.135
## iteration:  28 log likelihood:  -1339.134
## iteration:  29 log likelihood:  -1339.133
```

```
print("Actual mu values")
```

```
## [1] "Actual mu values"
```

```
matrix(c(mu1, mu2, mu3), nrow = 3, ncol = 2, byrow = TRUE)
```

```
##      [,1] [,2]
## [1,]    0    0
## [2,]    5    7
## [3,]    8    3
```

```
print("GMM mu values (rounded to 2 decimal places")
```

```
## [1] "GMM mu values (rounded to 2 decimal places"
```

```
round(mu, 2)
```

```
##      [,1] [,2]
## [1,] 0.26 0.25
## [2,] 4.94 7.02
## [3,] 7.76 2.89
```

```
print("Actual Sigma values")
```

```
## [1] "Actual Sigma values"
```

```
Sigma1
```

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

```
Sigma2
```

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

```
Sigma3
```

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

```
print("GMM Sigma values")
```

```
## [1] "GMM Sigma values"
```

```
Sigma
```

```
## , , 1
##
##          [,1]     [,2]
## [1,] 5.154362 3.637591
## [2,] 3.637591 6.450587
##
## , , 2
##
##           [,1]      [,2]
## [1,]  4.440287 -3.256524
## [2,] -3.256524  4.889810
##
## , , 3
##
##          [,1]     [,2]
## [1,] 4.052096 2.756186
## [2,] 2.756186 3.803429
```

```
print("Actual pi values")
```

```
## [1] "Actual pi values"
```

```
c(1/3, 1/3, 1/3)
```

```
## [1] 0.3333333 0.3333333 0.3333333
```

```
print("GMM Sigma values")
```

```
## [1] "GMM Sigma values"
```

```
pi
```

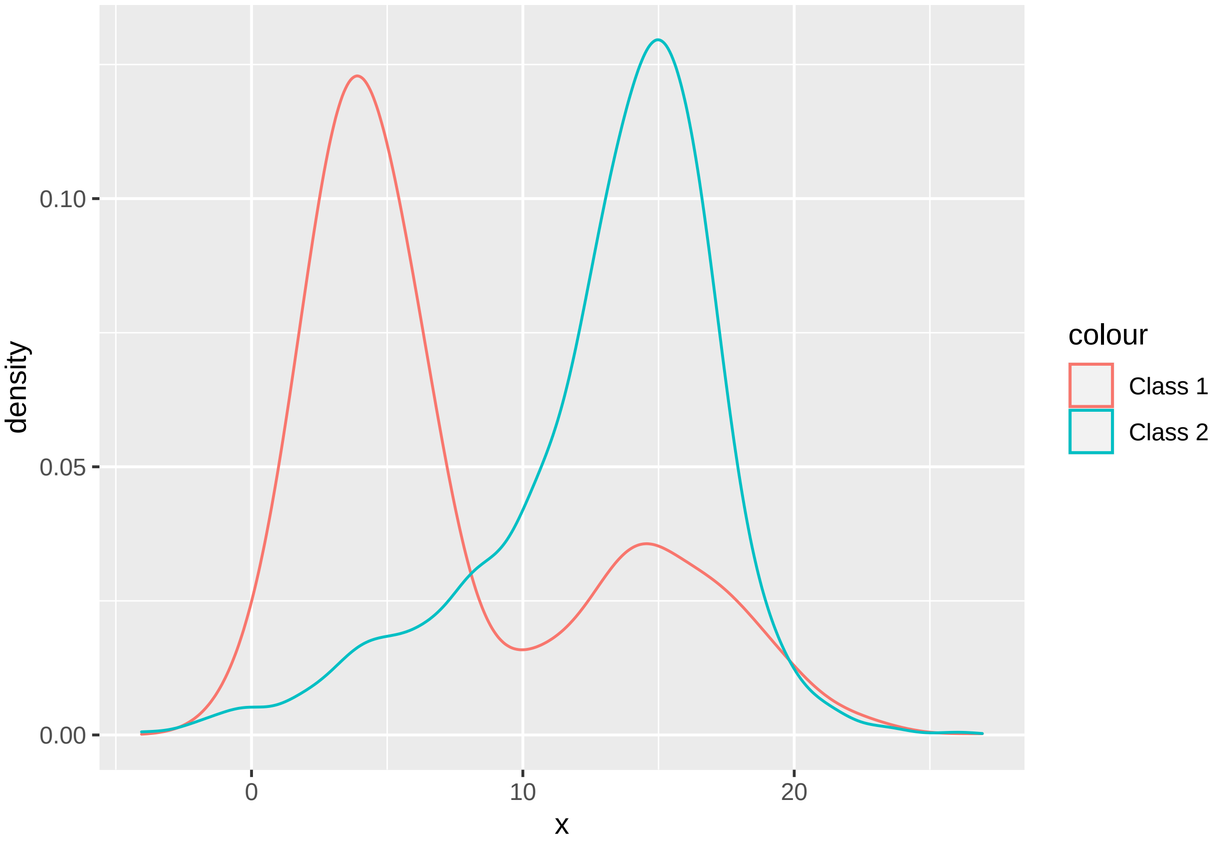
```
## [1] 0.3273498 0.3470995 0.3255508
```

The values of mu, Sigma, and pi are close to the original ones. If the estimated mu values were rounded to a whole number, all of the estimations would perfectly fit the actual mu values. This is also true for most of the Sigma values. The pi values also almost perfectly show the even distribution of the three classes in the sense the actual three class labels belonged to one third of the data points each. One can also see that the log-likelihood strictly increases up until the convergence criterion is reached.

Kernel Methods

```
set.seed(1234567890)

N_class1 <- 1500
N_class2 <- 1000

data_class1 <- NULL
for (i in 1:N_class1) {
    a <- rbinom(n = 1, size = 1, prob = 0.3)
    b <- rnorm(n = 1, mean = 15, sd = 3) * a + (1 - a) * rnorm(n = 1, mean = 4,
        sd = 2)
    data_class1 <- c(data_class1, b)
}

data_class2 <- NULL
for (i in 1:N_class2) {
    a <- rbinom(n = 1, size = 1, prob = 0.4)
    b <- rnorm(n = 1, mean = 10, sd = 5) * a + (1 - a) * rnorm(n = 1, mean = 15,
        sd = 2)
    data_class2 <- c(data_class2, b)
}
```

```
library(ggplot2)

ggplot() + geom_density(data.frame(x = data_class1), mapping = aes(x, color = "Class 1")) +
    geom_density(data.frame(x = data_class2), mapping = aes(x, color = "Class 2"))
```

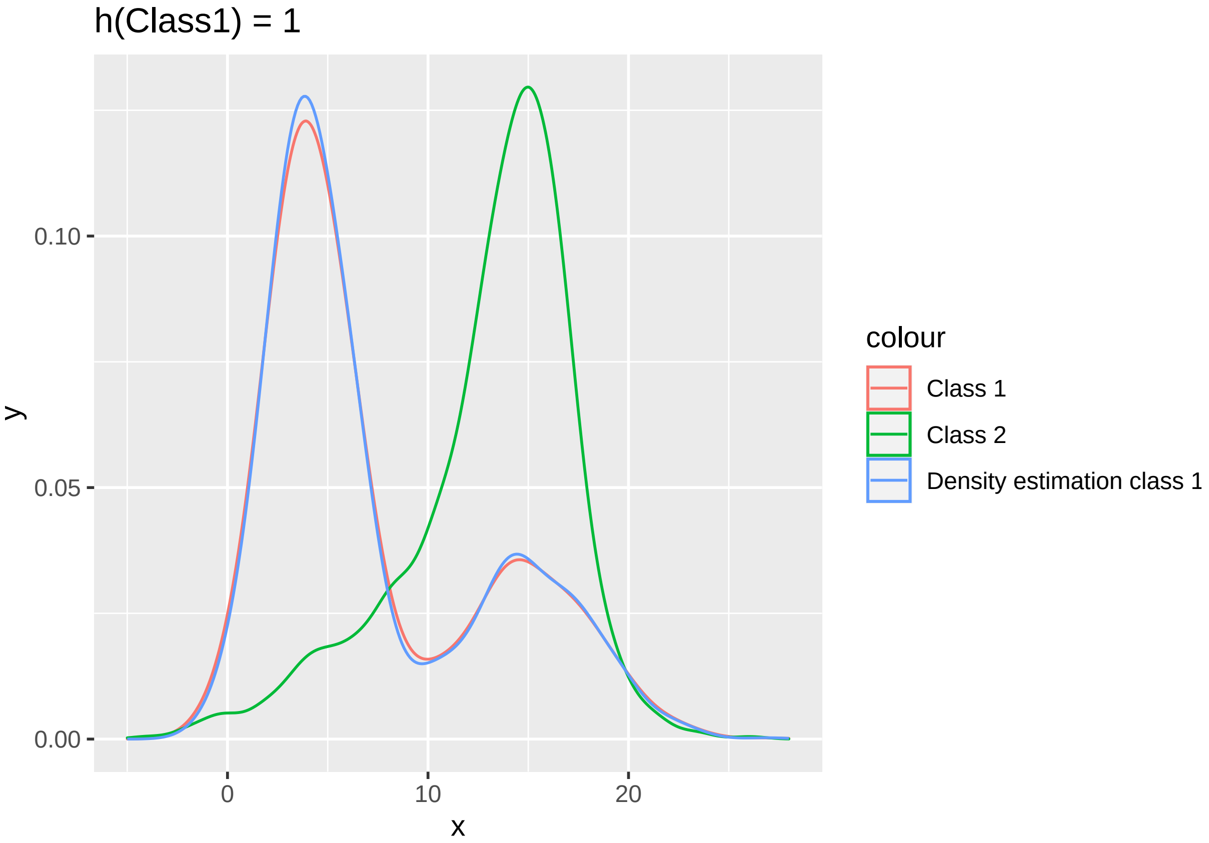


```
create_estimated_density <- function(data, h) {
    # returns the estimated density # function according to the
    # instructions for the given data and a value of h
    res <- function(x) {
        dists <- x - data
        dists <- dists/h  # The larger h, the smaller get the dists values
        return(mean(dnorm(dists)))  # the smaller the dists value, the higher the # value of dnorm(dist
    }
    return(res)
}
```

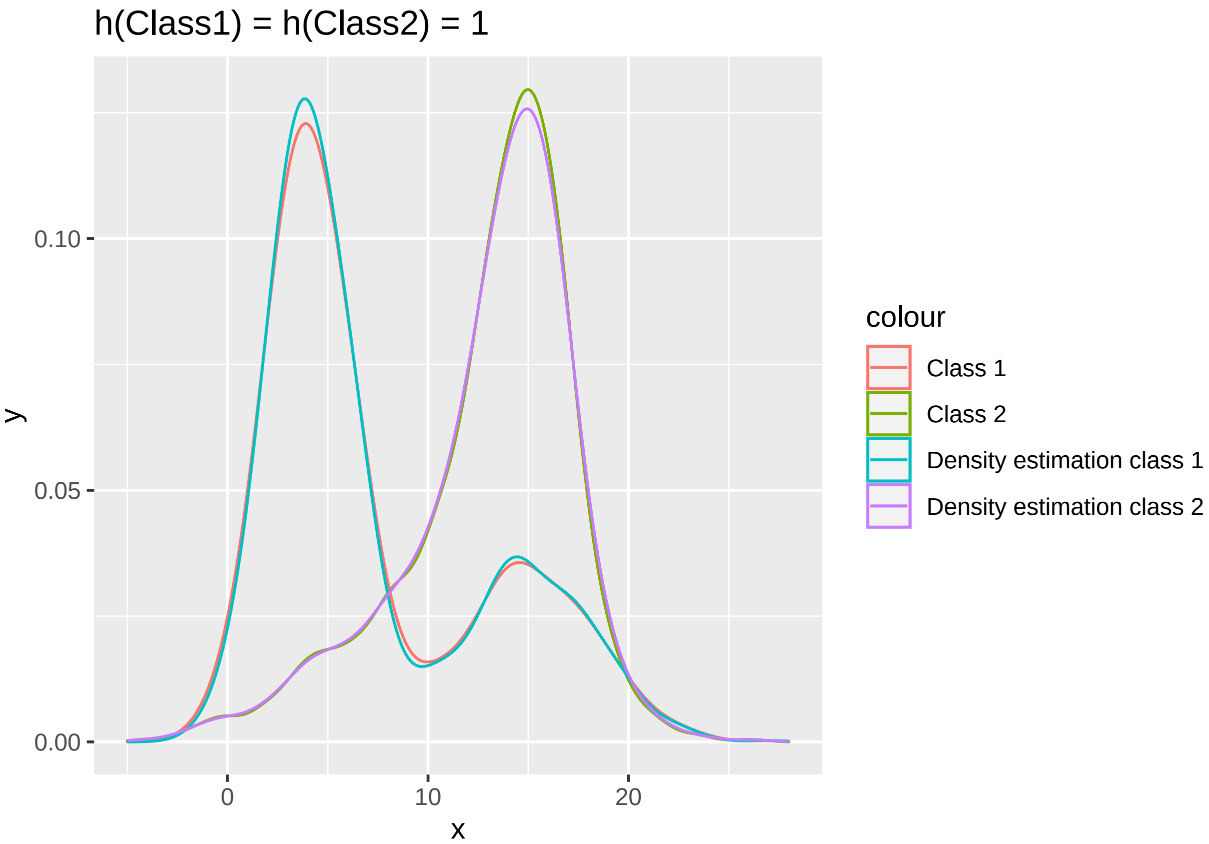
In the plot above one can see the density estimation of the data of the build in geom_density function of the ggplot2 package. It is nice as a comparison, as it is implemented in a way that the integral sums up to one which should be true for densities by definition. This condition is not automatically true for all values of h. If one considers, for example, that h would be infinity, all the kernels would have the value dnorm(0) = 0.3989423 for all input values x^* and therefore the integral over all these values would be infinity, not one. A value of h has to be chosen to make the integral over the estimated densities as close to 1 as possible. The ggplot function geom_density can therefore be helpful.

```
density_est1 <- create_estimated_density(data_class1, 1)
```

```
plot_data1 <- data.frame(x = seq(-5, 28, 0.1), y = sapply(seq(-5, 28, 0.1),
    density_est1))

ggplot() + geom_density(data.frame(x = data_class1), mapping = aes(x, color = "Class 1")) +
    geom_density(data.frame(x = data_class2), mapping = aes(x, color = "Class 2")) +
    geom_line(plot_data1, mapping = aes(x, y, color = "Density estimation class 1")) +
    labs(title = "h(Class1) = 1")
```



```
density_est2 <- create_estimated_density(data_class2, 1)
plot_data2 <- data.frame(x = seq(-5, 28, 0.1), y = sapply(seq(-5, 28, 0.1),
    density_est2))
ggplot() + geom_density(data.frame(x = data_class1), mapping = aes(x, color = "Class 1")) +
    geom_density(data.frame(x = data_class2), mapping = aes(x, color = "Class 2")) +
    geom_line(plot_data1, mapping = aes(x, y, color = "Density estimation class 1")) +
    geom_line(plot_data2, mapping = aes(x, y, color = "Density estimation class 2")) +
    labs(title = "h(Class1) = h(Class2) = 1")
```



One can see that if both values of h are chosen to be 1, the fit to the ggplot density seems to be quite good and the integral over all values of x therefore close to 1. This means that h = 1 is a good value for both data_class1 and data_class2.

To get the probability of a class (e.g. class 1) given a value of x^* on could use Bayes Theorem which states that $P(Class = 1|x^*) = \frac{P(x^*|Class=1)P(Class=1)}{P(x^*)}$ where $P(x^*|Class = 1)$ is the function derived above, $P(Class = 1)$ is the probability of any point being in class one (so it is the proportion of data points falling in class 1, here 1500/(1000+1500) = 0.6 which is from now on called weight of class 1 or in general wi for class i) and $P(x^*)$ is the weighted sum of the functions $P(x^*|Class = i)$ so $P(x^*) = w_1 P(x^*|Class = 1) + w_2 P(x^*|Class = 2)$. In total this results in $P(Class = 1|x^*) = \frac{0.6P(x^*|Class=1)}{0.6P(x^*|Class=1)+0.4P(x^*|Class=2)}$ for class 1 and $P(Class = 1|x^*) = \frac{0.4P(x^*|Class=1)}{0.6P(x^*|Class=1)+0.4P(x^*|Class=2)}$for class 2.