January 2021

Assignment 1

Task 2

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

#preparing data
data <- data[-c(1,3,4)]
#data$AGE <- data$AGE/100

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

```
#likelihood function
neg_likelihood<-function(input_data,w){

  Y <- as.matrix(input_data[,3])
  X <- input_data[,-3]
  X0 <- rep(1,nrow(input_data))
  X_new <- cbind(X0,X)
  n <- nrow(input_data)
  logl <- 0
  for (i in 1:length(Y)) {
    logl <- sum(logl + log(1 + exp(-Y[i]*t(w)*X[i,])))
    }
  return(logl/n) # negative log-likelihood
}

parameter_a <- c(0,1,0)
neg_likelihood_a <- neg_likelihood(train,parameter_a)

parameter_b <- c(0,0,1)
neg_likelihood_b <- neg_likelihood(train,parameter_b)

parameter_c <- c(1,1,1)
neg_likelihood_c <- neg_likelihood(train,parameter_c)
```

```
df_neg_loglik <- data.frame("Negative Log Likelihodd" =c(neg_likelihood_a,
                                            neg_likelihood_b,
                                            neg_likelihood_c))

row.names(df_neg_loglik) <- c("w=(0,1,0)","w=(0,0,1)","w=(1,1,1)")
knitr::kable(df_neg_loglik)
```

	Negative.Log.Likelihodd
w=(0,1,0)	9.557426e+237
w=(0,0,1)	1.155479e+238
w=(1,1,1)	9.458429e+237

```
# negative_log_like <- function(training_data, w){
#   Y <- training_data$default_payment # reponse
#   X <- cbind(rep(1, length(Y)),training_data$AGE / 100, training_data$SEX) #predictors
#   n <- nrow(X)
#
#   logl <- -sum(
#     Y*(X%*%w - log(1+exp(X%*%w))) + (1-Y)*(-log(1+exp(X%*%w))))
#
#   return(logl) # negative log-likelihood
# }
# negative_log_like(train, w = c(0,1,0))
# negative_log_like(train, w = c(0,0,1))
# negative_log_like(train, w = c(1,1,1))
```

The negative log-likelihood value of a regression model is a way to measure the goodness of fit for a model. The lower the value of the log-likelihood, the better a model fits a data set. Logistic regression is a classical linear method for binary classification. By comparing, the log-likelihood values from the above table, it could be seen that the log-likelihood using the third parameter vector $(1, 1, 1)$ has the lowest value; thus the target value would be better predicted by using the second parameter vector.

Task 3

```
optimal <- function(input_data){
  res <- optim( par = c(1,1,1), fn = neg_likelihood, input_data = train)
  return(res)
}

best_par <- optimal(train)
best_par$par
```

```
## [1]  36.9   4.5 -24.2
```

The decision boundary is:

$$33.25926 - 271.92593Sex - 97.49630Age$$

```
library(glmnet)
```

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

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

```r
log_reg <- glm(default_payment ~ ., data = train, family=binomial)

pred_train <- predict(log_reg, newdata = train)
pred_test <- predict(log_reg, newdata = test)

misclass <- function(actual_val,fitted_val){
  confusion_matrix <- table(actual_val,fitted_val)
  n <- length(actual_val)
  error <- 1 - (sum(diag(confusion_matrix))/n)
  return(error)
}

train_error <- misclass(train$default_payment,pred_train)
test_error <- misclass(test$default_payment,pred_test)

df_error <- data.frame("Train error" = train_error,
                       "Test error" = test_error)

row.names(df_error) <- c("Misclassification Rates")
knitr::kable(df_error)
```

	Train.error	Test.error
Misclassification Rates	0.99375	0.9933333

From the above table it could be seen that both of the errors are pretty high. Both values are almost the same, the test error provided a slightly worse error compared to the test error.

Assignment 2

Mixture Models (Semi-Supervised GMM)

```r
set.seed(1234567890)
min_change <- 0.1 # min change in log likelihood between two consecutive EM iterations
N=300 # number of training points
D=10 # number of dimensions
x <- matrix(nrow=N, ncol=D) # training data
true_pi <- vector(length = 3) # true mixing coefficients
true_mu <- matrix(nrow=3, ncol=D) # true conditional distributions
true_pi=c(1/3, 1/3, 1/3)
true_mu[1,]=c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
true_mu[2,]=c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
true_mu[3,]=c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5)
# plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
# points(true_mu[2,], type="o", col="red")
# points(true_mu[3,], type="o", col="green")
```

```r
true_k <- array(dim = 300) # true component labels

# Producing the training data
for(n in 1:N) {
  k <- sample(1:3,1,prob=true_pi)
```

```r
  # 30 % of the training points have a label
  true_k[n] <- k * sample(0:1,1,prob = c(1,0))
  for(d in 1:D) {
    x[n,d] <- rbinom(1,1,true_mu[k,d]) }
}
```

```r
K = 3
w <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional distributions
```

```r
# Assuming that we know the true class
# of 10 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
```

```r
# Mean of the known components for each class
mu[1, ] <- colMeans(x[1:100, ])
mu[2, ] <- colMeans(x[101:110, ])
mu[3, ] <- colMeans(x[201:210, ])
#mu
```

```r
# set max_it to some value
# max number of EM iterations
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) {
  # plot(mu[1,], type="o", col="blue", ylim=c(0,1))
  # points(mu[2,], type="o", col="red")
  # points(mu[3,], type="o", col="green")
  # points(mu[4,], type="o", col="black")
  Sys.sleep(0.5)

  # E-step: Computation of the fractional component assignments
  mux <- matrix(nrow=N, ncol=K)
  for (n in 1:N) {
    for (k in 1:K) {
      mux[n,k] <- prod(mu[k,]^x[n,],(1-mu[k,])^(1-x[n,]))
    }
  }
  w <- t(pi*t(mux))/rowSums(t(pi*t(mux)))

  # 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){
```

```r
      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.
  # E <- sum(log(rowSums(t(pi*t(mux)))))
  E <- 0
  for (n in 1:N) {
    for (k in 1:K) {
      a <- 0
        for (i in 1:D) {
          a <- a+x[n,i]*log(mu[k,i])+(1-x[n,i])*log(1-mu[k,i])
        }
      a <- log(pi[k])+a
      E <- E+w[n,k]*a
    }
  }

  llik[it] <- E
  #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
  pi <- colSums(w)/N
  for (k in 1:K) {
    for (i in 1:D) {
      mu[k,i] <- x[,i]%*%w[,k]/sum(w[,k])
    }
  }
}
# plot(mu[1,], type="o", col="blue", ylim=c(0,1))
# points(mu[2,], type="o", col="red")
# points(mu[3,], type="o", col="green")
# plot(llik[1:it], type="o", col="navy")
```