ML Block 2 Lab 1

Zoe Patton, Rojan Karakaya, Suhani Ariga

11/20/2020

Contents

Contributions

For Block 2 Lab 1, Zoe focused on assignment one, Rojan focused on assignment two and Suhani focused on assignment three.

Assignment 1: Ensemble Methods

Creating the training data and loading required package:

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.0.3
```

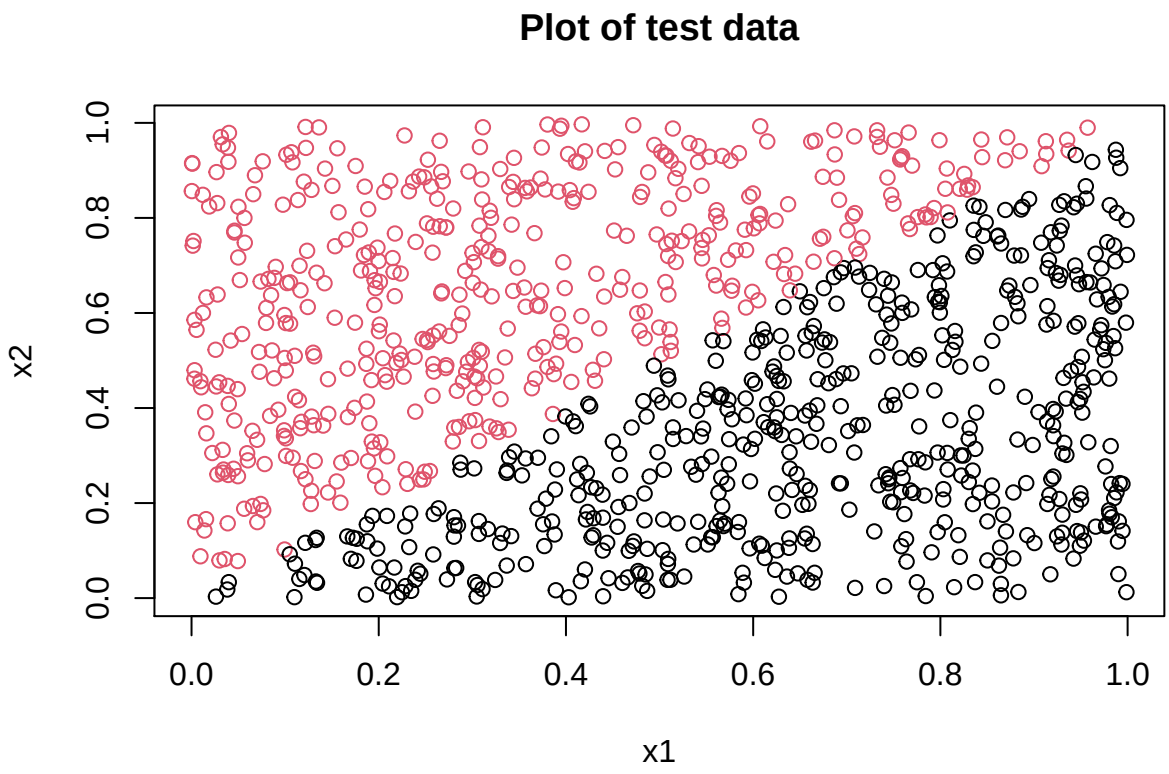
```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
set.seed(1234)
trdata=list()
for(i in 1:1000){
  x1<-runif(100)
  x2<-runif(100)
  trdata[[i]]<-cbind(x1,x2)
}

set.seed(1234)
trlabels=list()
for(i in 1:1000){
  trlabels[[i]]<-as.factor(as.numeric(trdata[[i]][,1]<trdata[[i]][,2]))
}
```

a.

```
#creating training data
x1<-runif(100)
x2<-runif(100)
trdata<-cbind(x1,x2)
y<-as.numeric(x1<x2)
trlabels<-as.factor(y)

a_tree1<-replicate(n=1000,
                   exp=randomForest(trlabels~.,data=trdata,importance=TRUE,
                                    ntree=1,nodesize=25,keep.forest = TRUE),
                   simplify=FALSE)

a_tree10<-replicate(n=1000,
                    exp=randomForest(trlabels~.,data=trdata,importance=TRUE,
                                     ntree=10,nodesize=25,keep.forest = TRUE),
                    simplify=FALSE)

a_tree100<-replicate(n=1000,
                    exp=randomForest(trlabels~.,data=trdata,importance=TRUE,
                                     ntree=100,nodesize=25,keep.forest = TRUE),
                    simplify=FALSE)
```



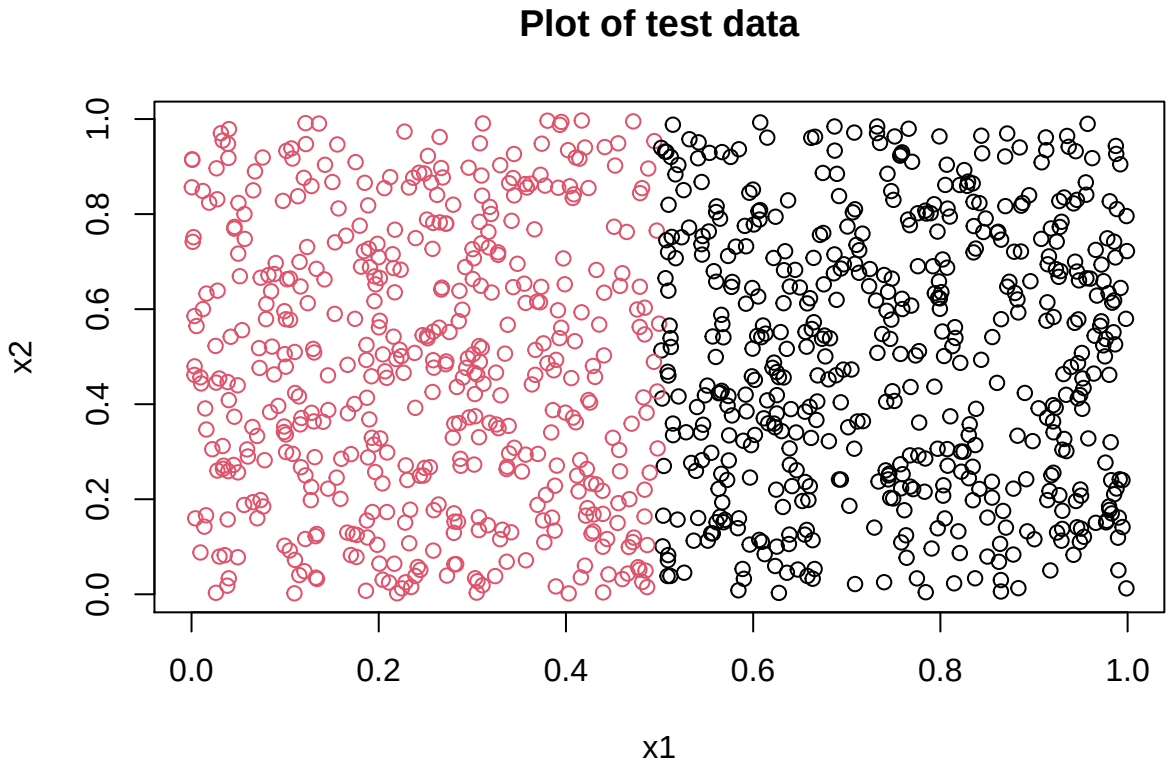
Statistical values:

```
##                1 tree      10 tree    100 trees
## Mean     0.202904000 0.1352080000 0.0982970000
## Variance 0.003969292 0.0008494722 0.0002430158
```

b.

```
#creating training data
x1<-runif(100)
x2<-runif(100)
trdata<-cbind(x1,x2)
y<-as.numeric(x1<0.5)
trlabels<-as.factor(y)

#learn random forests with 1, 10, 100 trees to training data
b_tree1<-replicate(n=1000,
                   exp=randomForest(trlabels~.,data=trdata,importance=TRUE,
                                    ntree=1,nodesize=25,keep.forest = TRUE),
                   simplify=FALSE)

b_tree10<-replicate(n=1000,
                    exp=randomForest(trlabels~.,data=trdata,importance=TRUE,
                                     ntree=10,nodesize=25,keep.forest = TRUE),
```

```
                    simplify=FALSE)

b_tree100<-replicate(n=1000,
                     exp=randomForest(trlabels~.,data=trdata,importance=TRUE,
                                      ntree=100,nodesize=25,keep.forest = TRUE),
                     simplify=FALSE)
```



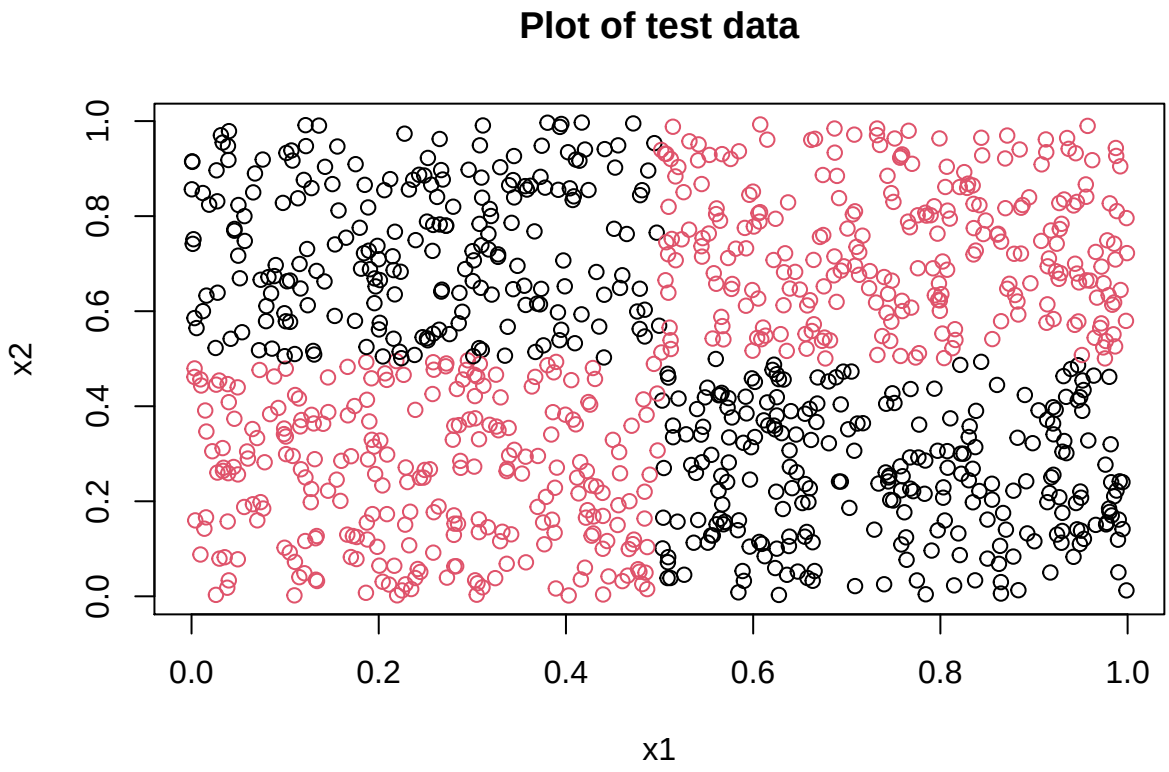
Statistical values:

```
##              1 tree      10 tree    100 trees
## Mean     0.08700800 0.0119830000 6.297000e-03
## Variance 0.01457772 0.0002248696 2.683474e-06
```

c.

```
#creating training data
x1<-runif(100)
x2<-runif(100)
trdata<-cbind(x1,x2)
y<-as.numeric((x1<0.5 & x2<0.5) | (x1>0.5 & x2>0.5))
trlabels<-as.factor(y)

#learn random forests with 1, 10, 100 trees to training data
```

```
c_tree1<-replicate(n=1000,
                   exp=randomForest(trlabels~.,data=trdata,importance=TRUE,
                                    ntree=1,nodesize=12,keep.forest = TRUE),
                   simplify=FALSE)

c_tree10<-replicate(n=1000,
                    exp=randomForest(trlabels~.,data=trdata,importance=TRUE,
                                     ntree=10,nodesize=12,keep.forest = TRUE),
                    simplify=FALSE)

c_tree100<-replicate(n=1000,
                     exp=randomForest(trlabels~.,data=trdata,importance=TRUE,
                                      ntree=100,nodesize=12,keep.forest = TRUE),
                     simplify=FALSE)
```



Statistical values:

```
##              1 tree     10 tree    100 trees
## Mean     0.22081300 0.075267000 3.403500e-02
## Variance 0.01421297 0.001196452 1.447425e-05
```

d.

Means of error rate for each case:

```
##                1       10      100
## Case a 0.202904 0.135208 0.098297
## Case b 0.087008 0.011983 0.006297
## Case c 0.220813 0.075267 0.034035
```

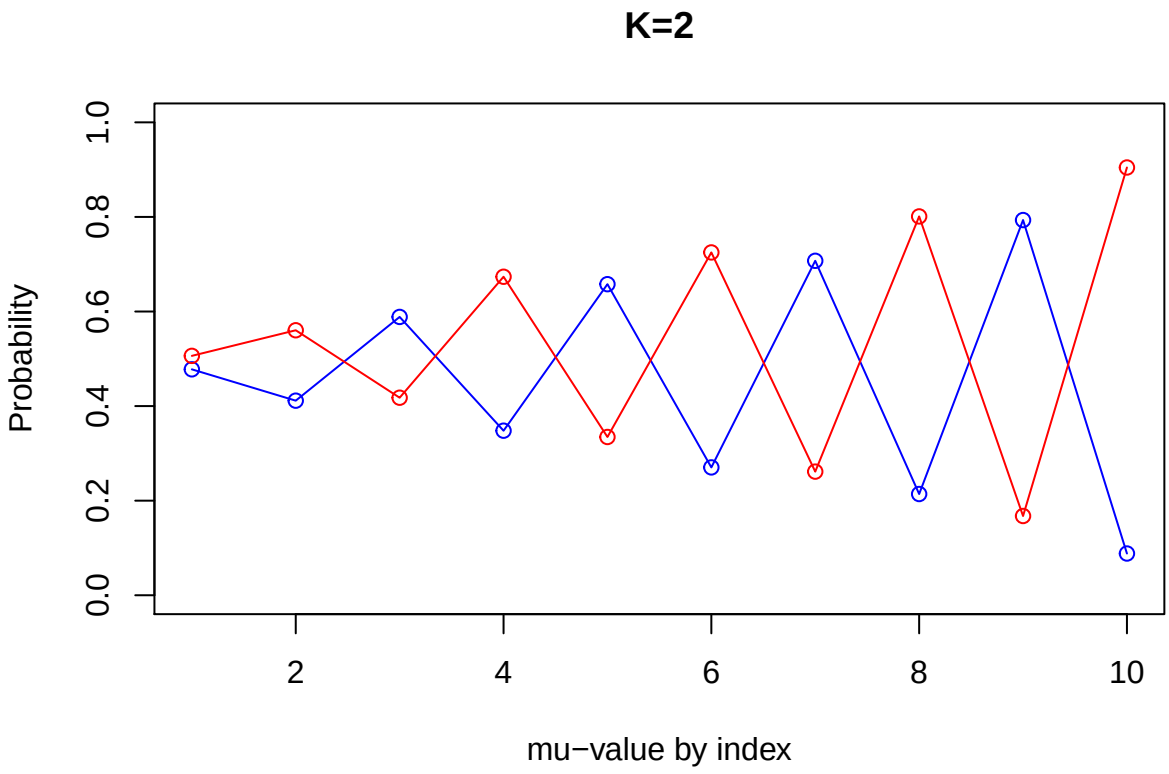
Variance of error rate for each case:

```
##                   1           10          100
## Case a 0.003969292 0.0008494722 2.430158e-04
## Case b 0.014577716 0.0002248696 2.683474e-06
## Case c 0.014212973 0.0011964522 1.447425e-05
```

As the number of trees in the random forest grows, the mean of error rate decreases for each case. Similarly, the variance of the error rate also decreases for each case. There is better performance for the third dataset because the number of classes is reduced. The more complicated classification of y-values results in less evaluation of classes in the random forest. It is desirable to have low error variance because it is a measure of how much out prediction would change if different training data is used. A lower variance means that a general trend is recognized that can be used to find fairly accurate predictions on different sets of data. A high variance likely indicates overfitting to the training data, establishing false patterns within the data caused by randomness.

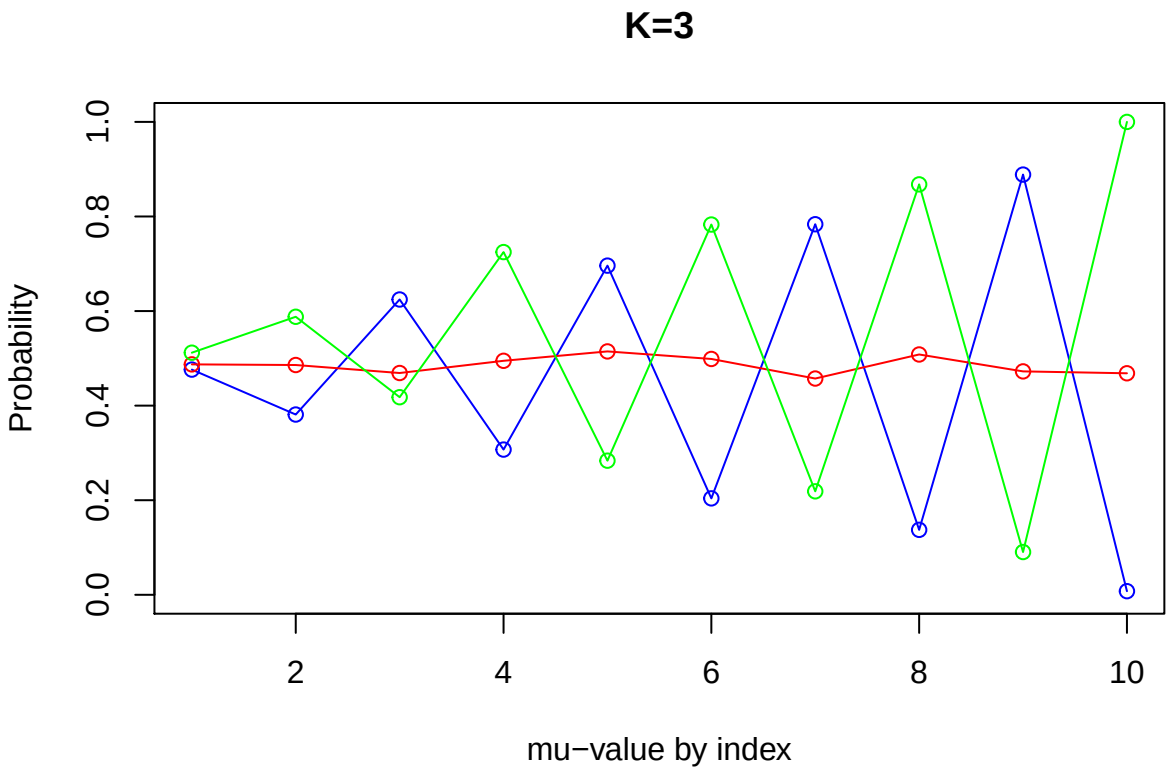
Assignment 2: Mixture Models

The fewer the clusters, the faster the convergence. Likelihood increases as the number of clusters increase, although the increase is much smaller when going from K=3 to K=4 than when going from K=2 to K=3.



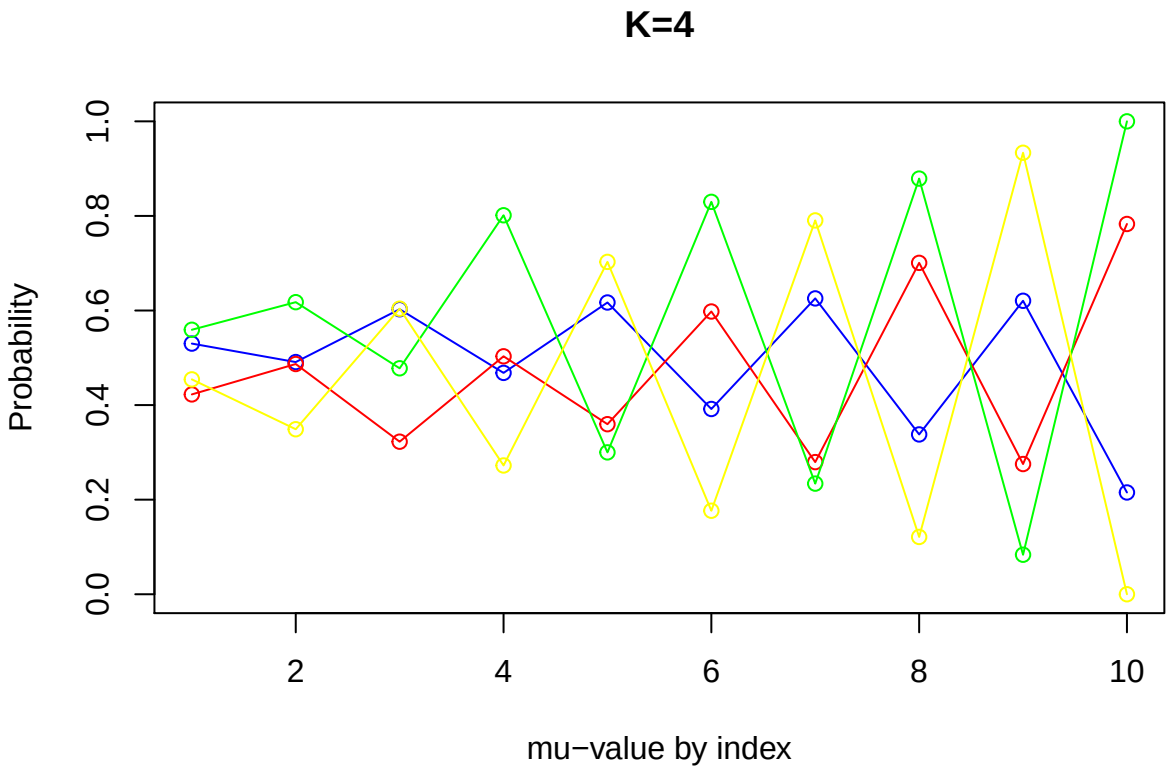
For the case where K=2, the graph for the mu-values of the two fitted distributions is in line with two of the three original distributions, except for the fact that all the values are biased towards 0.5. The third original one has its mu-values right between the other twos'; it is clear that the algorithm has divided the points from that distribution among the other two, as that is the best it can do given the starting assumption. In other words, this model underfits the data and the values of the middle cluster get baked in to the estimates of the outer ones.

Convergence after 16 iterations with a log-likelihood of -5803.425.



The case with three clusters looks like the original graph, barring some noise.

Convergence after 39 iterations with a log-likelihood of -5647.483.



The case with four clusters show clear signs of overfitting. The outermost lines are more extreme than the two original zig-zagging lines were, and the innermost ones are much less so. The algorithm has "split" the middle line into two clusters, and in so doing has added some data points from the zig-zagging distributions. The result is that the mu-values of the outermost clusters move further out from the middle, whereas the inner ones replicate the zig-zag pattern.

Convergence after 83 iterations with a log-likelihood of -5625.059.

R-code

```
#rm(list=ls())
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=1000 # 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)
```

```r
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")

# Producing the training data
for(n in 1:N) {
  k <- sample(1:3,1,prob=true_pi)
  for(d in 1:D) {
    x[n,d] <- rbinom(1,1,true_mu[k,d])
  }
}

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 distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations

# Random initialization of the paramters
pi <- runif(K,0.49,0.51)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0.49,0.51)
}
pi
mu
```

```r
normalize=function(xx){
  return(xx/sum(xx))
}
#y_{i,j}
clusterBelongingProbability=function(xx,mumu,NN,KK){
  #likelihood of belonging to a certain cluster times the probability of that cluster / all other likel
  notX = 1-xx

  lik1 = matrix(nrow=NN, ncol=KK)
  for(row in 1:KK){
    oneMinusMu=1-mumu[row,]
    x21=t(mumu[row,]*t(xx))
    x22=t(oneMinusMu*t(notX))
    x2=x21+x22

    lik1[,row]=apply(x2,1,prod)
  }
  pp=pi*lik1
  probs=apply(pp,1,normalize)

  return(probs)
}
```

```
newMuAndPi=function(xx,mumu,NN,KK){
  #mean is ML-estimator for p
  probs=clusterBelongingProbability(xx,mumu,NN,KK)

  newMu=matrix(nrow=KK, ncol=ncol(xx))
  newPi1=apply(probs,1,sum)
  newMu=probs%*%xx / apply(probs,1,sum)

  parameters=list(mu=newMu,pi=newPi1/N)
  return(parameters)
}
logLikelihoodGivenMuPi=function(xx,mumu,NN,KK){
  notX=1-xx
  probs=clusterBelongingProbability(xx,mumu,NN,KK)

  llikelihood=0
  for(row in 1:KK){
    oneMinusMu=1-mumu[row,]
    x21=t(mumu[row,]*t(xx))
    x22=t(oneMinusMu*t(notX))
    x2=x21 + x22

    x3=probs[row,]*log(x2)

    llikelihood=llikelihood + sum(x3)
  }
  return(llikelihood)
}



llik=rep(0,100)#max_it)
for(it in 1:100) {#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="yellow")
  Sys.sleep(0.5)


  llik[it]=logLikelihoodGivenMuPi(x,mu,N,K)

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()

  if (it>1){
    if (abs(llik[it]-llik[it-1])<0.1){
      break
    }
  }

  params=newMuAndPi(x,mu,N,K)
  mu=params$mu
```

```
  pi=params$pi
}
pi
mu
plot(llik[1:it], type="o")
```

Assignment 3: High Dimensional Methods

Loading the below required packages and data.

```
library(pamr)
```

```
## Warning: package 'pamr' was built under R version 4.0.3
```

```
library(kernlab)
```

```
## Warning: package 'kernlab' was built under R version 4.0.3
```

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.0.3
```

```
library(sgof)
```

```
## Warning: package 'sgof' was built under R version 4.0.3
```

```
## Warning: package 'poibin' was built under R version 4.0.3
```

```
library(readr)
library(ggplot2)
```

```
## Warning: replacing previous import 'vctrs::data_frame' by 'tibble::data_frame'
## when loading 'dplyr'
```

1.

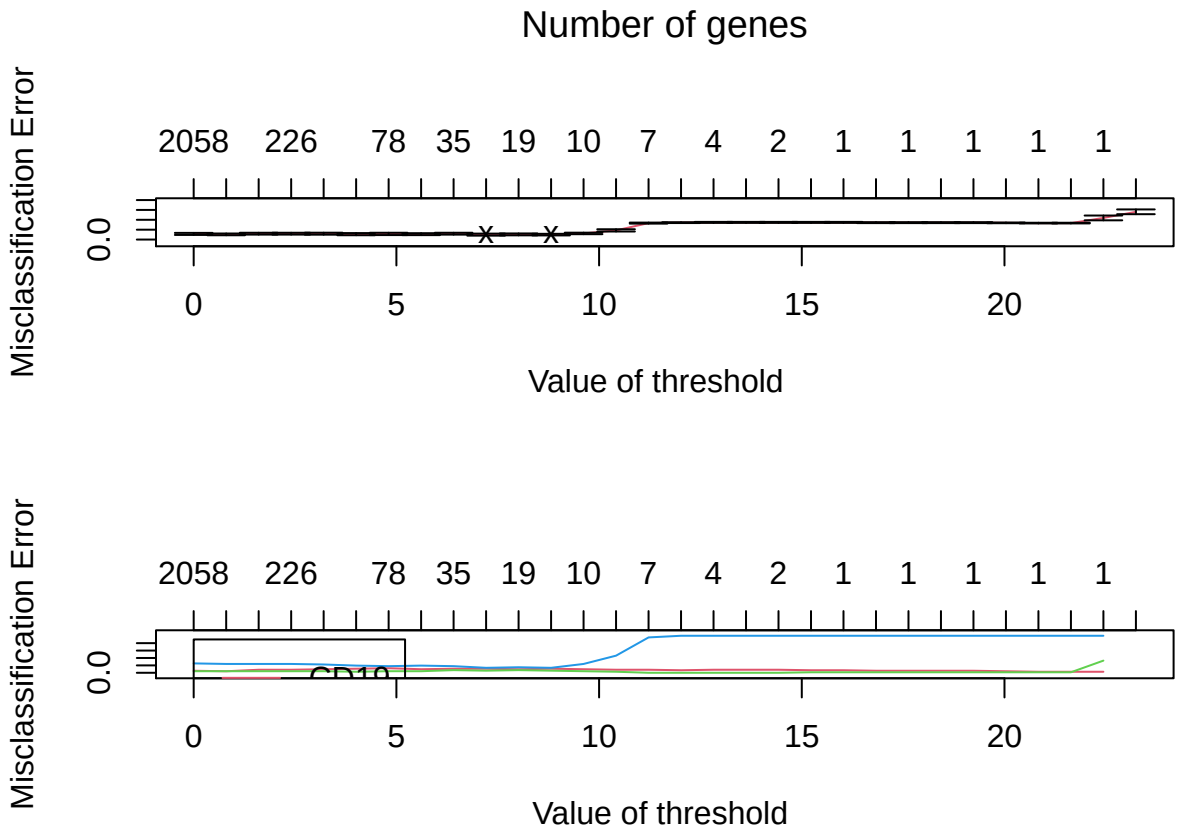
Importing R data. The data is divided into testing and training data sets. 70% of randomly selected data goes to training data set, and remaining 30% are used in the testing data set.

```
geneexp = read_csv("geneexp.csv")
data <- geneexp[ ,-1]
set.seed(12345)
n <- dim(data)[1]
data$CellType <- as.factor(data$CellType)
ind   <- sample(1:n, floor(n*0.7))
train <- data[ind,]
test  <- data[-ind,]
```

Performing Nearest Shrunken Centroid classification

```r
#train
rownames(train)=1:nrow(train)
x_train = t(train[,-ncol(train)])
y_train = train[[ncol(train)]]
mytrain_data    <- list(x = x_train,
                        y = as.factor(y_train),
                        geneid    = as.character(1:nrow(x_train)),
                        genenames = colnames(train[,-ncol(train)]))

#test
rownames(test)=1:nrow(test)
x_test = t(test[ ,-ncol(test)])
y_test = test[[ncol(test)]]
```

```r
# Cross Validation for the shrunken centroid
model = pamr.train(mytrain_data)
cvmodel = pamr.cv(model, mytrain_data)
pamr.plotcv(cvmodel)
```

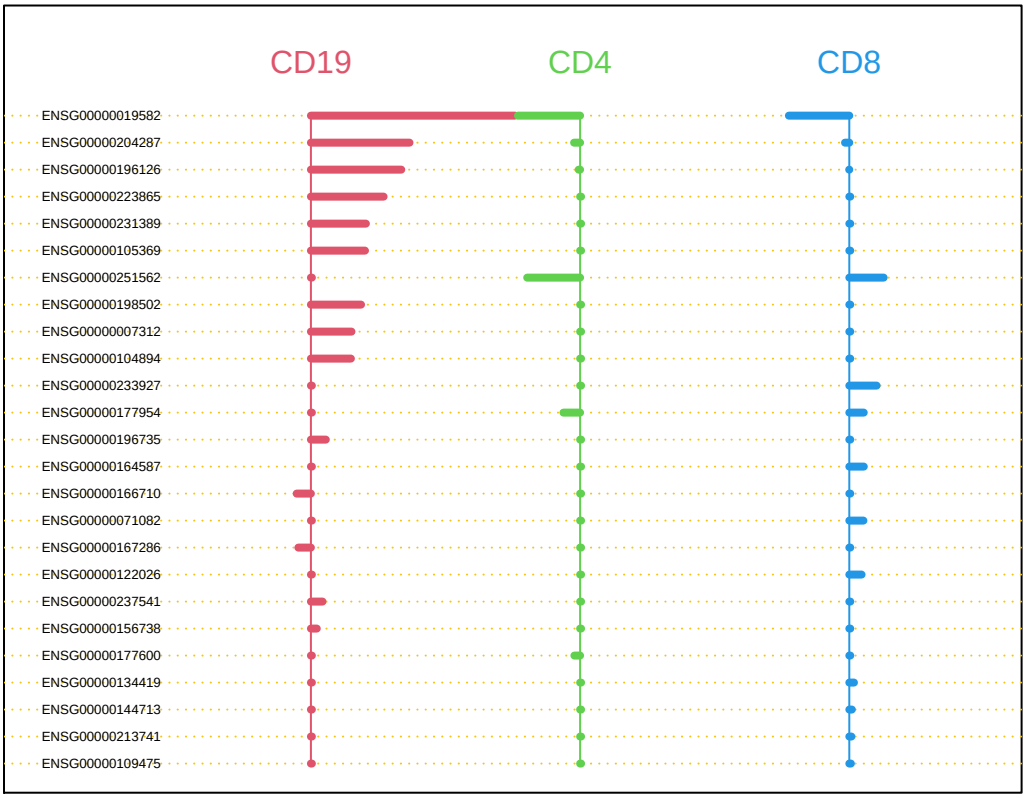


The threshold for the min error of cvmodel

```r
best_threshold = cvmodel$threshold[which.min(cvmodel$error)]
cat(paste("The optimal threshold : ", best_threshold))
```

```
## The optimal threshold :  7.2129439923563
```

```
# Training a model with the best threshold from the cross validation
pamr.plotcen(model, mytrain_data, threshold = best_threshold)
```



Number of genes selected

```
features_nsc = pamr.listgenes(model,mytrain_data,threshold=best_threshold)
```

```
##       id  CD19-score CD4-score CD8-score
##  [1,] 2   1.5153     -0.4617   -0.4489
##  [2,] 15  0.7328     -0.0448   -0.032
##  [3,] 31  0.6715     -0.0105   -0.001
##  [4,] 32  0.5403     0         0
##  [5,] 37  0.4089     0         0
##  [6,] 138 0.4019     0         0
##  [7,] 1   0          -0.3933   0.2553
##  [8,] 90  0.3727     0         0
##  [9,] 172 0.3016     0         0
## [10,] 126 0.2974     0         0
## [11,] 79  0          0         0.2028
## [12,] 11  0          -0.1239   0.1074
## [13,] 110 0.1109     0         0
## [14,] 21  0          0         0.1078
## [15,] 3   -0.1057    0         0
## [16,] 50  0          0         0.1051
## [17,] 207 -0.0933    0         0
```

```
## [18,] 29  0          0         0.0914
## [19,] 192 0.0862     0         0
## [20,] 309 0.0438     0         0
## [21,] 28  0          -0.0426   0
## [22,] 38  0          0         0.0346
## [23,] 18  0          0         0.0189
## [24,] 127 0          0         0.0156
## [25,] 36  0          0         0.0119
```

```
cat(paste("Number of genes selected : ", nrow(features_nsc)))
```

```
## Number of genes selected :  25
```

Centroid plot shows distance from gene mean value to global mean value. Positive values means that the gene mean is greater than the global mean and negative values means that the gene mean is smaller than the global mean. No, it can't happen that all the values for some gene in centroid plot are positive.

2.

The 2 most contributing features:

```
feature = colnames(data)[as.numeric(features_nsc[,1])][1:2]
feature
```

```
## [1] "ENSG00000019582" "ENSG00000204287"
```

The alternative name of "ENSG00000019582" gene is CD74 and "ENSG00000204287" gene is HLA-DRA. Above 2 genes are marker genes for B cells.

Test error nearest shrunk

```
pred_nsc = pamr.predict(model,
                        newx = x_test,
                        threshold = best_threshold)
tab_nsc = table(y_test, pred_nsc)
mis_nsc  <- 1 - sum(diag(tab_nsc)) / sum(tab_nsc)
cat(paste("Test Error:", round(mis_nsc,5)))
```

```
## Test Error: 0.11111
```

```
res_nsc <- list("Test Error" = round(mis_nsc,5), "Features Selected" = nrow(features_nsc))
```

3.

a. Elastic net

```
elastic_net = cv.glmnet(x = t(x_train),
                        y = y_train,
                        family = "multinomial",
                        alpha = 0.5)
# create prediction
pred_elastic = predict(elastic_net,
                       newx = t(x_test),
                       type = "class",
                       s = elastic_net$lambda.min)
# s penalty parameter
tab_elastic  = table(y_test, pred_elastic)
mis_elastic <- 1 - (sum(diag(tab_elastic))/sum(tab_elastic))
min_lambda = elastic_net$lambda.min
features_elastic = as.numeric(elastic_net$nzero[which(elastic_net$lambda==min_lambda)])
res_elastic <- list("Error Rate" = round(mis_elastic,5), "Features Selected" = features_elastic)
```

b. Support Vector Machine

```
ign <- capture.output({
svm_fit  = ksvm(x = t(x_train),
                y = y_train,
                kernel = "vanilladot")
})
pred_svm = predict(svm_fit,
                   newdata = t(x_test),
                   type = "response")

tab_svm = table(y_test, pred_svm)

mis_svm = 1 - (sum(diag(tab_svm))/sum(tab_svm))
features_svm = ncol(data)
res_svm <- list("Error Rate" = round(mis_svm,5), "Features Selected" = features_svm)
```

Comparative table

```
result <- rbind("NSC" = res_nsc, "Elastic Net" = res_elastic, "SVM" = res_svm)

# print the test error rates and contributing features for all models
knitr::kable(result, caption = "Comparsion of Models on Test dataset")
```

Table 1: Comparsion of Models on Test dataset

	Test Error	Features Selected
NSC	0.11111	25
Elastic Net	0.05556	54
SVM	0.02222	2086

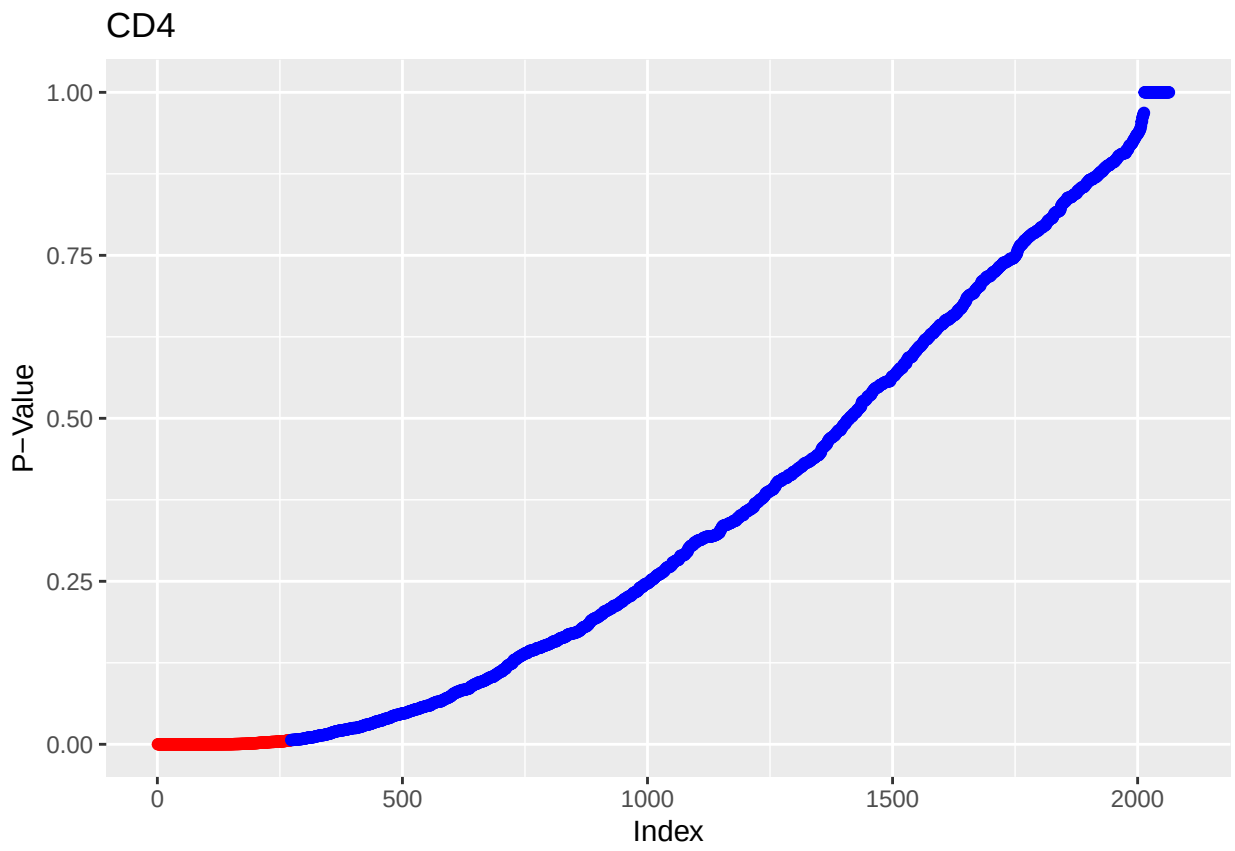
From the model comparison, we see that overall choosing SVM gives us the best accuracy, while NSC Model and Elastic Net model both have the lesser accuracy than SVM, however number of features is more in SVM which would be overfit. And NSC has less accuracy with less number of features which is not feasible. Hence

we prefer Elastic net.

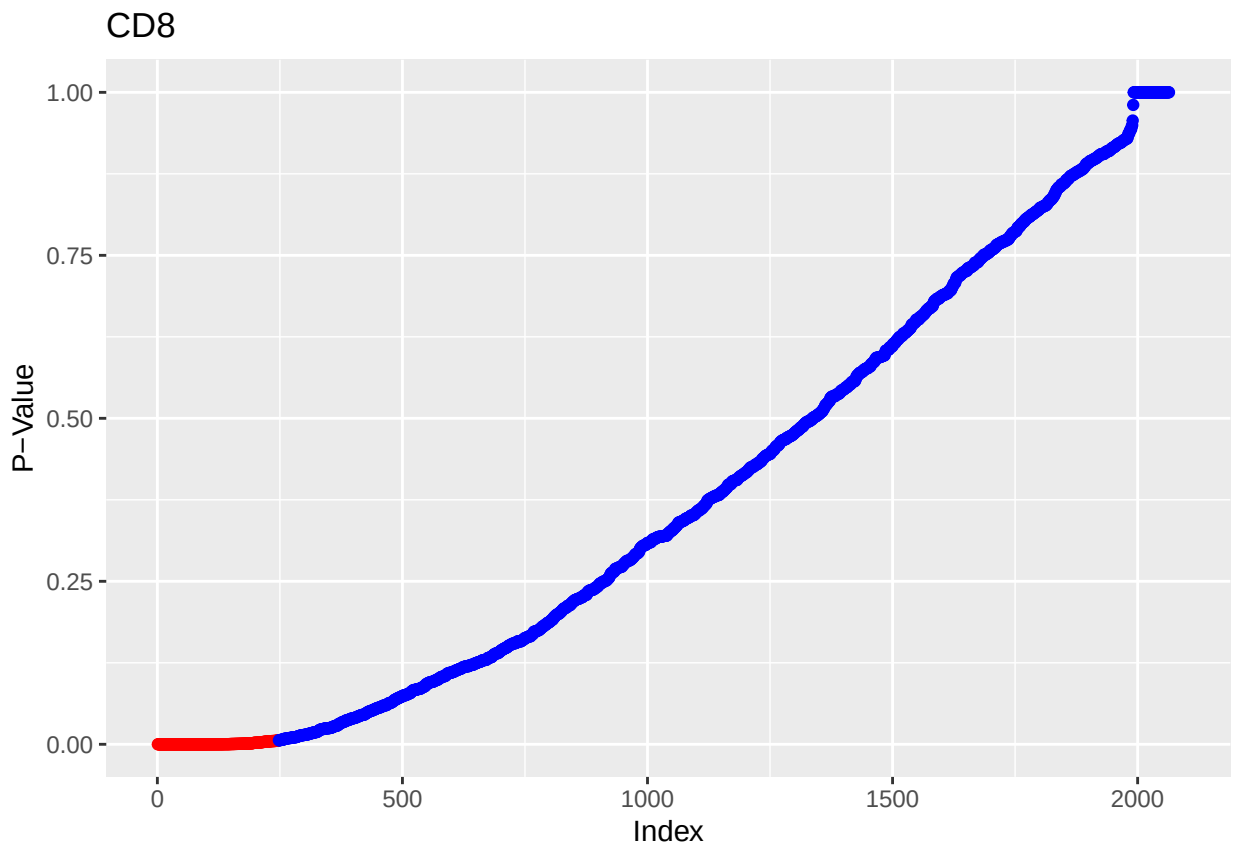
4.

Implement Benjamini-Hochberg method and compute p-values

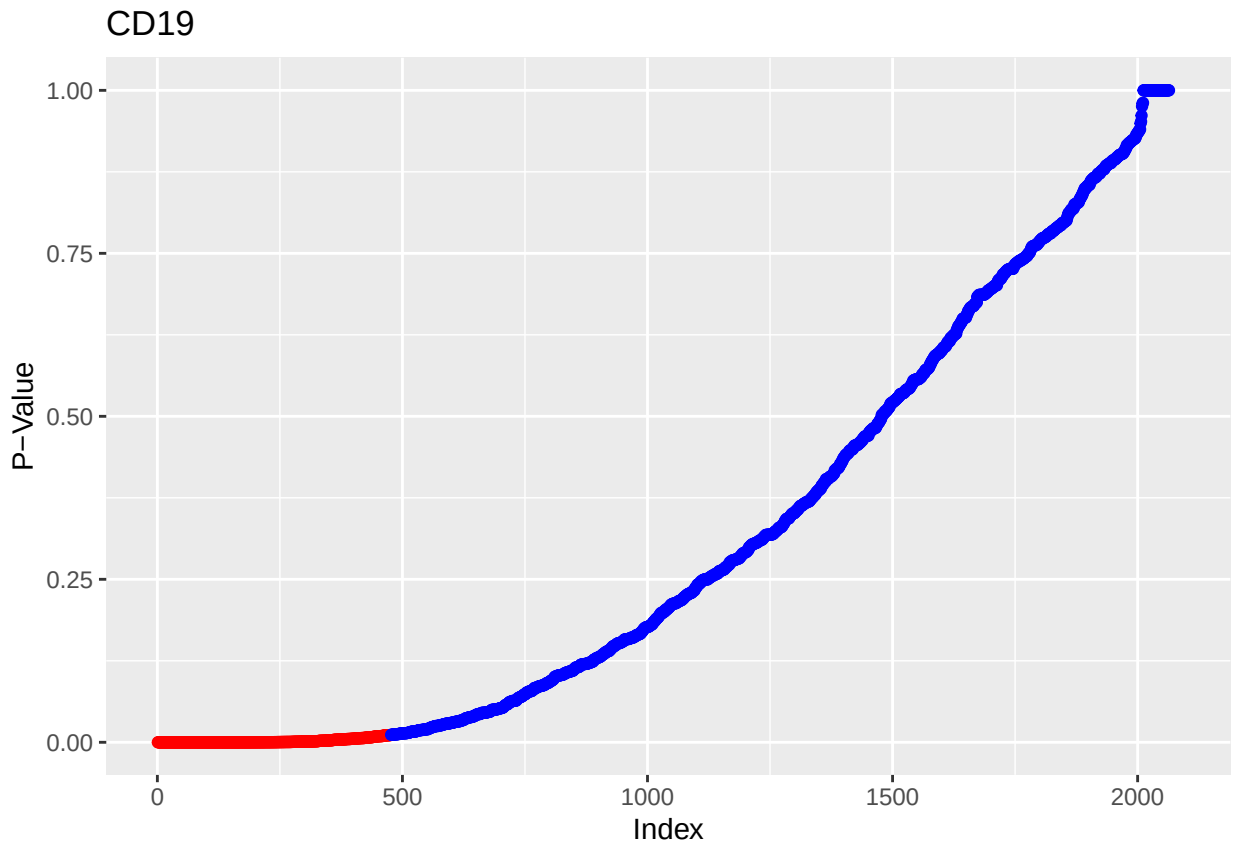
```
BH = function(cell_name){
y <- ifelse(data$CellType == cell_name, 1, 0)
df <- data.frame(name=character(),pvalue=numeric())
for (i in 1:(ncol(data)-1)){
  test <- t.test(unlist(data[,i])~y, data = data, alternative = "two.sided")$p.value
  test_df = data.frame(name=colnames(data)[i],pvalue=test)
  df <- rbind(df, test_df)
 }
df<-df[order(as.numeric(df$pvalue)),]
  M = ncol(data)-1
  alpha = 0.05
  for (i in 1:M) {
    if(as.numeric(df$pvalue[i]) > (alpha * i/M)){
      break
    }
  }
  L = i-1
  total <<- rbind(total,c(cell_name,L))
  p <- as.numeric(df$pvalue[L])
  rejected <- ifelse(as.numeric(df$pvalue) <= p,1,0)

  row.names(df) = c(1:length(df$pvalue))

  ggplot() +
  ylab("P-Value") + xlab("Index") + ggtitle(cell_name) +
  geom_point(data=data.frame(x=c(1:L),
                             y=df$pvalue[1:L]),
             aes(x=x, y=y), col="red") +
  geom_point(data=data.frame(x=c(L+1:M),y=df$pvalue[L+1:M]),
             aes(x=x, y=y), col="blue") +
  scale_y_continuous(limits=c(0,1)) + scale_x_continuous(limits = c(0, M))
}

total = data.frame(name=character(),total = numeric())

BH("CD4")
```




```
BH("CD8")
```




```
BH("CD19")
```




```
colnames(total) = c("Cell", "Rejected")
knitr::kable(total, caption = "Genes correspond to the rejected hypothesis for each cell type")
```

Table 2: Genes correspond to the rejected hypothesis for each cell type

Cell	Rejected
CD4	272
CD8	247
CD19	476

In the above plot, points represents genes and red color in the plot are the rejected genes.

Apendix

```
knitr::opts_chunk$set(echo = TRUE)
library(randomForest)

set.seed(1234)
trdata=list()
for(i in 1:1000){
```

```
  x1<-runif(100)
  x2<-runif(100)
  trdata[[i]]<-cbind(x1,x2)
}

set.seed(1234)
trlabels=list()
for(i in 1:1000){
  trlabels[[i]]<-as.factor(as.numeric(trdata[[i]][,1]<trdata[[i]][,2]))
}
#creating training data
x1<-runif(100)
x2<-runif(100)
trdata<-cbind(x1,x2)
y<-as.numeric(x1<x2)
trlabels<-as.factor(y)

a_tree1<-replicate(n=1000,
                   exp=randomForest(trlabels~.,data=trdata,importance=TRUE,
                                    ntree=1,nodesize=25,keep.forest = TRUE),
                   simplify=FALSE)

a_tree10<-replicate(n=1000,
                    exp=randomForest(trlabels~.,data=trdata,importance=TRUE,
                                     ntree=10,nodesize=25,keep.forest = TRUE),
                    simplify=FALSE)

a_tree100<-replicate(n=1000,
                    exp=randomForest(trlabels~.,data=trdata,importance=TRUE,
                                     ntree=100,nodesize=25,keep.forest = TRUE),
                    simplify=FALSE)
#producing test data
set.seed(1234)
x1<-runif(1000)
x2<-runif(1000)
tedata<-cbind(x1,x2)
y<-as.numeric(x1<x2)
telabels<-as.factor(y)
plot(x1,x2,col=(y+1),main = "Plot of test data")

#predicting on test data
a_predTest1<-list()
a_error1<-list()
for(i in 1:1000){
  a_predTest1[[i]]<-predict(a_tree1[[i]],tedata,type="class")
  a_error1[[i]]<-mean(a_predTest1[[i]] != telabels)
}

a_predTest10<-list()
a_error10<-list()
for(i in 1:1000){
  a_predTest10[[i]]<-predict(a_tree10[[i]],tedata,type="class")
  a_error10[[i]]<-mean(a_predTest10[[i]] != telabels)
```

```
}

a_predTest100<-list()
a_error100<-list()
for(i in 1:1000){
  a_predTest100[[i]]<-predict(a_tree100[[i]],tedata,type="class")
  a_error100[[i]]<-mean(a_predTest100[[i]] != telabels)
}

#reporting missclassification results
a_me1<-mean(unlist(a_error1))
a_me10<-mean(unlist(a_error10))
a_me100<-mean(unlist(a_error100))

a_ve1<-var(unlist(a_error1))
a_ve10<-var(unlist(a_error10))
a_ve100<-var(unlist(a_error100))

a_me<-c(a_me1,a_me10,a_me100)
a_ve<-c(a_ve1,a_ve10,a_ve100)

a<-rbind(a_me,a_ve)
row.names(a)<-c("Mean","Variance")
colnames(a)<-c("1 tree","10 tree","100 trees")
a
#creating training data
x1<-runif(100)
x2<-runif(100)
trdata<-cbind(x1,x2)
y<-as.numeric(x1<0.5)
trlabels<-as.factor(y)

#learn random forests with 1, 10, 100 trees to training data
b_tree1<-replicate(n=1000,
                   exp=randomForest(trlabels~.,data=trdata,importance=TRUE,
                                    ntree=1,nodesize=25,keep.forest = TRUE),
                   simplify=FALSE)

b_tree10<-replicate(n=1000,
                    exp=randomForest(trlabels~.,data=trdata,importance=TRUE,
                                     ntree=10,nodesize=25,keep.forest = TRUE),
                    simplify=FALSE)

b_tree100<-replicate(n=1000,
                     exp=randomForest(trlabels~.,data=trdata,importance=TRUE,
                                      ntree=100,nodesize=25,keep.forest = TRUE),
                     simplify=FALSE)
#producing test data
set.seed(1234)
x1<-runif(1000)
x2<-runif(1000)
b_tedata<-cbind(x1,x2)
y<-as.numeric(x1<0.5)
```

```
b_telabels<-as.factor(y)
plot(x1,x2,col=(y+1),main = "Plot of test data")

#predicting on test data
b_predTest1<-list()
b_error1<-list()
for(i in 1:1000){
  b_predTest1[[i]]<-predict(b_tree1[[i]],b_tedata,type="class")
  b_error1[[i]]<-mean(b_predTest1[[i]] != b_telabels)
}

b_predTest10<-list()
b_error10<-list()
for(i in 1:1000){
  b_predTest10[[i]]<-predict(b_tree10[[i]],b_tedata,type="class")
  b_error10[[i]]<-mean(b_predTest10[[i]] != b_telabels)
}

b_predTest100<-list()
b_error100<-list()
for(i in 1:1000){
  b_predTest100[[i]]<-predict(b_tree100[[i]],b_tedata,type="class")
  b_error100[[i]]<-mean(b_predTest100[[i]] != b_telabels)
}

#reporting missclassification results

b_me1<-mean(unlist(b_error1))
b_me10<-mean(unlist(b_error10))
b_me100<-mean(unlist(b_error100))

b_ve1<-var(unlist(b_error1))
b_ve10<-var(unlist(b_error10))
b_ve100<-var(unlist(b_error100))

b_me<-c(b_me1,b_me10,b_me100)
b_ve<-c(b_ve1,b_ve10,b_ve100)

b<-rbind(b_me,b_ve)
row.names(b)<-c("Mean","Variance")
colnames(b)<-c("1 tree","10 tree","100 trees")
b
#creating training data
x1<-runif(100)
x2<-runif(100)
trdata<-cbind(x1,x2)
y<-as.numeric((x1<0.5 & x2<0.5) | (x1>0.5 & x2>0.5))
trlabels<-as.factor(y)

#learn random forests with 1, 10, 100 trees to training data
c_tree1<-replicate(n=1000,
                   exp=randomForest(trlabels~.,data=trdata,importance=TRUE,
                                    ntree=1,nodesize=12,keep.forest = TRUE),
```

```
                    simplify=FALSE)

c_tree10<-replicate(n=1000,
                    exp=randomForest(trlabels~.,data=trdata,importance=TRUE,
                                     ntree=10,nodesize=12,keep.forest = TRUE),
                    simplify=FALSE)

c_tree100<-replicate(n=1000,
                     exp=randomForest(trlabels~.,data=trdata,importance=TRUE,
                                      ntree=100,nodesize=12,keep.forest = TRUE),
                     simplify=FALSE)
#producing test data
set.seed(1234)
x1<-runif(1000)
x2<-runif(1000)
c_tedata<-cbind(x1,x2)
y<-as.numeric((x1<0.5 & x2<0.5) | (x1>0.5 & x2>0.5))
c_telabels<-as.factor(y)
plot(x1,x2,col=(y+1),main = "Plot of test data")

#predicting on test data
c_predTest1<-list()
c_error1<-list()
for(i in 1:1000){
  c_predTest1[[i]]<-predict(c_tree1[[i]],c_tedata,type="class")
  c_error1[[i]]<-mean(c_predTest1[[i]] != c_telabels)
}

c_predTest10<-list()
c_error10<-list()
for(i in 1:1000){
  c_predTest10[[i]]<-predict(c_tree10[[i]],c_tedata,type="class")
  c_error10[[i]]<-mean(c_predTest10[[i]] != c_telabels)
}

c_predTest100<-list()
c_error100<-list()
for(i in 1:1000){
  c_predTest100[[i]]<-predict(c_tree100[[i]],c_tedata,type="class")
  c_error100[[i]]<-mean(c_predTest100[[i]] != c_telabels)
}

#reporting missclassification results

c_me1<-mean(unlist(c_error1))
c_me10<-mean(unlist(c_error10))
c_me100<-mean(unlist(c_error100))

c_ve1<-var(unlist(c_error1))
c_ve10<-var(unlist(c_error10))
c_ve100<-var(unlist(c_error100))

c_me<-c(c_me1,c_me10,c_me100)
```

```
c_ve<-c(c_ve1,c_ve10,c_ve100)

c<-rbind(c_me,c_ve)
row.names(c)<-c("Mean","Variance")
colnames(c)<-c("1 tree","10 tree","100 trees")
c
#means of error rate for each case
me<-rbind(a_me,b_me,c_me)
colnames(me)<-c("1","10","100")
row.names(me)<-c("Case a", "Case b","Case c")
me
#variance of error rate for each case
ve<-rbind(a_ve,b_ve,c_ve)
colnames(ve)<-c("1","10","100")
row.names(ve)<-c("Case a", "Case b","Case c")
ve

#rm(list=ls())
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=1000 # 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)

# Producing the training data
for(n in 1:N) {
  k <- sample(1:3,1,prob=true_pi)
  for(d in 1:D) {
    x[n,d] <- rbinom(1,1,true_mu[k,d])
  }
}

K=2 # 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 distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations

# Random initialization of the paramters
pi <- runif(K,0.49,0.51)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0.49,0.51)
}
```

```
normalize=function(xx){
  return(xx/sum(xx))
}
#y_{i,j}
clusterBelongingProbability=function(xx,mumu,NN,KK){
  #likelihood of belonging to a certain cluster times the probability of that cluster / all other likel
  notX = 1-xx

  lik1 = matrix(nrow=NN, ncol=KK)
  for(row in 1:KK){
    oneMinusMu=1-mumu[row,]
    x21=t(mumu[row,]*t(xx))
    x22=t(oneMinusMu*t(notX))
    x2=x21+x22

    lik1[,row]=apply(x2,1,prod)
  }
  pp=pi*lik1
  probs=apply(pp,1,normalize)

  return(probs)
}

newMuAndPi=function(xx,mumu,NN,KK){
  #mean is ML-estimator for p
  probs=clusterBelongingProbability(xx,mumu,NN,KK)

  newMu=matrix(nrow=KK, ncol=ncol(xx))
  newPi1=apply(probs,1,sum)
  newMu=probs%*%xx / apply(probs,1,sum)

  parameters=list(mu=newMu,pi=newPi1/N)
  return(parameters)
}
logLikelihoodGivenMuPi=function(xx,mumu,NN,KK){
  notX=1-xx
  probs=clusterBelongingProbability(xx,mumu,NN,KK)

  llikelihood=0
  for(row in 1:KK){
    oneMinusMu=1-mumu[row,]
    x21=t(mumu[row,]*t(xx))
    x22=t(oneMinusMu*t(notX))
    x2=x21 + x22

    x3=probs[row,]*log(x2)

    llikelihood=llikelihood + sum(x3)
  }
  return(llikelihood)
}
```

```
llik=rep(0,100)#max_it)
for(it in 1:100) {#max_it) {
  llik[it]=logLikelihoodGivenMuPi(x,mu,N,K)

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()

  if (it>1){
    if (abs(llik[it]-llik[it-1])<0.1){
      break
    }
  }

  params=newMuAndPi(x,mu,N,K)
  mu=params$mu
  pi=params$pi
}

  plot(mu[1,], type="o", col="blue", ylim=c(0,1),ylab="Probability",xlab="mu-value by index", main="K=2
  points(mu[2,], type="o", col="red")


#rm(list=ls())
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=1000 # 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)

# Producing the training data
for(n in 1:N) {
  k <- sample(1:3,1,prob=true_pi)
  for(d in 1:D) {
    x[n,d] <- rbinom(1,1,true_mu[k,d])
  }
}

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 distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations
```

```r
# Random initialization of the paramters
pi <- runif(K,0.49,0.51)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0.49,0.51)
}



normalize=function(xx){
  return(xx/sum(xx))
}
#y_{i,j}
clusterBelongingProbability=function(xx,mumu,NN,KK){
  #likelihood of belonging to a certain cluster times the probability of that cluster / all other likel
  notX = 1-xx

  lik1 = matrix(nrow=NN, ncol=KK)
  for(row in 1:KK){
    oneMinusMu=1-mumu[row,]
    x21=t(mumu[row,]*t(xx))
    x22=t(oneMinusMu*t(notX))
    x2=x21+x22

    lik1[,row]=apply(x2,1,prod)
  }
  pp=pi*lik1
  probs=apply(pp,1,normalize)

  return(probs)
}

newMuAndPi=function(xx,mumu,NN,KK){
  #mean is ML-estimator for p
  probs=clusterBelongingProbability(xx,mumu,NN,KK)

  newMu=matrix(nrow=KK, ncol=ncol(xx))
  newPi1=apply(probs,1,sum)
  newMu=probs%*%xx / apply(probs,1,sum)

  parameters=list(mu=newMu,pi=newPi1/N)
  return(parameters)
}
logLikelihoodGivenMuPi=function(xx,mumu,NN,KK){
  notX=1-xx
  probs=clusterBelongingProbability(xx,mumu,NN,KK)

  llikelihood=0
  for(row in 1:KK){
    oneMinusMu=1-mumu[row,]
    x21=t(mumu[row,]*t(xx))
    x22=t(oneMinusMu*t(notX))
    x2=x21 + x22
```

```
    x3=probs[row,]*log(x2)

    llikelihood=llikelihood + sum(x3)
  }
  return(llikelihood)
}



llik=rep(0,100)#max_it)
for(it in 1:100) {#max_it) {
  llik[it]=logLikelihoodGivenMuPi(x,mu,N,K)

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()

  if (it>1){
    if (abs(llik[it]-llik[it-1])<0.1){
      break
    }
  }

  params=newMuAndPi(x,mu,N,K)
  mu=params$mu
  pi=params$pi
}

  plot(mu[1,], type="o", col="blue", ylim=c(0,1),ylab="Probability",xlab="mu-value by index", main="K=3
  points(mu[2,], type="o", col="red")
  points(mu[3,], type="o", col="green")


#rm(list=ls())
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=1000 # 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)

# Producing the training data
for(n in 1:N) {
  k <- sample(1:3,1,prob=true_pi)
  for(d in 1:D) {
    x[n,d] <- rbinom(1,1,true_mu[k,d])
```

```
  }
}

K=4 # 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 distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations

# Random initialization of the paramters
pi <- runif(K,0.49,0.51)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0.49,0.51)
}



normalize=function(xx){
  return(xx/sum(xx))
}
#y_{i,j}
clusterBelongingProbability=function(xx,mumu,NN,KK){
  #likelihood of belonging to a certain cluster times the probability of that cluster / all other likel
  notX = 1-xx

  lik1 = matrix(nrow=NN, ncol=KK)
  for(row in 1:KK){
    oneMinusMu=1-mumu[row,]
    x21=t(mumu[row,]*t(xx))
    x22=t(oneMinusMu*t(notX))
    x2=x21+x22

    lik1[,row]=apply(x2,1,prod)
  }
  pp=pi*lik1
  probs=apply(pp,1,normalize)

  return(probs)
}

newMuAndPi=function(xx,mumu,NN,KK){
  #mean is ML-estimator for p
  probs=clusterBelongingProbability(xx,mumu,NN,KK)

  newMu=matrix(nrow=KK, ncol=ncol(xx))
  newPi1=apply(probs,1,sum)
  newMu=probs%*%xx / apply(probs,1,sum)

  parameters=list(mu=newMu,pi=newPi1/N)
  return(parameters)
}
logLikelihoodGivenMuPi=function(xx,mumu,NN,KK){
```

```
  notX=1-xx
  probs=clusterBelongingProbability(xx,mumu,NN,KK)

  llikelihood=0
  for(row in 1:KK){
    oneMinusMu=1-mumu[row,]
    x21=t(mumu[row,]*t(xx))
    x22=t(oneMinusMu*t(notX))
    x2=x21 + x22

    x3=probs[row,]*log(x2)

    llikelihood=llikelihood + sum(x3)
  }
  return(llikelihood)
}



llik=rep(0,100)#max_it)
for(it in 1:100) {#max_it) {
  llik[it]=logLikelihoodGivenMuPi(x,mu,N,K)

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()

  if (it>1){
    if (abs(llik[it]-llik[it-1])<0.1){
      break
    }
  }

  params=newMuAndPi(x,mu,N,K)
  mu=params$mu
  pi=params$pi
}

  plot(mu[1,], type="o", col="blue", ylim=c(0,1),ylab="Probability",xlab="mu-value by index", main="K=4
  points(mu[2,], type="o", col="red")
  points(mu[3,], type="o", col="green")
  points(mu[4,], type="o", col="yellow")


#rm(list=ls())
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=1000 # 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
```

```r
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")

# Producing the training data
for(n in 1:N) {
  k <- sample(1:3,1,prob=true_pi)
  for(d in 1:D) {
    x[n,d] <- rbinom(1,1,true_mu[k,d])
  }
}

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 distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations

# Random initialization of the paramters
pi <- runif(K,0.49,0.51)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0.49,0.51)
}
pi
mu




normalize=function(xx){
  return(xx/sum(xx))
}
#y_{i,j}
clusterBelongingProbability=function(xx,mumu,NN,KK){
  #likelihood of belonging to a certain cluster times the probability of that cluster / all other likel
  notX = 1-xx

  lik1 = matrix(nrow=NN, ncol=KK)
  for(row in 1:KK){
    oneMinusMu=1-mumu[row,]
    x21=t(mumu[row,]*t(xx))
    x22=t(oneMinusMu*t(notX))
    x2=x21+x22

    lik1[,row]=apply(x2,1,prod)
  }
```

```
  pp=pi*lik1
  probs=apply(pp,1,normalize)

  return(probs)
}

newMuAndPi=function(xx,mumu,NN,KK){
  #mean is ML-estimator for p
  probs=clusterBelongingProbability(xx,mumu,NN,KK)

  newMu=matrix(nrow=KK, ncol=ncol(xx))
  newPi1=apply(probs,1,sum)
  newMu=probs%*%xx / apply(probs,1,sum)

  parameters=list(mu=newMu,pi=newPi1/N)
  return(parameters)
}
logLikelihoodGivenMuPi=function(xx,mumu,NN,KK){
  notX=1-xx
  probs=clusterBelongingProbability(xx,mumu,NN,KK)

  llikelihood=0
  for(row in 1:KK){
    oneMinusMu=1-mumu[row,]
    x21=t(mumu[row,]*t(xx))
    x22=t(oneMinusMu*t(notX))
    x2=x21 + x22

    x3=probs[row,]*log(x2)

    llikelihood=llikelihood + sum(x3)
  }
  return(llikelihood)
}



llik=rep(0,100)#max_it)
for(it in 1:100) {#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="yellow")
  Sys.sleep(0.5)


  llik[it]=logLikelihoodGivenMuPi(x,mu,N,K)

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()

  if (it>1){
    if (abs(llik[it]-llik[it-1])<0.1){
```

```
      break
    }
  }

  params=newMuAndPi(x,mu,N,K)
  mu=params$mu
  pi=params$pi
}
pi
mu
plot(llik[1:it], type="o")

library(pamr)
library(kernlab)
library(glmnet)
library(sgof)
library(readr)
library(ggplot2)
geneexp = read_csv("geneexp.csv")
data <- geneexp[ ,-1]
set.seed(12345)
n <- dim(data)[1]
data$CellType <- as.factor(data$CellType)
ind   <- sample(1:n, floor(n*0.7))
train <- data[ind,]
test  <- data[-ind,]
#train
rownames(train)=1:nrow(train)
x_train = t(train[,-ncol(train)])
y_train = train[[ncol(train)]]
mytrain_data    <- list(x = x_train,
                        y = as.factor(y_train),
                        geneid    = as.character(1:nrow(x_train)),
                        genenames = colnames(train[,-ncol(train)]))

#test
rownames(test)=1:nrow(test)
x_test = t(test[ ,-ncol(test)])
y_test = test[[ncol(test)]]
# Cross Validation for the shrunken centroid
model = pamr.train(mytrain_data)
cvmodel = pamr.cv(model, mytrain_data)
pamr.plotcv(cvmodel)
best_threshold = cvmodel$threshold[which.min(cvmodel$error)]
cat(paste("The optimal threshold : ", best_threshold))

# Training a model with the best threshold from the cross validation
pamr.plotcen(model, mytrain_data, threshold = best_threshold)
features_nsc = pamr.listgenes(model,mytrain_data,threshold=best_threshold)
cat(paste("Number of genes selected : ", nrow(features_nsc)))
feature = colnames(data)[as.numeric(features_nsc[,1])][1:2]
feature
pred_nsc = pamr.predict(model,
```

```
                          newx = x_test,
                          threshold = best_threshold)
tab_nsc = table(y_test, pred_nsc)
mis_nsc  <- 1 - sum(diag(tab_nsc)) / sum(tab_nsc)
cat(paste("Test Error:", round(mis_nsc,5)))
res_nsc <- list("Test Error" = round(mis_nsc,5), "Features Selected" = nrow(features_nsc))
elastic_net = cv.glmnet(x = t(x_train),
                        y = y_train,
                        family = "multinomial",
                        alpha = 0.5)
# create prediction
pred_elastic = predict(elastic_net,
                       newx = t(x_test),
                       type = "class",
                       s = elastic_net$lambda.min)
# s penalty parameter
tab_elastic  = table(y_test, pred_elastic)
mis_elastic <- 1 - (sum(diag(tab_elastic))/sum(tab_elastic))
min_lambda = elastic_net$lambda.min
features_elastic = as.numeric(elastic_net$nzero[which(elastic_net$lambda==min_lambda)])
res_elastic <- list("Error Rate" = round(mis_elastic,5), "Features Selected" = features_elastic)
ign <- capture.output({
svm_fit  = ksvm(x = t(x_train),
                y = y_train,
                kernel = "vanilladot")
})
pred_svm = predict(svm_fit,
                   newdata = t(x_test),
                   type = "response")

tab_svm = table(y_test, pred_svm)

mis_svm = 1 - (sum(diag(tab_svm))/sum(tab_svm))
features_svm = ncol(data)
res_svm <- list("Error Rate" = round(mis_svm,5), "Features Selected" = features_svm)
result <- rbind("NSC" = res_nsc, "Elastic Net" = res_elastic, "SVM" = res_svm)

# print the test error rates and contributing features for all models
knitr::kable(result, caption = "Comparsion of Models on Test dataset")

BH = function(cell_name){
y <- ifelse(data$CellType == cell_name, 1, 0)
df <- data.frame(name=character(),pvalue=numeric())
for (i in 1:(ncol(data)-1)){
  test <- t.test(unlist(data[,i])~y, data = data, alternative = "two.sided")$p.value
  test_df = data.frame(name=colnames(data)[i],pvalue=test)
  df <- rbind(df, test_df)
 }
df<-df[order(as.numeric(df$pvalue)),]
  M = ncol(data)-1
  alpha = 0.05
  for (i in 1:M) {
    if(as.numeric(df$pvalue[i]) > (alpha * i/M)){
```

```
      break
    }
  }
  L = i-1
  total <<- rbind(total,c(cell_name,L))
  p <- as.numeric(df$pvalue[L])
  rejected <- ifelse(as.numeric(df$pvalue) <= p,1,0)

  row.names(df) = c(1:length(df$pvalue))

  ggplot() +
  ylab("P-Value") + xlab("Index") + ggtitle(cell_name) +
  geom_point(data=data.frame(x=c(1:L),
                             y=df$pvalue[1:L]),
             aes(x=x, y=y), col="red") +
  geom_point(data=data.frame(x=c(L+1:M),y=df$pvalue[L+1:M]),
             aes(x=x, y=y), col="blue") +
  scale_y_continuous(limits=c(0,1)) + scale_x_continuous(limits = c(0, M))
}

total = data.frame(name=character(),total = numeric())

BH("CD4")
BH("CD8")
BH("CD19")

colnames(total) = c("Cell", "Rejected")
knitr::kable(total, caption = "Genes correspond to the rejected hypothesis for each cell type")
```