Machine Learning: Computer Lab Block 2

Siddharth Saminathan, Salvador Marti Roman, Nicolas Taba

01/12/2020

Statement of contribution

Siddharth Saminathan solved assignment 1, Salvador Marti Roman solved assignment 2 and Nicolas Taba solved assignment 3. All members of the group discussed solutions and problems during all the assignment period.

Assignment 1: Ensemble Methods

Misclassification Errors for all Models with 1,10,100 Trees

```
##          NTree = 1 NTree = 10 NTree = 100
## Accuracy    81.200     85.800      88.800
## error        0.188      0.142       0.112
```

a.

Creating 1000 Training Datasets

```
## 'data.frame':    100000 obs. of  3 variables:
##  $ x1: num  0.2245 0.993 0.9293 0.0486 0.8403 ...
##  $ x2: num  0.245 0.841 0.891 0.727 0.852 ...
##  $ y : Factor w/ 2 levels "0","1": 2 1 1 2 2 1 2 2 1 2 ...
```

Misclassification Errors for x1<x2

```
##              Ntrre=1     NTree=10    Ntree=100
## Mean     0.208890000 0.1399920000 0.1139040000
## Variance 0.002968615 0.0009705405 0.0009491419
```

b.

Misclassification Errors for x1<0.5

```
##             Ntrre=1     NTree=10   Ntree=100
## Mean     0.09006400 0.0163450000 6.12200e-03
## Variance 0.01640528 0.0008395715 5.05857e-05
```

c.

Misclassification Errors for x1<0.5&x2<0.5

```
##             Ntrre=1    NTree=10   Ntree=100
## Mean     0.24627100 0.117753000 0.073424000
## Variance 0.01264449 0.002855435 0.001242287
```

d

a. As the number of trees in the random forest grows we can expect the mean and variance to drop.This is because the randomness is minimized by constructing more trees. Irrespective of the 1000 data sets we used to train the model our variance and mean significantly drop as the number of trees increases. This shows that as the number of trees increase the randomness or size of the training data does not affect the ROC or AUC of the Random Forest. However the change or decrease in error rate remains the same after a specific value of ntree. This is because there is an increase in computational cost after a certain number of tress hence the change in error rate is negligible.

b The results obtained for the third statement x1<0.5&&x2<0.5 gives us a better performance despite of being complex. This is true for when ntree=10 , ntree=100. This shows that the training data does not affect our predictions for higher number of trees in the random forest. However, if we set the node size to 25 instead of 12 our performance drops significantly. Reducing the node size to 12 simply means that there are lesser samples at each node for the model to learn from.This means that lesser randomness in OOB. The OOB ensmeble converges quickly resulting in lesser run time and lower tree correlation, thus performing better within an effective runtime.

c. Smaller variances in error is important because it minimizes the overall error.The smaller the variance of error the lesser the overall error will be.

Assignment 2. Mixture models

Your task is to implement the EM algorithm for mixtures of multivariate Bernoulli distributions. Please use the R template below to solve the assignment. Then, use your implementation to show what happens when your mixture model has too few and too many components, i.e. set K=2,3,4 and compare results. Please provide a short explanation as well.

E-step: For a mixture of multivariate Bernoulli distributions, z is calculated as follows.

$$z = \frac{\pi_k * p(x)}{\sum pi_k * Bernoulli(x \mid \mu_k)}$$

$$Bernoulli(x \mid \mu_k) = \prod_i \mu_{ki}^{x_i}(1-\mu_{ki})^{1-x_i}$$

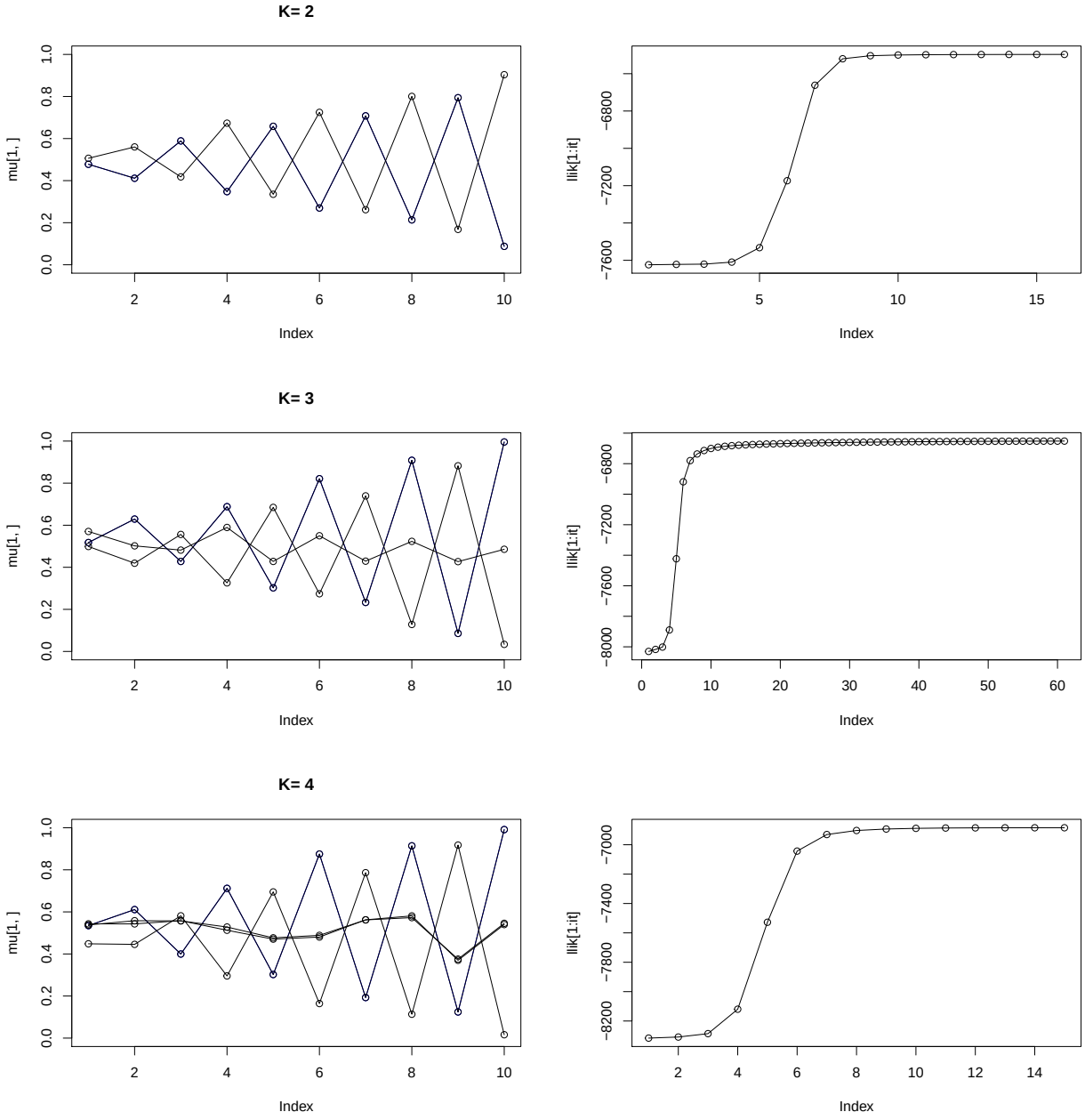
The log likelihood function is for a given sample x_n, k_n of size N is:

$$\sum_n \sum_k z_{nk}[\log \pi_k + \sum_i [x_{ni} \log \mu_{ki} + (1-x_{ni}) \log(1-\mu_{ki})]]$$

M-step: After setting to zero the derivatives and taking into account the constraints the updated parameters are:

$$\pi_k^{ML} = \frac{\sum_n z_{nk}}{N}$$

$$\mu_{ki}^{ML} = \frac{\sum_n z_{nk} x_{ni}}{\sum_n znk}$$



Results

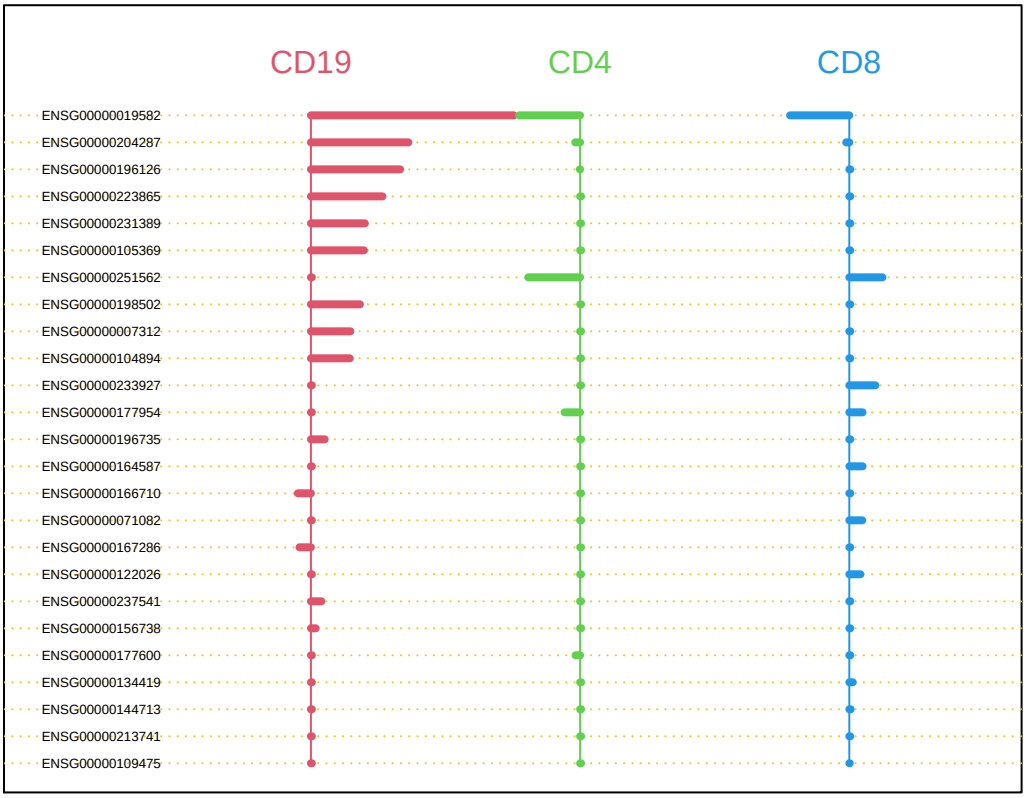
K	Iterations until convergence	Log-Likelihood
2	**16**	**-6496.662**
3	61	-6652.034
4	15	-6684.276
5	100+	-6811.211

When the mixture model has too many components the Log-Likelihood not only decreases but it can also take much longer to converge. For instance for K=5 the EM fails to converge in less than 100 iterations.

The poor results obtained in spite of the very clean generated data can be attributed to the EM algorithm's propensity to get stuck in local optima.

Assignment 3

Question 1



25 features were selected using the nearest shrunk centroid method with a threshold value of **7.3**. The positive and negative values represent how far the centroids and the data is from a given class compared to other classes (in this case, this is the cell types). A positive centroid plot denotes a positive contribution of the gene for the class whereas a negative value denotes a low contribution of the gene when it is expressed. The values cannot be all positive for a gene in all centroid plots because that would mean that all the values of this gene would be far away from the centroids. The shrinkage step of the algorithm (the threshold reduction towards the mean) would reduce the contribution of this gene to 0 and eliminate it from the final plot.

Question 2

```
##      CD19 CD4 CD8 Class Error rate
## CD19   66   7   0       0.09589041
## CD4     0  66   4       0.05714286
## CD8     0   8  59       0.11940299
## Overall error rate= 0.09
```

The two genes are that contribute the most are :ENSG00000019582 ENSG00000204287. These genes have as alternative names: CD74 and HLA-DRA. Both these genes are marker genes for B-cells, luminal epithelial cells and macrophages.

Question 3

```
##
## Confusion matrix for the elastic net method
```

```
##            elastic_net_pred
## test_target CD19 CD4 CD8
##        CD19   27   0   0
##        CD4     0  29   1
##        CD8     1   3  29
```

```
##  Setting default kernel parameters
```

```
##
## Condusion matrix for the SVM method
```

```
##            model_svm
## test_target CD19 CD4 CD8
##        CD19   27   0   0
##        CD4     0  30   0
##        CD8     0   2  31
```

Method	Missclassification errors	Nbr. features
NSC	0.09	25
EN	0.056	54
SVM	0.022	74

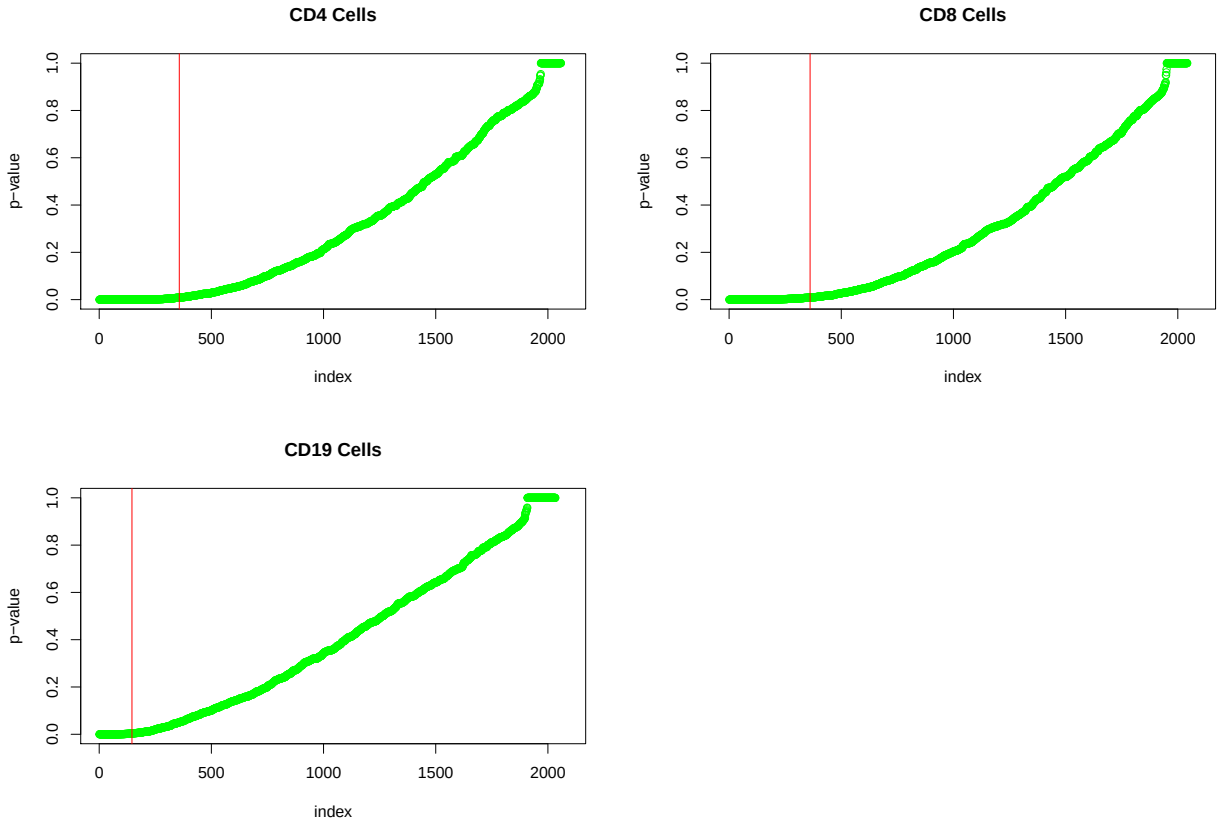
In terms of accuracy, we might be tempted to choose the support vector machine method as it has the lowest error rate (2% compared to respectively 9% and 5.6% for the NSC and Elastic net methods).However, we might want to be careful as this method also makes use of more features (higher dimensionality) than other methods. Having reduced the amount of features from ~2000 to 74 should be our best option given if computation is cheap.

Question 4

```
## The number of rejected genes for the CD4 cells is:  357
```

```
## The number of rejected genes for the CD8 cells is:  361
```

```
## The number of rejected genes for the CD19 cells is:  146
```



Our null-hypothesis is that some of these gene expressions do not contribute to the other two cell types. For each of the presented graphs, we reject the null-hypothesis for the p-values of the genes that are located to the left of the threshold value marked by the red line. It can also be viewed as the p-values below the line α/M where α is our confidence and M is the number of genes selected. These gene expressions are significant and contribute to the other two cell type.

Appendix: All code for this report

```
knitr::opts_chunk$set(echo = TRUE, warning=FALSE, message = FALSE)

library(ggplot2)
library(randomForest)

#training Data
set.seed(12345)


x1<-runif(100)
x2<-runif(100)
trdata<-cbind(x1,x2)
y<-as.numeric(x1<x2)
trlabels<-as.factor(y)
final_data_train<-as.data.frame(cbind(trdata,trlabels))


set.seed(12345)

x1<-runif(1000)
x2<-runif(1000)
tedata<-cbind(x1,x2)
y<-as.numeric(x1<x2)
telabels<-as.factor(y)
final_data_test<-as.data.frame(cbind(tedata,telabels))
#summary(final_data_test)
final_data_test$telabels<-as.factor(final_data_test$telabels)
#Training Model with 1 Tree
model_1<-randomForest(factor(trlabels)~.,data=final_data_train,ntree=1,nodesize=25,keep.forest=TRUE)
#Training Model with 10 Tree
model_10<-randomForest(factor(trlabels)~.,data=final_data_train,ntree=10,nodesize=25,keep.forest=TRUE)
#Training Model with 100 Tree
model_100<-randomForest(factor(trlabels)~.,data=final_data_train,ntree=100,nodesize=25,keep.forest=TRUE)
#Predicting Model with 1 Tree
pred_1<-predict(model_1,newdata=final_data_test)
#Predicting Model with 10 Tree
pred_2<-predict(model_10,newdata=final_data_test)
#Predicting Model with 100 Tree
pred_3<-predict(model_100,newdata=final_data_test,type="class")
confm_1<-table(obs=final_data_test[,3],pred=pred_1)

confm_10<-table(obs=final_data_test[,3],pred=pred_2)

confm_100<-table(obs=final_data_test[,3],pred=pred_3)

Accuracy<-(sum(diag(confm_1))/sum(confm_1))*100
error<-1-sum(diag(confm_1))/sum(confm_1)

Accuracy10<-(sum(diag(confm_10))/sum(confm_10))*100
error10<-1-sum(diag(confm_10))/sum(confm_10)
```

```
Accuracy100<-(sum(diag(confm_100))/sum(confm_100))*100
error100<-1-sum(diag(confm_100))/sum(confm_100)

misclassification1<-as.data.frame(rbind(Accuracy,error))

misclassification10<-as.data.frame(rbind(Accuracy10,error10))

misclassification100<-as.data.frame(rbind(Accuracy100,error100))

final_error<-cbind(misclassification1,misclassification10,misclassification100)
colnames(final_error)[1]="NTree = 1"
colnames(final_error)[2]="NTree = 10"
colnames(final_error)[3]="NTree = 100"

final_error

#Creating 1000 Training Datasets

x1_train<-(replicate(n = 1000,
          expr =runif(100), simplify = FALSE
))

x2_train<-(replicate(n = 1000,
              expr =runif(100), simplify = FALSE
               ))

y<-as.numeric(unlist(x1_train)<unlist(x2_train))
val<-as.data.frame(cbind(unlist(x1_train),unlist(x2_train)))

final_val<-as.data.frame(cbind(val,y))
colnames(final_val)[1]="x1"
colnames(final_val)[2]="x2"
final_val$y<-as.factor(final_val$y)
str(final_val)
#summary(final_val)
#Training 1000 Data Sets

#Function to train Models
 start=1
end=100
RF_1<-function(x,y,n)
{

 final_train_1<-list()

  for(i in 1:1000)
{
final_train_1[[i]]<-randomForest(factor(x$y[start:end])~.,data=x[start:end,],ntree=n,nodesize=25,keep.f



i=i+1
start=end
end=end+100
```

```
}
return(final_train_1)
}

#Function to Predict test Data

  RF_pred<-function(x,y)
  {
      pred_1000_1<-predict(x,newdata=y)
     return(pred_1000_1)
  }


 #Function to calculate Confusion Matrix of Random Forest

 confusionmatrix<-function(x,y)
 {confm<-list()
  for(i in 1:1000)
 {
 confm[[i]]<-table(obs=x[,3],pred=unlist(y[i]))
 i=i+1

  }
 return(confm)

 }

#Function to calculate Misclassification Errors

 error<-function(x)
 {errorval=list()
 for(i in 1:1000)
{
errorval[[i]]<-1-sum(diag(as.matrix(unlist(x[[i]]))))/sum(unlist(x[i]))
i=i+1
}
 return(errorval)
 }



#Training
#NTree=1
errors_tree_1<-RF_1(x=final_val,y=x$y,n=1)
#Ntree=10
errors_tree_10<-RF_1(x=final_val,y=x$y,n=10)
#NTree=100
errors_tree_100<-RF_1(x=final_val,y=x$y,n=100)


#Predicting
#NTree=1
 pred_val_1<-list()
```

```
  for (i in 1:1000 ) {

    pred_val_1[[i]]<-RF_pred(x=errors_tree_1[[i]],final_data_test)
    i=i+1

  }
 #NTree=10
 pred_val_10<-list()
 for (i in 1:1000 ) {

  pred_val_10[[i]]<-RF_pred(x=errors_tree_10[[i]],final_data_test)
  i=i+1

 }
 #NTree=100
 pred_val_100<-list()
for (i in 1:1000 ) {

  pred_val_100[[i]]<-RF_pred(x=errors_tree_100[[i]],final_data_test)
  i=i+1

}

 #errors
 #NTree=1
 confm_1000_1<-confusionmatrix(x=final_data_test,pred_val_1)
 error_1000_1<-error(confm_1000_1)
 #NTree=10
 confm_1000_10<-confusionmatrix(final_data_test,pred_val_10)
 error_1000_10<-error(confm_1000_10)
#NTree=100
 confm_1000_100<-confusionmatrix(final_data_test,pred_val_100)
 error_1000_100<-error(confm_1000_100)

#Mean and Variance of errors
 final_mean_error<-cbind(mean(unlist(error_1000_1)),mean(unlist(error_1000_10)),mean(unlist(error_1000_

final_variance_error<-cbind(var(unlist(error_1000_1)),var(unlist(error_1000_10)),var(unlist(error_1000_

final_error<-as.data.frame(rbind(final_mean_error,final_variance_error))
colnames(final_error)=c("Ntrre=1","NTree=10","Ntree=100")
rownames(final_error)=c("Mean","Variance")
(final_error)

#Generating Train Data for X1<0.5

y_b<-as.factor(as.numeric(unlist(x1_train)<0.5))
final_val_b<-as.data.frame(cbind(val,y_b))
colnames(final_val_b)[1]="x1"
colnames(final_val_b)[2]="x2"
#summary(final_val_b)
```

```r
#Generating Test Data for X1<0.5

set.seed(12345)
y_b<-as.factor(as.numeric(x1<0.5))
telabels_b<-as.factor(y_b)
final_data_test_b<-as.data.frame(cbind(tedata,y_b))
final_data_test_b$y_b=factor(y_b ,labels = c(0,1))
#summary(final_data_test_b)

# Function for Training Model
start=1
end=100
RF_1_b<-function(x,n)
{
  final_train_1<-list()

  for(i in 1:1000)
  {
    final_train_1[[i]]<-randomForest((x$y_b[start:end])~.,data=x[start:end,],ntree=n,nodesize=25,keep.f


    i=i+1
    start=end
    end=end+100
  }
  return(final_train_1)
}

#Training
#Ntree=1
errors_tree_1_b<-RF_1_b(x=final_val_b,n=1)
#Ntree=10
errors_tree_10_b<-RF_1_b(x=final_val_b,n=10)
#NTree=100
errors_tree_100_b<-RF_1_b(x=final_val_b,n=100)

#Predicting
#NTree=1
pred_val_1_b<-list()
for (i in 1:1000 ) {

  pred_val_1_b[[i]]<-RF_pred(x=errors_tree_1_b[[i]],final_data_test_b)
  i=i+1

}
#NTree=10
pred_val_10_b<-list()
for (i in 1:1000 ) {

  pred_val_10_b[[i]]<-RF_pred(x=errors_tree_10_b[[i]],final_data_test_b)
  i=i+1

}
```

```
#NTree=100
pred_val_100_b<-list()
for (i in 1:1000 ) {

  pred_val_100_b[[i]]<-RF_pred(x=errors_tree_100_b[[i]],final_data_test_b)
  i=i+1

}

#errors
#NTree=1
confm_1000_1_b<-confusionmatrix(x=final_data_test_b,pred_val_1_b)
error_1000_1_b<-error(confm_1000_1_b)
#NTree=10
confm_1000_10_b<-confusionmatrix(final_data_test_b,pred_val_10_b)
error_1000_10_b<-error(confm_1000_10_b)
#NTree=100
confm_1000_100_b<-confusionmatrix(final_data_test_b,pred_val_100_b)
error_1000_100_b<-error(confm_1000_100_b)

#Mean and Variance of errors
final_mean_error_b<-cbind(mean(unlist(error_1000_1_b)),mean(unlist(error_1000_10_b)),mean(unlist(error_

final_variance_error_b<-cbind(var(unlist(error_1000_1_b)),var(unlist(error_1000_10_b)),var(unlist(error

final_error_b<-as.data.frame(rbind(final_mean_error_b,final_variance_error_b))
colnames(final_error_b)=c("Ntrre=1","NTree=10","Ntree=100")
rownames(final_error_b)=c("Mean","Variance")
(final_error_b)
{

# Creating Training Data for x1<0.5&&x2<0.5

y_c<-as.factor(as.numeric((unlist(x1_train)<0.5)&(unlist(x2_train)<0.5)|(unlist(x1_train)>0.5)&(unlist(
final_val_c<-as.data.frame(cbind(val,y_c))
colnames(final_val_c)[1]="x1"
colnames(final_val_c)[2]="x2"
#summary(final_val_c)
}

{
  # Creating Test Data for x1<0.5&&x2<0.5

  set.seed(12345)
y_c<-as.factor(as.numeric((x1<0.5)&(x2<0.5)|(x1>0.5)&(x2>0.5)))
telabels_c<-as.factor(y_c)
final_data_test_c<-as.data.frame(cbind(tedata,y_c))
final_data_test_c$y_c = factor(y_c,labels = c(0,1))
#summary(final_data_test_c)
}

#Function to train model
```

```
RF_1_c<-function(x,n)
{
  final_train_1<-list()

  for(i in 1:1000)
  {
    final_train_1[[i]]<-randomForest(factor(x$y_c[start:end])~.,data=x[start:end,],ntree=n,nodesize=12,


    i=i+1
    start=end
    end=end+100
  }
  return(final_train_1)
}

#Training
#Ntree=1
errors_tree_1_c<-RF_1_c(x=final_val_c,n=1)
#Ntree=10
errors_tree_10_c<-RF_1_c(x=final_val_c,n=10)
#NTree=100
errors_tree_100_c<-RF_1_c(x=final_val_c,n=100)


#Predicting
#NTree=1
pred_val_1_c<-list()
for (i in 1:1000 ) {

  pred_val_1_c[[i]]<-RF_pred(x=errors_tree_1_c[[i]],final_data_test_c)
  i=i+1

}
#NTree=10
pred_val_10_c<-list()
for (i in 1:1000 ) {

  pred_val_10_c[[i]]<-RF_pred(x=errors_tree_10_c[[i]],final_data_test_c)
  i=i+1

}
#NTree=100
pred_val_100_c<-list()
for (i in 1:1000 ) {

  pred_val_100_c[[i]]<-RF_pred(x=errors_tree_100_c[[i]],final_data_test_c)
  i=i+1

}

#errors
#NTree=1
```

```r
confm_1000_1_c<-confusionmatrix(x=final_data_test_c,pred_val_1_c)
error_1000_1_c<-error(confm_1000_1_c)
#NTree=10
confm_1000_10_c<-confusionmatrix(final_data_test_c,pred_val_10_c)
error_1000_10_c<-error(confm_1000_10_c)
#NTree=100
confm_1000_100_c<-confusionmatrix(final_data_test_c,pred_val_100_c)
error_1000_100_c<-error(confm_1000_100_c)

#Mean and Variance of error
final_mean_error_c<-cbind(mean(unlist(error_1000_1_c)),mean(unlist(error_1000_10_c)),mean(unlist(error_

final_variance_error_c<-cbind(var(unlist(error_1000_1_c)),var(unlist(error_1000_10_c)),var(unlist(error

final_error_c<-as.data.frame(rbind(final_mean_error_c,final_variance_error_c))
colnames(final_error_c)=c("Ntrre=1","NTree=10","Ntree=100")
rownames(final_error_c)=c("Mean","Variance")
(final_error_c)


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

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")

bern_em = function(K){
  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



  # 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= K # 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 parameters
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 # Random vector of size K proportions
#mu # random matrix of size KxD

for(it in 1:max_it) {
    # E-step: Computation of the fractional component assignments
    # Your code here
    p_x = c()
    #For every K
    for(k in 1:K){
      p_x_n = c()
      #For every datapoint
      for(n in 1:N){
        berk_k_n = (mu[k,]**x[n,]) * ((1-mu[k,])**(1-x[n,]))
        p_x_n = cbind(p_x_n,pi[k]*prod(berk_k_n))
      }
      p_x = cbind(p_x, colSums(p_x_n))
    }
    z = (pi*p_x)/rowSums(pi*p_x)

    #Log likelihood computation.
    # Your code here
    components = c()
    for(n in 1:N){
      for(k in 1:K){
        likelihood = log(pi[k]) + sum((x[n,] * log(mu[k,])) + ((1-x[n,]) * log(1-mu[k,])))
        proportion = z[n,k]
        components = c(components, proportion * likelihood)
      }
    }
    llik[it] = sum(components)

    #cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
    flush.console()
    # Stop if the lok likelihood has not changed significantly
    # Your code here
    if(isTRUE(abs(llik[it] - llik[it-1])< min_change)){break}
    #M-step: ML parameter estimation from the data and fractional component assignments
    ## adjust parameters to fit points assigned to them
    # Your code here

    #for all k do this
```

```
      for(k in 1:K){
        pi[k] = sum(z[,k]/N)
      }
      # for all k, i do this
      for(k in 1:K){
        for(i in 1:D){
          mu[k,i] = sum(z[,k]*x[,i])/sum(z[,k])
        }
      }
  }
  #pi
  #mu
  plot(mu[1,], type="o", col="blue", ylim=c(0,1),main=paste("K=",as.character(K)))
  for(i in 1:K){
    points(mu[i,], type="o")
  }

  plot(llik[1:it], type="o")
}

bern_em(2)
bern_em(3)
bern_em(4)

# rm(list = ls())
library(pamr)
library(glmnet)
library(kernlab)
setwd(dir = "~/732A99_MachineLearning")
data <- read.csv("geneexp.csv", header = TRUE, row.names = 1, sep = ",")


# Missclassification rate function
missclass <- function(x, x1){
  return(1-(sum(diag(table(x, x1)))/sum(table(x,x1))))
}


# Split the data 70/30 train/test
n <- dim(data)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.7))
train_set <- data[id,]
test_set <- data[-id,]

#### DATA###
# Training data
rownames(train_set) <- 1:nrow(train_set)
train_features <- t(train_set[,-2086]) #transpose because genetic data is handled in this way by pamr
train_target <- train_set[[2086]]
df_train <- list(x = train_features, y = as.factor(train_target), geneid = as.character(1:nrow(train_fe
#Test data
rownames(test_set) <- 1:nrow(test_set)
```

```
test_features <- t(test_set[,-2086])
test_target <- test_set[[2086]]
df_test <- list(x = test_features, y = as.factor(test_target), geneid = as.character(1:nrow(test_feature
#############

# Nearest shrunken centroid threshold chosen by cross-validation
model <- pamr.train(df_train, threshold = seq(0,30,0.1))
cvmodel <- pamr.cv(model, df_train)
# cv_plot <- pamr.plotcv(cvmodel)  # Bad output of the cross validation plot in the pdf
# print(cvmodel)

# Best threshold
optimal_threshold <- cvmodel$threshold[which.min(cvmodel$error)]
# print(optimal_threshold)


pamr.plotcen(model, df_train, threshold = optimal_threshold)
# Features selected
features_selected <- pamr.listgenes(model, df_train,threshold = optimal_threshold)
number_features <- nrow(features_selected)
# print(number_features)

# Pick 2 biggest contributing features:
best_2 <- as.matrix(colnames(data)[as.numeric(features_selected[,1])][1:2])
# print(best_2)


# test error nearest shrunk
test_error_nearestshrank <- pamr.confusion(model, optimal_threshold, extra=TRUE)

t_train_features <- t(train_features)
t_test_features <- t(test_features)

#elastic net fit
elastic_net <- cv.glmnet(x = t_train_features,
                         y = train_target,
                         family = "multinomial",
                         alpha = 0.5)

#Best lambda
min_lambda <- elastic_net$lambda.min
features_selected_elastic <- elastic_net$nzero[which(elastic_net$lambda == min_lambda)]

elastic_net_pred <- predict(elastic_net,
                            t_test_features,
                            type = "class",
                            s = min_lambda)

#penalty parameter
cm_elastic_net <- table(test_target, elastic_net_pred)
cat("\nConfusion matrix for the elastic net method\n")
print(cm_elastic_net)
error_elastic_net <- missclass(test_target, elastic_net_pred)
```

```r
## SUpport vector machines

svm <- ksvm(x = t_train_features, y = as.factor(train_target), kernel = "vanilladot", scale = FALSE)
features_selected_svm <- svm@nSV

model_svm <- predict(svm, t_test_features, type = "response")
cm_svm <- table(test_target, model_svm)
cat("\nCondusion matrix for the SVM method\n")
print(cm_svm)
error_svm <- missclass(test_target, model_svm)



#Comparing table
ct <- data.frame(Missclass. = c(" 0.09",
                                round(error_elastic_net, digits = 3),
                                round(error_svm, digits = 3)),
                 Features = c(number_features,
                              features_selected_elastic,
                              features_selected_svm),
                 row.names = c("NSC",
                               "Elastic Net",
                               "SVM"))


# print(ct)

## Benjamini-Hochberg
######## Prepare data #############
#CD4
CD4_indices <- which(data[[2086]] == "CD4")
CD4_data <-data[CD4_indices,]
CD4_not_data <- data[-CD4_indices, ]
CD4_data_form <- colnames(data[CD4_indices,- 2086])
#CD8
CD8_indices <- which(data[[2086]] == "CD8")
CD8_data <- data[CD8_indices,]
CD8_not_data <- data[-CD8_indices, ]
CD8_data_form <- colnames(data[CD8_indices, -2086])
#CD19
CD19_indices <- which(data[[2086]] == "CD19")
CD19_data <- data[CD19_indices,]
CD19_not_data <- data[-CD19_indices, ]
CD19_data_form <- colnames(data[CD19_indices, -2086])
###################################

# Calculate p-values for each set: "each cell type versus the remaining ones".
CD4_p_values <- sapply(CD4_not_data[-2086], function(x) t.test(x ~ CD4_not_data$CellType)$p.value)
CD8_p_values <- sapply(CD8_not_data[-2086], function(x) t.test(x ~ CD8_not_data$CellType)$p.value)
CD19_p_values <- sapply(CD19_not_data[-2086], function(x) t.test(x ~ CD19_not_data$CellType)$p.value)
#sort p-values
```

```r
CD4_sorted <- CD4_p_values[order(unlist(CD4_p_values), decreasing = FALSE)]
CD8_sorted <- CD8_p_values[order(unlist(CD8_p_values), decreasing = FALSE)]
CD19_sorted <- CD19_p_values[order(unlist(CD19_p_values), decreasing = FALSE)]

# Function to find the cutoff value for rejection
benjamini <- function(ordered){
  M <- length(ordered)
  alpha <- 0.05
  for(i in 1:M){
    if(is.nan(ordered[i]) == TRUE){break} #Deal with NaN values
    else if(ordered[i] < alpha*(i/M)){
      max_i <- i
    }
  }
  return(max_i)
}

#calculate how many genes get rejected
BH_CD4 <- benjamini(CD4_sorted)
BH_CD8 <- benjamini(CD8_sorted)
BH_CD19 <- benjamini(CD19_sorted)
cat("The number of rejected genes for the CD4 cells is: ", BH_CD4, "\n")
cat("The number of rejected genes for the CD8 cells is: ", BH_CD8, "\n")
cat("The number of rejected genes for the CD19 cells is: ", BH_CD19, "\n")



plot1 <- plot(x = 1:length(CD4_sorted), y = CD4_sorted, col="green", ylab = "p-value", xlab = "index", m
abline(abline(v = BH_CD4, col="red"))

plot2 <- plot(x = 1:length(CD8_sorted), y = CD8_sorted, col="green", ylab = "p-value", xlab = "index", m
abline(abline(v = BH_CD8, col="red"))

plot3 <- plot(x = 1:length(CD19_sorted), y = CD19_sorted, col="green", ylab = "p-value", xlab = "index"
abline(abline(v = BH_CD19, col="red"))
```