Computer Lab Block 2

Linköping University

Jakob Fjellström
Mowniesh Asokan
Tim Yuki Washio

November 30, 2020

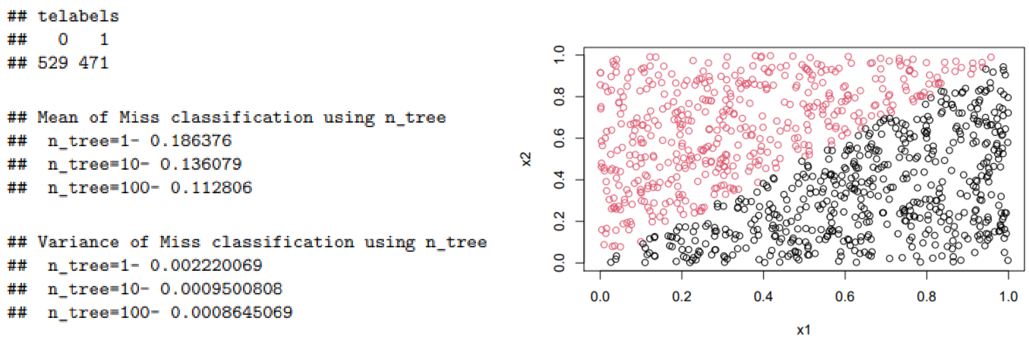
Mowniesh Worked on assignment 1, Jakob on assignment 2, and Yuki on assignment 3.

Assignment 1: Ensemble Methods

Working of random forest.

When the training set for the current tree is drawn by sampling with replacement, about one-third of the cases are left out of the sample. This oob (out-of-bag) data is used to get a running unbiased estimate of the classification error as trees are added to the forest. It is also used to get estimates of variable importance.
After each tree is built, all of the data are run down the tree, and proximities are computed for each pair of cases. If two cases occupy the same terminal node, their proximity is increased by one. At the end of the run, the proximities are normalized by dividing by the number of trees. Proximities are used in replacing missing data, locating outliers, and producing illuminating low-dimensional views of the data.

a.create 1000 training datasets of size 100, learn a random forest from each dataset, and compute the misclassification error in the same test dataset of size 1000. Report results for when the random forest has 1, 10 and 100 trees.
Random forest model can evaluated by using the data x1 and x2 ,where target y=x1 < x2

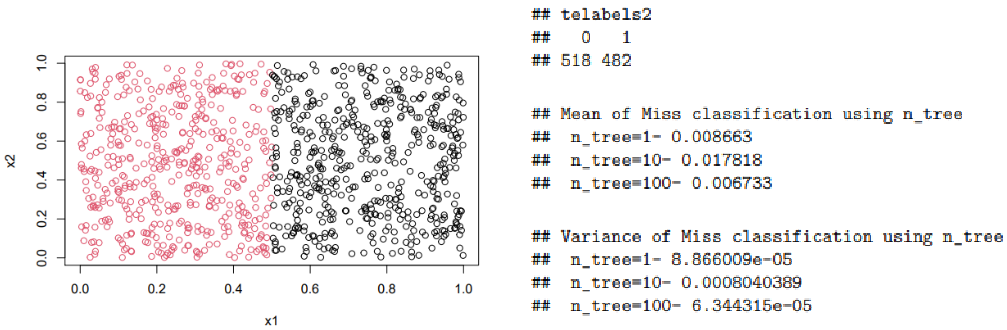


```
## telabels
##   0   1
## 529 471

## Mean of Miss classification using n_tree
##  n_tree=1- 0.186376
##  n_tree=10- 0.136079
##  n_tree=100- 0.112806

## Variance of Miss classification using n_tree
##  n_tree=1- 0.002220069
##  n_tree=10- 0.0009500808
##  n_tree=100- 0.0008645069
```

(a) (b)

Figure 1

b. Repeat the exercise above but this time use the condition (x1 < 0.5) instead of (x1 < x2) when producing the training and test datasets.



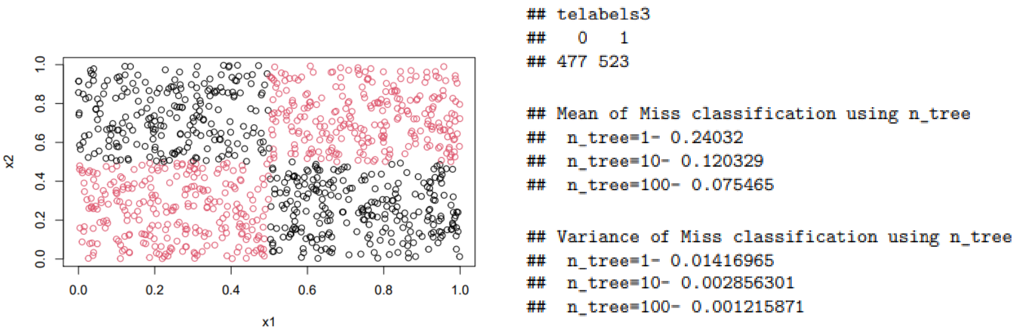
(a)

```
## telabels2
##   0   1
## 518 482

## Mean of Miss classification using n_tree
##  n_tree=1- 0.008663
##  n_tree=10- 0.017818
##  n_tree=100- 0.006733

## Variance of Miss classification using n_tree
##  n_tree=1- 8.866009e-05
##  n_tree=10- 0.0008040389
##  n_tree=100- 6.344315e-05
```

(b)

Figure 2

c. Repeat the exercise above but this time use the condition ((x1 < 0.5   x2 < 0.5) | (x1 > 0.5   x2 > 0.5)) instead of (x1 < x2) when producing the training and test datasets. Unlike above, use nodesize = 12 for this exercise.



(a)

```
## telabels3
##   0   1
## 477 523

## Mean of Miss classification using n_tree
##  n_tree=1- 0.24032
##  n_tree=10- 0.120329
##  n_tree=100- 0.075465

## Variance of Miss classification using n_tree
##  n_tree=1- 0.01416965
##  n_tree=10- 0.002856301
##  n_tree=100- 0.001215871
```

(b)

Figure 3

d. Answer the following questions:

a. What happens with the mean and variance of the error rate when the number of trees in the random forest grows?

To increase the predictiveness of the model as much as possible at each partitioning so that the model keeps gaining information about the dataset by increasing the number of trees used in the model.From the above observations mean of miss classification error rate is decreased by the increasing the number of trees.At the same time variance of the error rate is keep decreasing.

b. The third dataset represents a slightly more complicated classification problem than the first one. Still, you should get better performance for it when using sufficient trees in the random forest. Explain why you get better performance.

Random forest uses bagging (picking a sample of observations rather than all of them) and random subspace method (picking a sample of features rather than all of them, in other words - attribute bagging) to grow a tree. If the number of observations is large, but the number of trees is too small, then some observations will be predicted only once or even not at all. If the number of predictors is large but the number of trees is too small, then some features can (theoretically) be missed in all subspaces used. Both cases results in the decrease of random forest predictive power. But the last is a rather extreme case, since the selection of subspace is performed at each node.
But in this case , number of predictors (p=2) that is very small.To estimate an average of a real-valued random variable, from x1 and x2 we can take a sample.The expected variance will decrease as the square root of the sample size, and at a certain point the cost of collecting a larger sample will be higher than the benefit in accuracy obtained from such larger sample.
In this case we observe that in a multiple experiment on a single test set a forest of 12 trees performs better than a forest of 25 trees. This may be due to statistical variance.
Finally, ntree for the model is selected based on the number of predictors that we used in our data.

c. Why is it desirable to have low error variance?

The variances denote how well the model can generalize the data and how much our prediction accuracy is varying for training and test data. The model can Randomize the training data clearly for the all 1000 datasets of 100 rows.So the variance of miss classification error is low for test data.

Assignment 2: Mixture Models

In this assignment we were tasked to implement the EM algorithm for a mixture of multivariate Bernoulli distributions. In the Appendix the full code is under "Assignment 2", here, I will only present the code I wrote to solve the assignment and one short summary at the end.
The first part was to code the expectation step in the EM-algorithm. This step is aimed at computing the expected value of the following expression.

$$p(z_{nk}|x_n, \mu, \pi) = \frac{\pi_k p(x_n|\mu_k)}{\sum_k \pi_k p(x_n|\mu_k)} \tag{1}$$

So, what is this? To start off, the EM algorithm is used for maximizing likelihoods in problems where data is either missing or unobserved. One example could be that we have a dataset $\mathbf{X}$ of person characteristics (e.g. age, height etc), and we know that they are from K different cities and we want to classify them accordingly. However, the problem is that we don't know which city each person are from. So if let $\mathbf{Z}$ be the set of values for this latent variable, then when we try to estimate the parameters θ we'd get a difficult optimizing problem. However, if we know $\mathbf{Z}$ then this problem would obviously be less challenging. So the question that the EM algorithm is trying to answer is how we can find the values of $\mathbf{Z}$ and use them to estimate θ. And the way we find the values $\mathbf{Z}$ is by using Bayes rule, which is expressed in eq. (1).
So in the first step of the algorithm, we want to compute $p(z_{nk}|x_n, \mu, \pi)$. This is basically finding the probability distribution of the variables z. Now, in our case what these latent variables $z_{n,k}$ really means is the posterior probability of assigning the variable x_n to cluster K under the current parameter

estimates. Further, the variable π_k is the mixing coefficient for the k$_{th}$ component, i.e. the fraction of the number of observations in component K and the total number of observations N. Hence, π_k can be thought of as the prior probability of picking x_i from component K.

```
b_p <- exp(x %*% log(t(mu)) + (1 - x) %*% log(1 - t(mu)))
pi_m <- matrix(rep(pi,N), nrow=N, ncol=K)
p_x <- b_p * pi_m
z <- p_x / rowSums(p_x)
```

In the above snippet is the code used to find the posterior distribution $z_{n,k}$. So here b_p is the multivariate Bernoulli(μ) distribution, expressed for convenience as an one parameter exponential family. This is easily found through:

$$Ber(\mu) = \mu^x(1-\mu)^{(1-x)} =$$
$$exp(log(\mu^x(1-\mu)^{(1-x))}) =$$
$$exp(xlog(\mu) + (1-x)log(1-\mu))$$

Then we create the matrix pi_m, which is a $[1000xK]$ matrix of Bernoulli probabilities, i.e. P(X = x) for a given μ in component K and dimension D. We then multiply the likelihood by the prior, and scaling it by the marginal distribution p(x), to get the posterior distribution z.
Further, we were tasked to compute the log likelihood. Below is how we implemented it.

```
l_hood <- x %*% log(t(mu)) + (1-x) %*% log(1-t(mu))

 for(n in 1:N){
   for(k in 1:K){
     llik[it] <- llik[it] + z[n,k] * l_hood[n,k] + log(pi[k])
   }
 }
```

The variable *llik* is a given vector supposed to contain the log likelihood estimates of the EM iterations, and its being iterated over the main for loop, hence the index *it*. However, this is just the log likelihood function for the multivariate Bernoulli distribution with an added log(π_k) term. Further, if the log likelihood has not changed significantly, i.e. not changed more than 0.1 between two consecutive iterations, the loop should break. Below is how that was implemented.

```
 ifelse(it > 1 & llik[it] - llik[it-1] < min_change,
stop("EM has converged"), llik[it])
```

And then finally, the maximization step. Here the algorithm should update the old estimate of the parameters, based on the ML-estimation from the following two expressions.

$$\begin{aligned} \pi_k^{ML} &= \frac{\sum_n Z_{nk}}{N} \\ \mu_{ki}^{ML} &= \frac{\sum_n Z_{nk} x_{ni}}{\sum_n Z_{nk}} \end{aligned} \tag{2}$$

The R code for the M-step is below.

```
# Update pi
 pi <- colSums(z)/N
```

In this above code snippet, we update the values of the K different mixing coefficients. So since π_k is the ratio of N_k/N, we can find the number of observations in each component from z. By summing each column in z we are summing the posterior probabilities of each K, which in turn is the expected number of observations in each K.

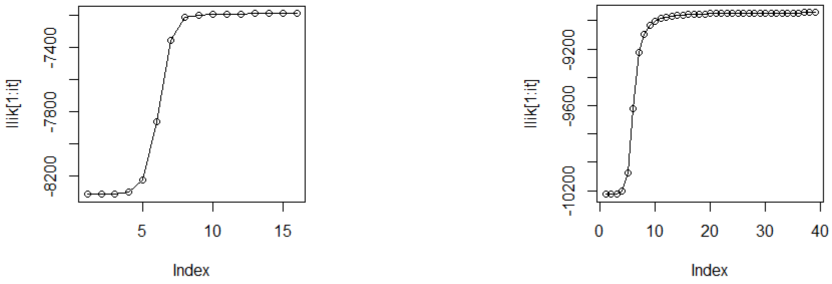
```
# Update mu
    mu <- t(z) %*% x/colSums(z)
```

Since μ is the parameter in the multivariate Bernoulli distribution, this just the expected value of each x_i in each component k.

Results.
We know that the data was generated as a mix of three multivariate Ber(μ) distributions, and we are here tasked to experiment and see what happens when we has an initial guess of the number of mixtures K = 2,3,4. Below is a table over some metrics.

	# Iterations	Converged	Log likelihood
$K = 2$	16	Yes	-7189.721
$K = 3$	39	Yes	-8943.678
$K = 4$	81	Yes	-11170.7

As we can see, after only 3 iterations, the algorithm converged in all the three cases. This is visualized in figure 1 and 2 below.



(a) Estimated log likelihood against number of iterations for K=2.

(b) Estimated log likelihood against number of iterations for K=3.

Figure 4

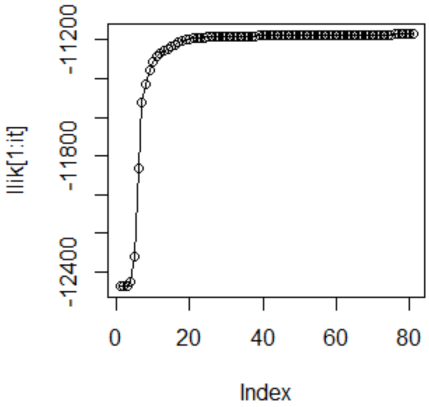


Figure 5: Estimated log likelihood against number of iterations for K=4.

We can see that the results are quite similar, even though the log likelihoods are different. But how about the final parameter estimates? Below is a table summarizing how well the EM algorithm managed to estimate the mixing coefficients π when the initial guess of K equals the true number of components.

	True π	$K = 3$ estimates
π_1	1/3	0.3307975
π_2	1/3	0.3286631
π_3	1/3	0.3405394

It seems as if we managed to estimate the true parameter quite well! Below is a table for the estimates when $K = 2$ & 3.

	True π	$K = 2$ estimates	$K = 4$ estimates
π_1	1/3	0.4992407	0.2529477
π_2	1/3	0.5007593	0.2453253
π_3	1/3	-	0.2505768
π_4	-	-	0.2511503

It's hard to compare the estimates when we know which how many components there are, and the true values off the mixing coefficients. However, we can note that the condition that $\sum \pi_i = 1$ holds in all three cases.
Finally below, there is a summary on how well we managed to estimate μ.

	True μ									
μ_1	1/2	3/5	2/5	7/10	3/10	4/5	1/5	9/10	1/10	1
μ_2	1/2	2/5	3/5	3/10	7/10	1/5	4/5	1/10	9/10	0
μ_3	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2

And the following table contains the estimates when initial $K = 3$. When creating this table, I had to take only the 4 first decimals.

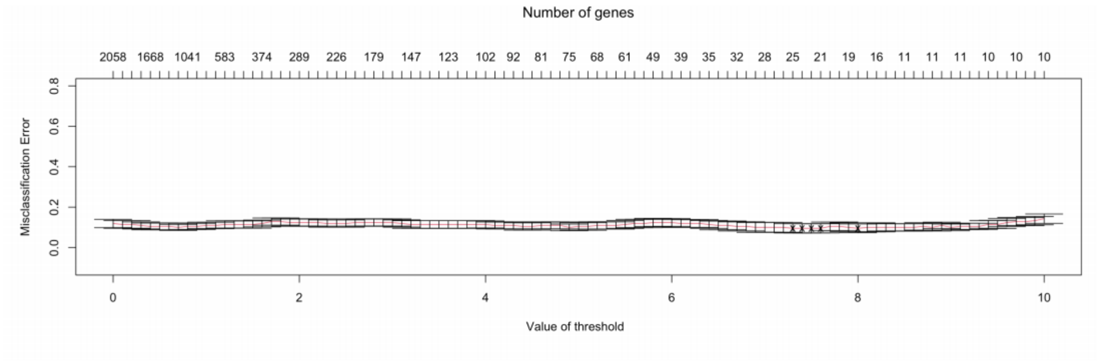
	μ									
μ_1	0.4761	0.3811	0.6244	0.3070	0.6960	0.2038	0.7835	0.1373	0.8885	0.0076
μ_2	0.4872	0.4859	0.4690	0.4948	0.5146	0.4987	0.4573	0.5082	0.4724	0.4684
μ_3	0.5119	0.5879	0.4178	0.7246	0.2836	0.7829	0.2187	0.8678	0.0904	0.9999

Even though we managed to estimate π quite well, these estimates of μ are very inaccurate as soon as the true μ is larger or smaller than 1/2. This might have something to with how the initial training data for μ was simulated (via runif(D,0.49,0.51)), which could explain why we more or less only see estimates inside that interval.

Assignment 3: High Dimensional Problems

1.

In the first task we were supposed to divide the data, and then perform nearest shrunken centroid classification of training data in which the threshold is chosen by cross-validation. Figure 3 below is showing the centroid plot.



```
## Best value of threshold:  7.3
## Number of genes:  25
```

Figure 6

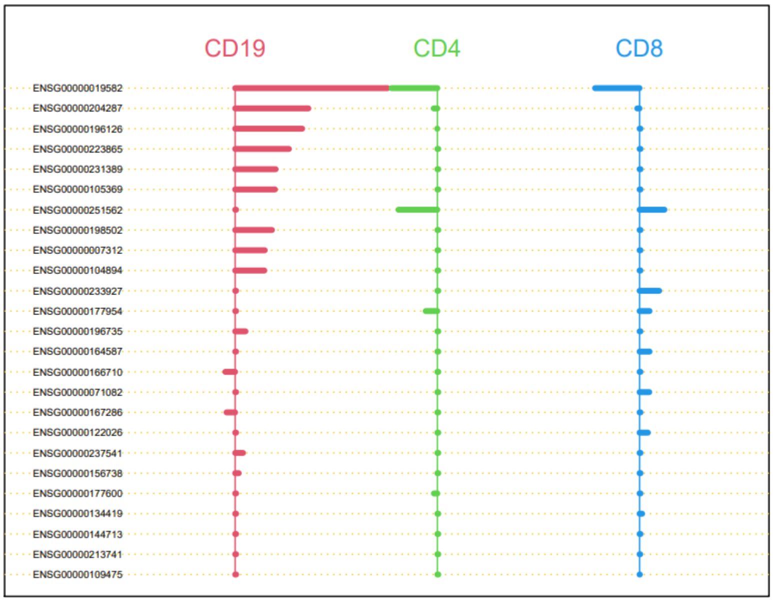
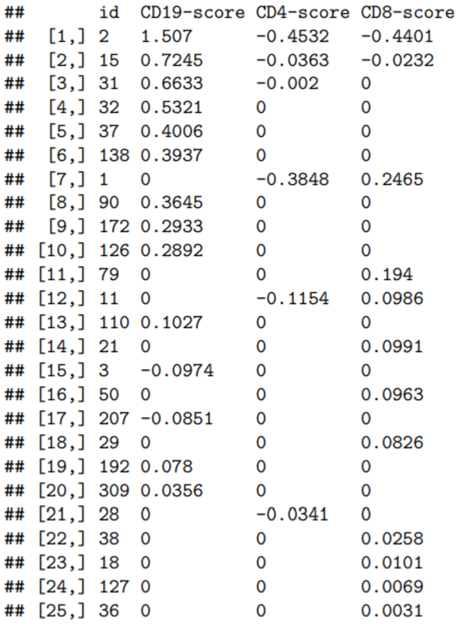


Figure 7



```
##       id  CD19-score CD4-score CD8-score
##  [1,] 2   1.507      -0.4532   -0.4401
##  [2,] 15  0.7245     -0.0363   -0.0232
##  [3,] 31  0.6633     -0.002    0
##  [4,] 32  0.5321     0         0
##  [5,] 37  0.4006     0         0
##  [6,] 138 0.3937     0         0
##  [7,] 1   0          -0.3848   0.2465
##  [8,] 90  0.3645     0         0
##  [9,] 172 0.2933     0         0
## [10,] 126 0.2892     0         0
## [11,] 79  0          0         0.194
## [12,] 11  0          -0.1154   0.0986
## [13,] 110 0.1027     0         0
## [14,] 21  0          0         0.0991
## [15,] 3   -0.0974    0         0
## [16,] 50  0          0         0.0963
## [17,] 207 -0.0851    0         0
## [18,] 29  0          0         0.0826
## [19,] 192 0.078      0         0
## [20,] 309 0.0356     0         0
## [21,] 28  0          -0.0341   0
## [22,] 38  0          0         0.0258
## [23,] 18  0          0         0.0101
## [24,] 127 0          0         0.0069
## [25,] 36  0          0         0.0031
```

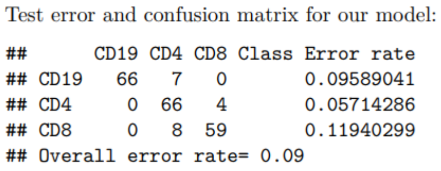
Figure 8

Each value in the centroid plot is the standardized difference in frequency for the gene in the given class versus all classes. A positive score indicates a higher frequency in that class while a negative score indicates a lower relative frequency.
No, it is not possible that all values in the centroid plot for a given gene are positive. This gene would be shrunken to zero.

2.

In the following table we list the names of the 2 most contributing genes.

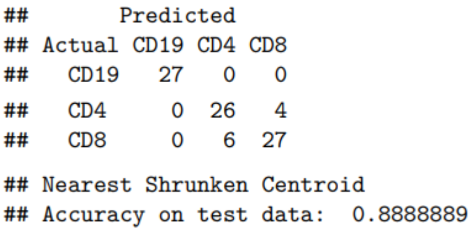
Gene	Name	Marker
ENSG00000019582	CD74	B-Cells, Luminal epithelial cells, Macrophages
ENSG00000204287	HLADRA	



Test error and confusion matrix for our model:

```
##       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
```

(a) Test error and confusion matrix for our model.

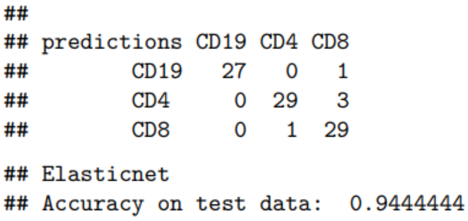


```
##        Predicted
## Actual CD19 CD4 CD8
##   CD19   27   0   0
##   CD4     0  26   4
##   CD8     0   6  27

## Nearest Shrunken Centroid
## Accuracy on test data:  0.8888889
```

(b) Testing our model by predicting outcomes on our test set.

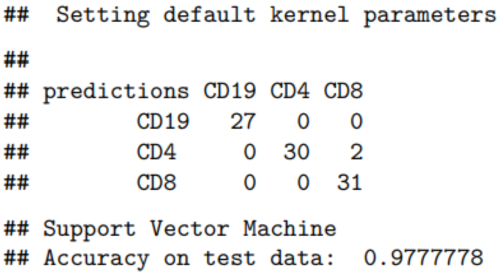
Figure 9

3.



```
##
## predictions CD19 CD4 CD8
##        CD19   27   0   1
##        CD4     0  29   3
##        CD8     0   1  29

## Elasticnet
## Accuracy on test data:  0.9444444
```

(a) Elasticnet.



```
##  Setting default kernel parameters

##
## predictions CD19 CD4 CD8
##        CD19   27   0   0
##        CD4     0  30   2
##        CD8     0   0  31

## Support Vector Machine
## Accuracy on test data:  0.9777778
```

(b) Support Vector Machine.

Figure 10

The below table is showing the model comparison.

Model	Accuracy	Number Predictors
NSC	0.8888889	25
Elasticnet	0.9444444	54
SVM	0.9777778	74

For our test set we get the highest accuracy with our SVM model, then we observe the second best accuracy on the elasticnet model and finally the NSC model shows the lowest accuracy. From that point of view we would choose the SVM model as the best model. On the other hand the SVM model is the model that is least explainable since the number of predictors is highest in comparison.

4.

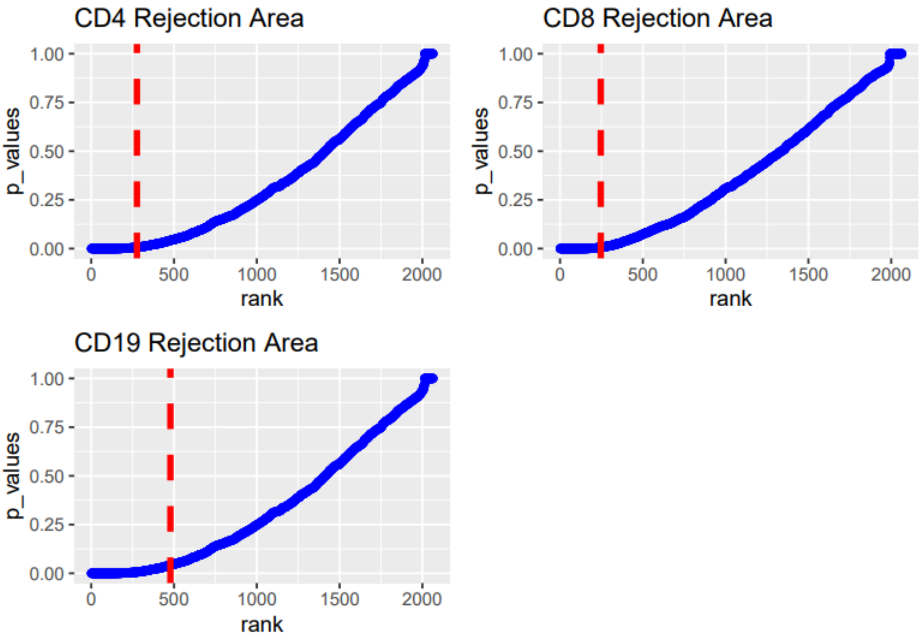


Figure 11

```
## CD4
## Number of genes corresponding to rejected hypothesis:  276
##
## CD8
## Number of genes corresponding to rejected hypothesis:  246
##
## CD19
## Number of genes corresponding to rejected hypothesis:  478
```

Figure 12

Plot interpretation: All genes with a rank (lowest p-value means rank 1, second lowest p-value means rank 2, and so on) lower than the rank with the highest p-value that is smaller than the Benjamini-Hochberg critical value will be rejected. The vertical red-dashed line represents this value for all different cell types. Therefore, every gene left from that line is inside the rejection area.

Appendix: R Code

Assignment 1: Ensemble Methods

```
library(randomForest)
set.seed(12345)
list1 = list()
for (i in 1:1000) { # Indicate number of iterations with "i"
x1 = runif(100)
x2 = runif(100)
y<-as.factor(as.numeric(x1<x2))
list1[[i]] = data.frame(x1,x2,y)
}

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))
table(telabels)
#m- Denotes the miss classification error rate (_number indicates
the number of trees)
#n- Denotes the accuracy

m_1<-c()
m_10<-c()
m_100<-c()
n_1<-c()
n_10<-c()
n_100<-c()
for (i in list1){
set.seed(12345)

n<-as.data.frame(i)
rf1=randomForest(y~.,data=n,ntree=1,nodesize=25,keep.forest = TRUE)
rf2=randomForest(y~.,data=n,ntree=10,nodesize=25,keep.forest = TRUE)
rf3=randomForest(y~.,data=n,ntree=100,nodesize=25,keep.forest = TRUE)

pred1<-predict(rf1 ,newdata = tedata)
pred2<-predict(rf2 ,newdata = tedata)
pred3<-predict(rf3 ,newdata = tedata)

accuracy1<-mean(telabels == pred1)
accuracy2<-mean(telabels == pred2)
accuracy3<-mean(telabels == pred3)

cm1 <- table(actual = telabels , fitted = pred1)
mmce1 <- 1 - (sum(diag(cm1))/sum(cm1))
cm2 <- table(actual = telabels , fitted = pred2)
mmce2 <- 1 - (sum(diag(cm2))/sum(cm2))
cm3 <- table(actual = telabels , fitted = pred3)
mmce3 <- 1 - (sum(diag(cm3))/sum(cm3))


n_1<-c(n_1,accuracy1)
n_10<-c(n_10,accuracy2)
n_100<-c(n_100,accuracy3)

m_1<-c(m_1,mmce1)
m_10<-c(m_10,mmce2)
m_100<-c(m_100,mmce3)
}
trd<-data.frame(n_1,n_10,n_100,m_1,m_10,m_100)

#Condition- 2 (x1<0.5)
set.seed(12345)
list2 = list()
for (i in 1:1000) { # Indicate number of iterations with ”i”
x1 = runif(100)
x2 = runif(100)
y<-as.factor(as.numeric(x1<0.5))
list2[[i]] = data.frame(x1,x2,y)
}

set.seed(1234)
x1<-runif(1000)
x2<-runif(1000)
tedata2<-cbind(x1,x2)
y<-as.numeric((x1<0.5))
telabels2<-as.factor(y)
```

```
plot(x1,x2,col=(y+1))
table(telabels2)


#list2
m2_1<-c()
m2_10<-c()
m2_100<-c()
n2_1<-c()
n2_10<-c()
n2_100<-c()
for (i in list2){
set.seed(12345)
n<-as.data.frame(i)

rf12=randomForest(y~.,data=n,ntree=1,nodesize=25,keep.forest = TRUE)
rf22=randomForest(y~.,data=n,ntree=10,nodesize=25,keep.forest = TRUE)
rf32=randomForest(y~.,data=n,ntree=100,nodesize=25,keep.forest = TRUE)

pred1<-predict(rf12,newdata = tedata2)
pred2<-predict(rf22,newdata = tedata2)
pred3<-predict(rf32,newdata = tedata2)

accuracy1<-mean(telabels2 == pred1)
accuracy2<-mean(telabels2 == pred2)
accuracy3<-mean(telabels2 == pred3)

cm1 <- table(actual = telabels2, fitted = pred1)
mmce1 <- 1 - (sum(diag(cm1))/sum(cm1))
cm2 <- table(actual = telabels2, fitted = pred2)
mmce2 <- 1 - (sum(diag(cm2))/sum(cm2))
cm3 <- table(actual = telabels2, fitted = pred3)
mmce3 <- 1 - (sum(diag(cm3))/sum(cm3))

n2_1<-c(n2_1,accuracy1)
n2_10<-c(n2_10,accuracy2)
n2_100<-c(n2_100,accuracy3)
m2_1<-c(m2_1,mmce1)
m2_10<-c(m2_10,mmce2)
m2_100<-c(m2_100,mmce3)
}
```

```
trd2<-data.frame(n2_1,n2_10,n2_100,m2_1,m2_10,m2_100)

#Condition- 3 ((x1<0.5 & x2<0.5) | (x1>0.5 & x2>0.5))
set.seed(12345)
list3 = list()
for (i in 1:1000) { # Indicate number of iterations with "i"
x1 = runif(100)
x2 = runif(100)
y<-as.factor(as.numeric((x1<0.5 & x2<0.5) | (x1>0.5 & x2>0.5)))
list3[[i]] = data.frame(x1,x2,y)
}

set.seed(1234)
x1<-runif(1000)
x2<-runif(1000)
tedata3<-cbind(x1,x2)
y<-as.numeric((x1<0.5 & x2<0.5) | (x1>0.5 & x2>0.5))
telabels3<-as.factor(y)
plot(x1,x2,col=(y+1))
table(telabels3)

#list3
m3_1<-c()
m3_10<-c()
m3_100<-c()
n3_1<-c()
n3_10<-c()
n3_100<-c()
for (i in list3){
set.seed(12345)
n<-as.data.frame(i)

rf13=randomForest(y~.,data=n,ntree=1,nodesize=12,keep.forest = TRUE)
rf23=randomForest(y~.,data=n,ntree=10,nodesize=12,keep.forest = TRUE)
rf33=randomForest(y~.,data=n,ntree=100,nodesize=12,keep.forest = TRUE)

pred1<-predict(rf13,newdata = tedata3)
pred2<-predict(rf23,newdata = tedata3)
pred3<-predict(rf33,newdata = tedata3)
accuracy1<-mean(telabels3 == pred1)
accuracy2<-mean(telabels3 == pred2)
accuracy3<-mean(telabels3 == pred3)

cm1 <- table(actual = telabels3, fitted = pred1)
mmce1 <- 1 - (sum(diag(cm1))/sum(cm1))
cm2 <- table(actual = telabels3, fitted = pred2)
mmce2 <- 1 - (sum(diag(cm2))/sum(cm2))
cm3 <- table(actual = telabels3, fitted = pred3)
mmce3 <- 1 - (sum(diag(cm3))/sum(cm3))

n3_1<-c(n3_1,accuracy1)
n3_10<-c(n3_10,accuracy2)
n3_100<-c(n3_100,accuracy3)
m3_1<-c(m3_1,mmce1)
m3_10<-c(m3_10,mmce2)
m3_100<-c(m3_100,mmce3)
}
trd3<-data.frame(n3_1,n3_10,n3_100,m3_1,m3_10,m3_100)
```

Assignment 2: EM Algorithm

```
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)
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

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="yellow")
  Sys.sleep(0.5)

  # E-step: Computation of the fractional component assignments
  b_p <- exp(x %*% log(t(mu)) + (1 - x) %*% log(1 - t(mu)))
  pi_m <- matrix(rep(pi,N), nrow=N, ncol=K)
  p_x <- b_p * pi_m
  z <- p_x / rowSums(p_x)

  #Log likelihood computation.
  l_hood <- x %*% log(t(mu)) + (1-x) %*% log(1-t(mu))
 for(n in 1:N){
   for(k in 1:K){
     llik[it] <- llik[it] + z[n,k] * l_hood[n,k] + log(pi[k])
   }
 }

  cat("iteration:␣", it, "log␣likelihood:␣", llik[it], "\n")
  flush.console()
  # Stop if the log likelihood has not changed significantly

  ifelse(it > 1 & llik[it] - llik[it-1] < min_change,
  stop("EM␣has␣converged"), llik[it])


  #M-step: ML parameter estimation from the data and fractional
  component assignments

  pi <- colSums(z)/N

  # Update mu

  mu <- t(z) %*% x/colSums(z)


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

Assignment 3: Higher Dimensional Models

```
data = read.csv("geneexp.csv")

# Remove X Column since this is not a genename
data = data[,-1]
n = dim(data)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.7))
train = data[id,]
test = data[-id,]
```

```r
rownames(train) = 1:nrow(train)
# transpose predictors to fit the prerequisities for pamr package
x_train = t(train[,-2086])
y_train = train[[2086]]
train_data = list(x=x_train, y=as.factor(y_train),
geneid=as.character(1:nrow(x_train)), genenames=rownames(x_train))
model = pamr.train(train_data, threshold=seq(0, 10, 0.1))

cvmodel = pamr.cv(model, train_data)
# pamr.plotcv(cvmodel)

lowest_error_index = which.min(cvmodel[["error"]])
best_threshold = cvmodel[["threshold"]][lowest_error_index]
number_of_genes_selected = cvmodel[["size"]][lowest_error_index]
cat("Best value of threshold: ", best_threshold,
"\nNumber of genes: ", number_of_genes_selected)

# Plot model with optimum threshold
pamr.plotcen(model, train_data, threshold=best_threshold)
a = pamr.listgenes(model, train_data, threshold=best_threshold)
most_contributing_genes <-
" | Gene | Name | Marker |
|--------------------|:--------------------:|------------------------------------------------------------------------:|
| ENSG00000019582 | CD74 | B-Cells, Luminal epithelial cells,
Macrophages |
| ENSG00000204287 | HLADRA | / |
"

cat(most_contributing_genes)
# Test error and confusion matrix
pamr.confusion(model, threshold=best_threshold, extra=TRUE)
x_test = t(test[,-2086])
y_test = test[[2086]]

predictions = pamr.predict(model, x_test , threshold=best_threshold)
confusion_matrix_test_nsc = as.matrix(table(Actual = y_test,
Predicted = as.vector(predictions)))
confusion_matrix_test_nsc
```

```
n = sum(confusion_matrix_test_nsc)
correct_predictions = diag(confusion_matrix_test_nsc)
accuracy = sum(correct_predictions) / n

cat(”Nearest␣Shrunken␣Centroid\nAccuracy␣on␣test␣data:␣”, accuracy)
cv_elnet = cv.glmnet(t(x_train), t(y_train), alpha=0.5,
family=”multinomial”)
opt_lambda = cv_elnet$lambda.min

predictions = predict(cv_elnet, t(x_test), type=”class”,
s=opt_lambda)
confusion_matrix_elnet = table(predictions, t(y_test))
confusion_matrix_elnet

n = sum(confusion_matrix_elnet)
correct_predictions = diag(confusion_matrix_elnet)
accuracy = sum(correct_predictions) / n

cat(”Elasticnet\nAccuracy␣on␣test␣data:␣”, accuracy)
train$CellType = as.factor(train$CellType)
classifier = ksvm(CellType~., data=train, kernel=”vanilladot”)

predictions = predict(classifier, t(x_test))
confusion_matrix_svm = table(predictions, t(y_test))
confusion_matrix_svm

n = sum(confusion_matrix_svm)
correct_predictions = diag(confusion_matrix_svm)
accuracy = sum(correct_predictions) / n

cat(”Support␣Vector␣Machine\nAccuracy␣on␣test␣data:␣”, accuracy)
tabl <- ”
|␣Model␣|␣Accuracy␣|␣Number␣Predictors␣|
|———————————|:——————————:|——————————————:|
|␣NSC␣|␣0.8888889␣|␣25␣|
|␣Elasticnet␣|␣0.9444444␣|␣54␣|
|␣SVM␣|␣0.9777778␣|␣74␣|
”

cat(tabl)
cd4 = data
```

```
cd4$CellType[(cd4$CellType == ”CD8”) | (cd4$CellType == ”CD19”)]
= ”OTHER”

cd8 = data
cd8$CellType[(cd8$CellType == ”CD4”) | (cd8$CellType == ”CD19”)]
= ”OTHER”

cd19 = data
cd19$CellType[(cd19$CellType == ”CD4”) | (cd19$CellType == ”CD8”)]
= ”OTHER”

predictors_to_test = colnames(data)[-2086]

cd4_pvals = sapply(cd4[-2086], function(x) t.test(x ~
cd4$CellType)$p.value)
cd8_pvals = sapply(cd4[-2086], function(x) t.test(x ~
cd8$CellType)$p.value)
cd19_pvals = sapply(cd4[-2086], function(x) t.test(x ~
cd19$CellType)$p.value)

cd4_sorted_pvals = cd4_pvals[order(unlist(cd4_pvals),
decreasing=FALSE)]
cd8_sorted_pvals = cd8_pvals[order(unlist(cd8_pvals),
decreasing=FALSE)]
cd19_sorted_pvals = cd19_pvals[order(unlist(cd19_pvals),
decreasing=FALSE)]

# Dropping NA values returned from t.test()
cd4_sorted_pvals = cd4_sorted_pvals[!sapply(cd4_sorted_pvals,
is.nan)]
cd8_sorted_pvals = cd8_sorted_pvals[!sapply(cd8_sorted_pvals,
is.nan)]
cd19_sorted_pvals = cd19_sorted_pvals[!sapply(cd19_sorted_pvals,
is.nan)]

alpha = 0.05
benjamini_hochberg <- function(pvals, alpha) {
M = length(pvals)
bh_vals = c()
for (ind in 1:M) {
critical_val = (ind / M) * alpha
bh_vals = c(bh_vals, critical_val)
}
return(bh_vals)
}

cd4 = data.frame(cbind(cd4_sorted_pvals,
benjamini_hochberg(cd4_sorted_pvals, alpha),
c(1:length(cd4_sorted_pvals))))

colnames(cd4) = c("p_values", "bh_values", "rank")
cd8 = data.frame(cbind(cd8_sorted_pvals,
benjamini_hochberg(cd8_sorted_pvals, alpha),
c(1:length(cd8_sorted_pvals))))
colnames(cd8) = c("p_values", "bh_values", "rank")
cd19 = data.frame(cbind(cd19_sorted_pvals,
benjamini_hochberg(cd19_sorted_pvals, alpha),
c(1:length(cd19_sorted_pvals))))
colnames(cd19) = c("p_values", "bh_values", "rank")

# Find maximum p_value that is smaller than bh_value.
Reject everything below that value.

cd4_max_pval = cd4[as.numeric(cd4$p_values) <
as.numeric(cd4$bh_values), ]
cd8_max_pval = cd8[as.numeric(cd8$p_values) <
as.numeric(cd8$bh_values), ]
cd19_max_pval = cd19[as.numeric(cd19$p_values) <
as.numeric(cd19$bh_values), ]

cd4_max_pval = max(as.numeric(cd4_max_pval$p_values))
cd8_max_pval = max(as.numeric(cd8_max_pval$p_values))
cd19_max_pval = max(as.numeric(cd19_max_pval$p_values))

cd4_rejected_genes = cd4_sorted_pvals[cd4_sorted_pvals <
cd4_max_pval]
cd8_rejected_genes = cd8_sorted_pvals[cd8_sorted_pvals <
cd8_max_pval]
cd19_rejected_genes = cd19_sorted_pvals[cd19_sorted_pvals <
cd19_max_pval]

cd4_plot = ggplot(cd4, aes(rank, p_values, title="CD4"))
```

```
+ geom_point(color="blue", size=1.5)
+ geom_vline(xintercept = length(cd4_rejected_genes),
linetype="dashed", color = "red", size=1.5)
+ ggtitle("CD4␣Rejection␣Area")

cd8_plot = ggplot(cd8, aes(rank, p_values, title="CD8"))
+ geom_point(color="blue", size=1.5)
+ geom_vline(xintercept = length(cd8_rejected_genes),
linetype="dashed", color = "red", size=1.5)
+ ggtitle("CD8␣Rejection␣Area")

cd19_plot = ggplot(cd4, aes(rank, p_values))
+ geom_point(color="blue", size=1.5)
+ geom_vline(xintercept = length(cd19_rejected_genes),
linetype="dashed", color = "red", size=1.5)
+ ggtitle("CD19␣Rejection␣Area")

grid.arrange(cd4_plot, cd8_plot, cd19_plot, nrow = 2)

cat("CD4\nNumber␣of␣genes␣corresponding␣to␣rejected␣hypothesis:␣",
length(cd4_rejected_genes), "\n\nCD8\nNumber␣of␣genes␣corresponding
to␣rejected␣hypothesis:␣", length(cd8_rejected_genes),
"\n\nCD19\nNumber␣of␣genes␣corresponding␣to␣rejected␣hypothesis:␣",
length(cd19_rejected_genes))
```