Computer lab 1 block 2

Martynas Lukosevicius, Alejo Perez Gomez, Shwetha Vandagadde Chandramouly

02/12/2020

Statement of Contribution

- Assignment 1 - Shwetha Vandagadde Chandramouly
- Assignment 2 - Alejo Perez Gomez
- Assignment 3 - Martynas Lukosevicius

Assignment 1. Ensemble methods

Fitting random forest to training data and test data

```r
rforest_1 = randomForest(trainlabels~., data = traindata, ntree = 1, nodesize = 25,
                         keep.forest = TRUE)
rforest_10 = randomForest(trainlabels~., data = traindata, ntree = 10, nodesize = 25,
                          keep.forest = TRUE)
rforest_100 = randomForest(trainlabels~., data = traindata, ntree = 100, nodesize = 25,
                           keep.forest = TRUE)

y_1 = predict(rforest_1,testdata)
y_10 = predict(rforest_10,testdata)
y_100 = predict(rforest_100,testdata)

missclass=function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}

y1_missclass = missclass(y_1,testlabels)
y10_missclass = missclass(y_10,testlabels)
y100_missclass = missclass(y_100,testlabels)
```

Misclassification for test data of n = 1000 with 1 tree = 0.188

Missclassification for test data of n = 1000 with 10 trees = 0.142

Missclassification for test data of n = 1000 with 100 trees = 0.112

Here we can observe that missclassification error reduces as the number of trees increases.

1.

Repeating the procedure for 1000 training datasets of size 100.

Mean and variance of misclassification errors for condition $x1 < x2$:

	mean	variance
1 tree	0.211323	0.0032972
10 trees	0.141111	0.0009715
100 trees	0.116073	0.0009574

2.

Repeating the same procedure as above with condition : $x1 < 0.5$

Mean and variance of misclassification errors :

	mean	variance
1 tree	0.094046	0.0173643
10 trees	0.015022	0.0005776
100 trees	0.006120	0.0000780

3.

Repeating the same procedure as above with condition : (x1 < 0.5 AND x2 < 0.5) OR (x1 > 0.5 AND x2 > 0.5)

Mean and variance of misclassification errors :

	mean	variance
1 tree	0.250956	0.0136136
10 trees	0.119573	0.0031013
100 trees	0.074899	0.0012565

4.

a.

The mean and variance of the error decreases as the number of trees grow. In general misclassification reduces and one gets better results with increasing the number of trees. However rate improvement in the result decreases with increase in trees, in above examples we can see that difference in mean of misclassification error between 1 tree and 10 trees is more that between 10trees and 100 trees.

b.

For the third condition, we can see that misclassification error was slightly higher than the other two conditions, but the results improve with the use of more trees. Random forest uses bagging , ie it picks a sample of observations rather than all of them. For a large observation, when the number of trees are too small, then some observations will be predicted only once or even not at all. This will result in a chance of high missclassification error. But using large number of trees solves this, resulting in better classification.

c.

Higher the variance , higher will be sensitivity of the model to the training data , and this will reduce the accuracy of the prediction when we fit it to the test data which which may be different from training data.

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.

First, it will be necessary to define our function of density of probability corresponding to the Bernoulli distribution. Let D be the dimensions of our vectors of observations x_n and μ_k.

$$Bernoulli(x_i, \mu_k) = \prod_{i=1}^{D} \mu_{ki}^{x_i}(1-\mu_{ki})^{1-x_i}$$

As for our matrix z_{nk}, its components will be defined as follows, with their values n comprised from 1 to N observations for X and its values k from 1 to K, being K the number of mixture components π_k. This will be the matrix of fractional component assignments.

$$z_{nk} = \frac{\pi_k * Bernoulli(x_n, \mu_k)}{\sum_{k=1}^{K} \pi_k * Bernoulli(x_n, \mu_k)}$$

Now, the log-likelihood function will be defined in this way:

$$\sum_{n=1}^{N}\sum_{k=1}^{K} p(z_n|x_n, \mu, \pi) * [ln(\pi_k) + \sum_{i=1}^{D}[x_{ni} * ln(\mu_{ki}) + (1-x_{ni}) * ln(1-\mu_{ki})]]$$

After Applying MLE over each mixture components and weights, we obtain:

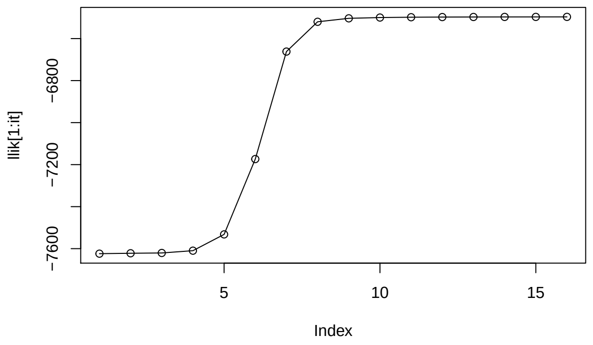
$$\hat{n_k}^{MLE} = \sum_{n=1}^{N} z_{nk}$$

$$\pi_k^{MLE} = \frac{\hat{n_k}}{N}$$

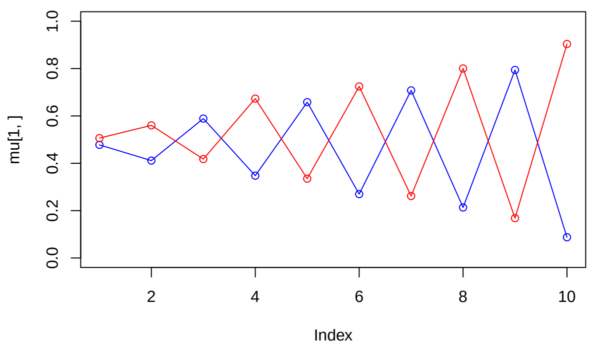
$$\mu_{ki}^{MLE} = \frac{\sum_{n=1}^{N} z_{nk} x_{ni}}{\hat{n}}$$

After this statement, executions of our implementation will be shown for K=2,3,4.

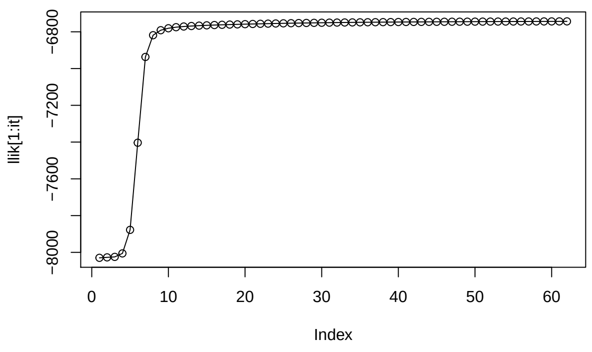
```
## Final iteration with K =  2 :  16 log likelihood:  -6496.662
## Here will be shown the log-likelihood plot of this experiment with K =  2
```



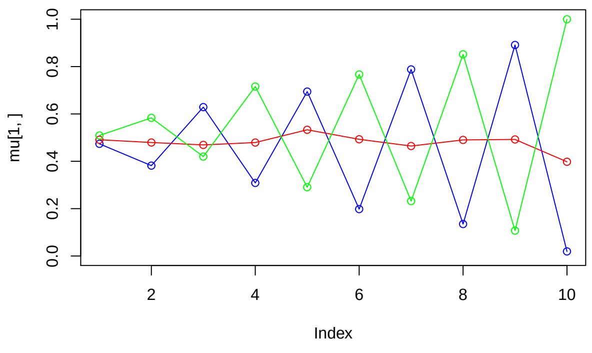
```
## Here will be shown the mu (mixture weights parameters) plot of this experiment with K =  2
```



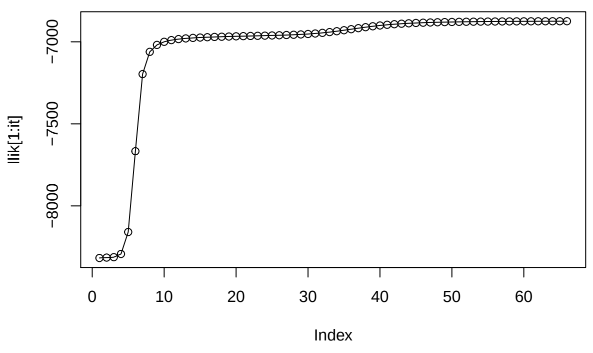
```
## Final iteration with K =  3 :  62 log likelihood:  -6743.326
## Here will be shown the log-likelihood plot of this experiment with K =  3
```



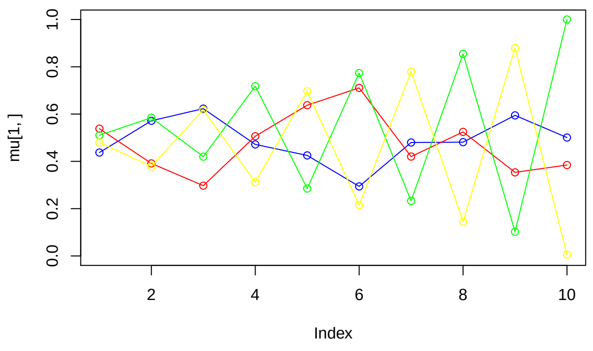
```
## Here will be shown the mu (mixture weights parameters) plot of this experiment with K =  3
```



```
## Final iteration with K =  4 :  66 log likelihood:  -6874.497
## Here will be shown the log-likelihood plot of this experiment with K =  4
```



```
## Here will be shown the mu (mixture weights parameters) plot of this experiment with K =  4
```



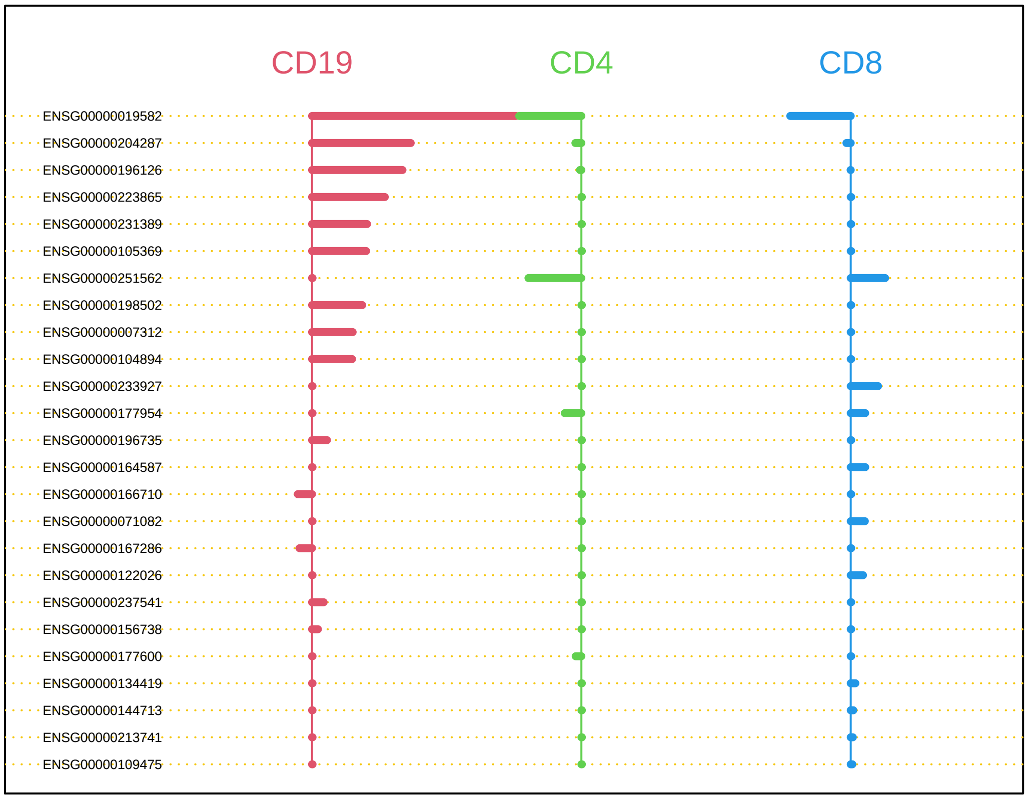
Once the experiments have been performed, let this table summarize results.

K	number of Iterations	Log-Likelihood Value
2	16	-6496.662
3	62	-6743.326
4	66	-6874.497

As we can see, creating 2 different clusters ($k = 2$), results in a faster convergence, needing just 16 iterations. As K increases, the log-likelihood decreases. This could be explained from an overfitting-based approach, leading us to think that too many components could result in worse performance.

Assignment 3. High-dimensional methods

1.



Centroid plot shows distance from gene mean value to global mean value. Positive or negative shows if mean is greater or smaller than the global mean. No, it can't happen that all values for some gene in centroid plot are positive, because distance is calculated to the mean of all values of the gene . Number of genes selected when threshold = 7.212944 : 25.

2.

Most contributed genes: ENSG00000019582, ENSG00000204287. alternative names: CD74, HLA-DRA. These genes are marker genes for B cells

Test error rate: 0.1111111

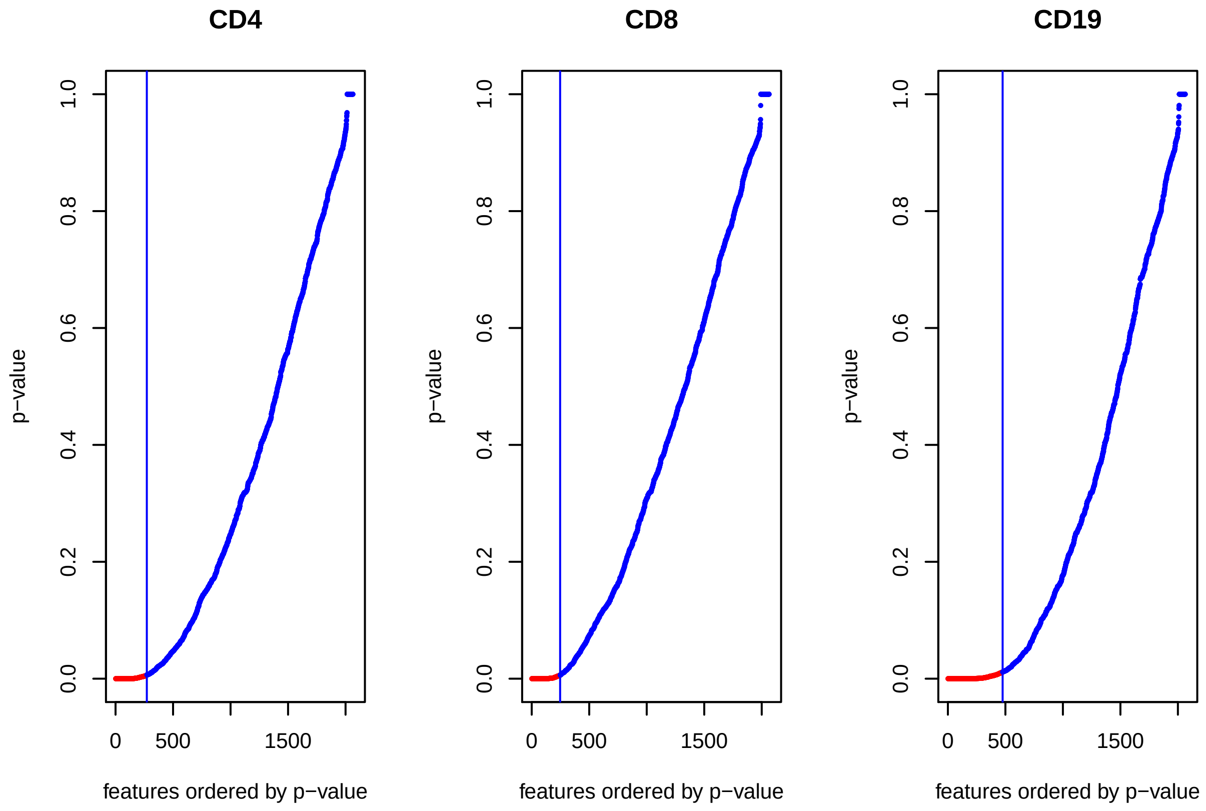
3.

Table bellow shows test error rate and number of features used.

	Test error rate	N. features
NSC	0.110	25
EN	0.056	54
SVM	0.022	2085

I prefer Elastic net, because SVM might overfit and use a lot of parameters, NSC might have underfited.

4.



points symbolize genes, red genes are rejected meaning that they have an effect on corresponding cell. Table bellow shows the amount of rejected genes for each cell.

cell	rejected
CD4	272
CD8	247
CD19	476

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
library(randomForest)
set.seed(12345)
x1 <- runif(100)
x2 <- runif(100)
traindata <- cbind(x1, x2)
y <- as.numeric(x1 < x2)
trainlabels <- as.factor(y)

set.seed(12345)
x1 <- runif(1000)
x2 <- runif(1000)
```

```r
testdata <- cbind(x1, x2)
y <- as.numeric(x1 < x2)
testlabels <- as.factor(y)
# plot(x1,x2,col=(y+1))
rforest_1 = randomForest(trainlabels ~ .,
    data = traindata, ntree = 1, nodesize = 25,
    keep.forest = TRUE)
rforest_10 = randomForest(trainlabels ~ .,
    data = traindata, ntree = 10, nodesize = 25,
    keep.forest = TRUE)
rforest_100 = randomForest(trainlabels ~
    ., data = traindata, ntree = 100, nodesize = 25,
    keep.forest = TRUE)

y_1 = predict(rforest_1, testdata)
y_10 = predict(rforest_10, testdata)
y_100 = predict(rforest_100, testdata)

missclass = function(X, X1) {
    n = length(X)
    return(1 - sum(diag(table(X, X1)))/n)
}

y1_missclass = missclass(y_1, testlabels)
y10_missclass = missclass(y_10, testlabels)
y100_missclass = missclass(y_100, testlabels)


missclass_large_sample = function(condition,
    ns = 25) {
    mce_1 = mce_10 = mce_100 = c()
    set.seed(12345)
    x1 <- runif(1000)
    x2 <- runif(1000)
    test <- cbind(x1, x2)
    y <- as.numeric(eval(parse(text = condition)))
    testlabels <- as.factor(y)
    for (i in 1:1000) {

        x1 <- runif(100)
        x2 <- runif(100)
        train <- cbind(x1, x2)
        y <- as.numeric(eval(parse(text = condition)))
        trainlabels <- as.factor(y)

        rf_1 = randomForest(trainlabels ~
            ., data = train, ntree = 1, nodesize = ns,
            keep.forest = TRUE)
        rf_10 = randomForest(trainlabels ~
            ., data = train, ntree = 10,
            nodesize = ns, keep.forest = TRUE)
        rf_100 = randomForest(trainlabels ~
            ., data = train, ntree = 100,
            nodesize = ns, keep.forest = TRUE)

        y_1 = predict(rf_1, newdata = test)
        y_10 = predict(rf_10, test)
        y_100 = predict(rf_100, test)

        mce_1[i] = missclass(y_1, testlabels)
        mce_10[i] = missclass(y_10, testlabels)
        mce_100[i] = missclass(y_100, testlabels)

    }
    mat = matrix(c(mean(mce_1), mean(mce_10),
        mean(mce_100), var(mce_1), var(mce_10),
        var(mce_100)), ncol = 2, nrow = 3)
    colnames(mat) = c("mean", "variance")
    rownames(mat) = c("1 tree", "10 trees",
        "100 trees")
    return(mat)
}
a = missclass_large_sample(condition = "x1 < x2")
knitr::kable(a)
b = missclass_large_sample(condition = "x1 < 0.5")
knitr::kable(b)

c = missclass_large_sample(condition = "(x1 < 0.5 & x2 < 0.5) |
                           (x1 > 0.5 & x2 > 0.5)",
    ns = 12)
knitr::kable(c)
library(ggplot2)
library(kknn)
library(reshape2)
library(knitr)
# Set so that long lines in R will be
# wrapped:
opts_chunk$set(tidy.opts = list(width.cutoff = 40),
    tidy = TRUE)
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(cache = TRUE)
whole_process <- function(K) {
    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)
```

```r
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 = 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 mu

for (it in 1:max_it) {

    Sys.sleep(0.5)

    # E-step: Computation of the fractional
    # component assignments
    divisorz_1 <- matrix(nrow = dim(z)[1],
        ncol = dim(z)[2])
    for (i in 1:N) {
        # loop for multinomial matrix NxK
        for (j in 1:K) {
            divisorz_1[i, j] <- prod((mu[j,
              ]^x[i, ]) * ((1 - mu[j,
              ])^(1 - x[i, ])))
        }
    }

    divisorz_2 <- matrix(pi, nrow = N,
        ncol = length(pi), byrow = TRUE)  #pi's at the divisor
    divisor_z <- rowSums(divisorz_2 *
        divisorz_1)
    z <- (divisorz_2 * divisorz_1)/(divisor_z)


    # Log likelihood computation.
    ln_pi <- log(divisorz_2)
    z_loglike_part1 <- matrix(nrow = dim(z)[1],
        ncol = dim(z)[2])
    for (i in 1:N) {
        # loop for multinomial matrix NxK
        for (j in 1:K) {
            z_loglike_part1[i, j] <- sum((log(mu[j,
              ]) * x[i, ]) + ((log(1 -
              mu[j, ])) * (1 - x[i, ])))
        }
    }

    llik[it] <- sum(rowSums((z_loglike_part1 +
        ln_pi) * z))

    # cat('iteration: ', it, 'log likelihood:
    # ', llik[it], '\n')
    flush.console()
    # Stop if the log likelihood has not
    # changed significantly
    if ((it != 1) && (abs(llik[it] -
        llik[it - 1]) < min_change)) {
        break
    }

    # M-step: ML parameter estimation from
    # the data and fractional component
    # assignments
    n_estimators <- colSums(z)

    pi <- n_estimators/N

    for (j in 1:K) {
        # Let's set the new mu estimator through
        # z (p(k|i))

        mu[j, ] <- (1/n_estimators[j]) *
            colSums(z[, j] * x)

    }
}

# pi mu
cat("Final iteration with K = ", K, ": ",
    it, "log likelihood: ", llik[it],
    "\n")
cat("Here will be shown the log-likelihood plot of this experiment with K = ",
    K)
    plot(llik[1:it], type = "o")

    cat("Here will be shown the mu (mixture weights parameters) plot of this experiment with K = ",
        K)
    if (K == 2) {
        plot(mu[1, ], type = "o", col = "blue",
            ylim = c(0, 1))
        points(mu[2, ], type = "o", col = "red")
    }
    if (K == 3) {
        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")
    }
    if (K == 4) {
        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")
    }


}
whole_process(2)
whole_process(3)
whole_process(4)
table_mat <- cbind(c(2, 3, 4), c(16, 62,
    66), c(-6496.662, -6743.326, -6874.497))
colnames(table_mat) <- c("K", "number of Iterations",
    "Log-Likelihood Value")
knitr::kable((table_mat))
library(readr)
data0 = read_csv("geneexp.csv")
data = data0[, -1]
data$CellType = as.factor(data$CellType)
set.seed(12345)
n <- dim(data)[1]
id = sample(1:n, floor(n * 0.7))
train = data[id, ]
test = data[-id, ]
library(pamr)
rownames(train) = 1:nrow(train)
genenames <- colnames(train[, -ncol(train)])
x <- t(train[, -ncol(train)])
y = train[[ncol(train)]]
mydata = list(x = x, y = as.factor(y), geneid = as.character(1:nrow(x)),
    genenames = genenames)
model = pamr.train(mydata)
cvmodel = pamr.cv(model, mydata)
minError <- which.min(cvmodel$error)
bestTh <- cvmodel$threshold[minError]
```

```r
pamr.plotcen(model, mydata, threshold = bestTh)
numberOfGenes <- model$nonzero[minError]
listOfGenes <- pamr.listgenes(model, mydata,
    bestTh)
bestGenes <- mydata$genenames[as.vector(as.numeric(listOfGenes[1:2,
    1]))]
rownames(test) = 1:nrow(test)
genenames <- colnames(test[, -ncol(test)])
x <- t(test[, -ncol(test)])
y = test[[ncol(test)]]
testdata = list(x = x, y = as.factor(y),
    geneid = as.character(1:nrow(x)), genenames = genenames)
predictions <- pamr.predict(model, testdata$x,
    threshold = bestTh)

# knitr::kable(table(testdata$y,confusionMatrix))
library(glmnet)
x <- as.matrix(train[, -ncol(train)])
cv.elastic <- cv.glmnet(x, train$CellType,
    alpha = 0.5, family = "multinomial")

res1 <- predict(cv.elastic, newx = as.matrix(test[,
    -ncol(test)]), s = cv.elastic$lambda.min,
    type = "class")
# knitr::kable(table(testdata$y,res1))
library("kernlab")

x <- as.matrix(unname(train[, -ncol(train)]))
svp = ksvm(x = x, y = train$CellType, kernel = "vanilladot")

res2 <- predict(svp, as.matrix(test[, -ncol(test)]))

results <- matrix(c(signif(sum(testdata$y !=
    predictions)/length(predictions), digits = 2),
    numberOfGenes), nrow = 1)
results <- rbind(results, c(signif(sum(testdata$y !=
    res1)/length(res1), digits = 2), cv.elastic$nzero[which(cv.elastic$glmnet.fit$lambda ==
    cv.elastic$lambda.min)]))
results <- rbind(results, c(signif(sum(testdata$y !=
    res2)/length(res2), digits = 2), nrow(mydata$x)))
row.names(results) <- c("NSC", "EN", "SVM")
colnames(results) <- c("Test error rate",
    "N. features")
knitr::kable(results)
test <- function(name) {
    y <- ifelse(data$CellType == name, 1,
        0)
    df <- data.frame(character(), numeric())
    for (i in 1:(dim(data)[2] - 1)) {
        test <- t.test(unlist(data[, i]) ~
            y, data = data, alternative = "two.sided")
        df <- rbind(df, c(colnames(data)[i],
            test$p.value))
    }
    colnames(df) <- c("name", "pvalue")
    return(df)
}

plotres <- function(name) {

    results <- test(name)

    results <- results[order(as.numeric(results$pvalue)),
        ]

    M <- length(results$pvalue)
    for (i in 1:M) {
        if (as.numeric(results$pvalue[i]) >
            (0.05 * i/M)) {
            break
        }

    }
    L <- i - 1
    amount <<- rbind(amount, c(name, L))
    p <- as.numeric(results$pvalue[L])
    rejected <- ifelse(as.numeric(results$pvalue) <=
        p, 1, 0)

    row.names(results) <- c(1:length(results$pvalue))

    title <- paste0(name)
    plot(c(1:L), results$pvalue[1:L], pch = 20,
        col = "red", xlim = c(0, M), ylim = c(0,
            1), cex = 0.5, xlab = "features ordered by p-value",
        ylab = "p-value", main = title)
    abline(v = L, col = "blue")
    points(c(L + 1:M), results$pvalue[L +
        1:M], pch = 20, col = "blue", xlim = c(0,
        M), ylim = c(0, 1), cex = 0.5)
}

amount = data.frame(name = character(), amount = numeric())
par(mfrow = c(1, 3))
plotres("CD4")
plotres("CD8")
plotres("CD19")
colnames(amount) <- c("cell", "rejected")
knitr::kable(amount)
```