Introduction to Machine Learning

Lab 3 Block 2

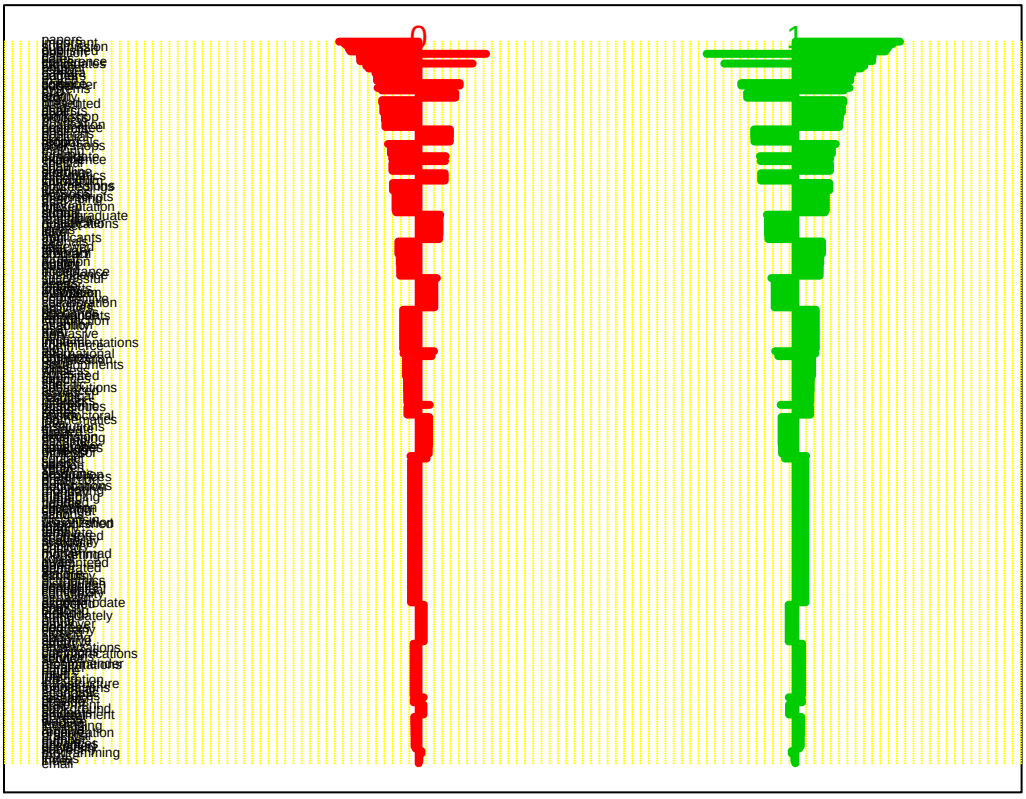
Rasmus Holm

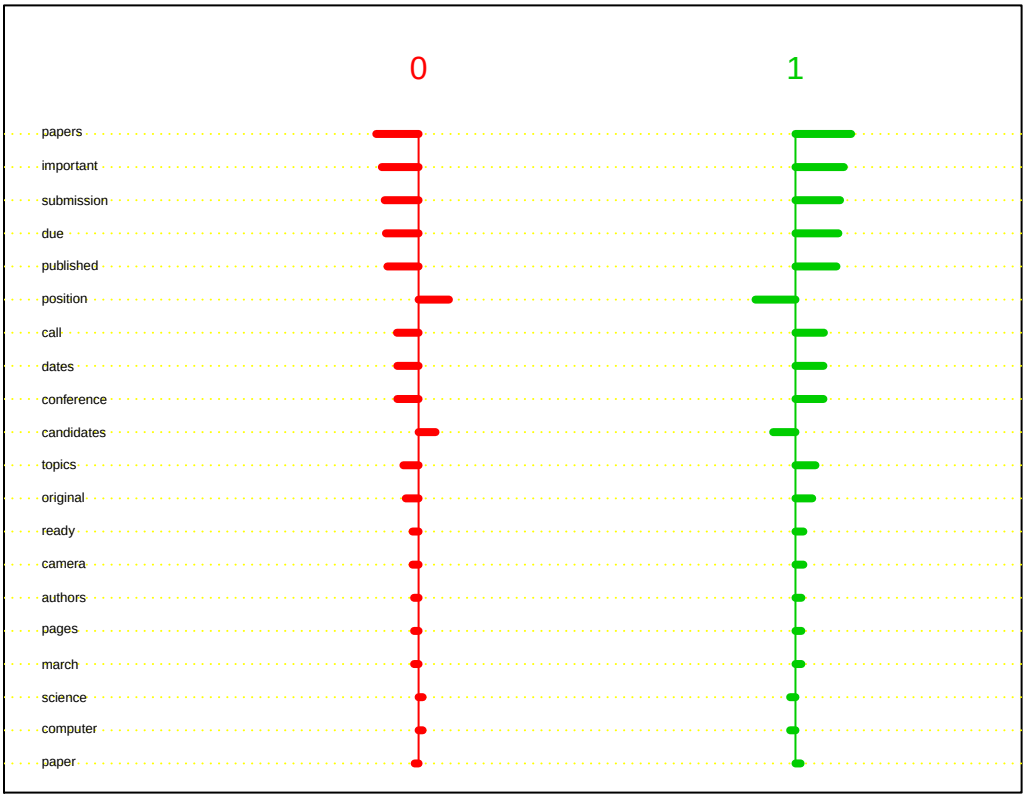
2016-12-14

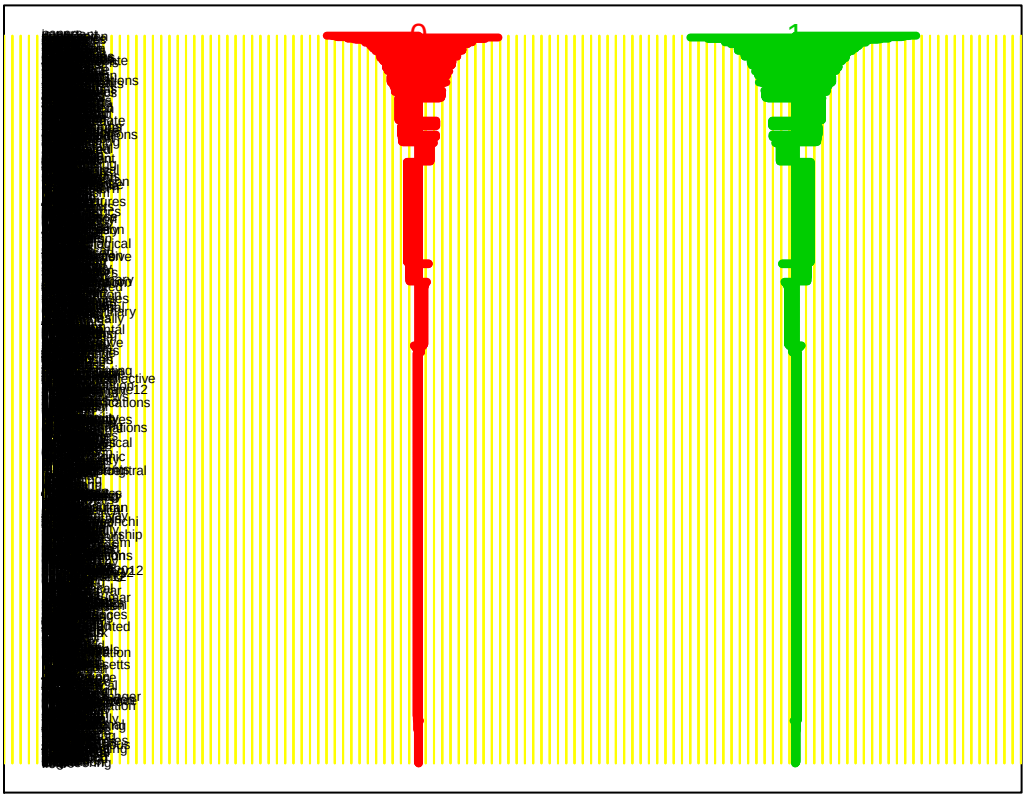
Contents

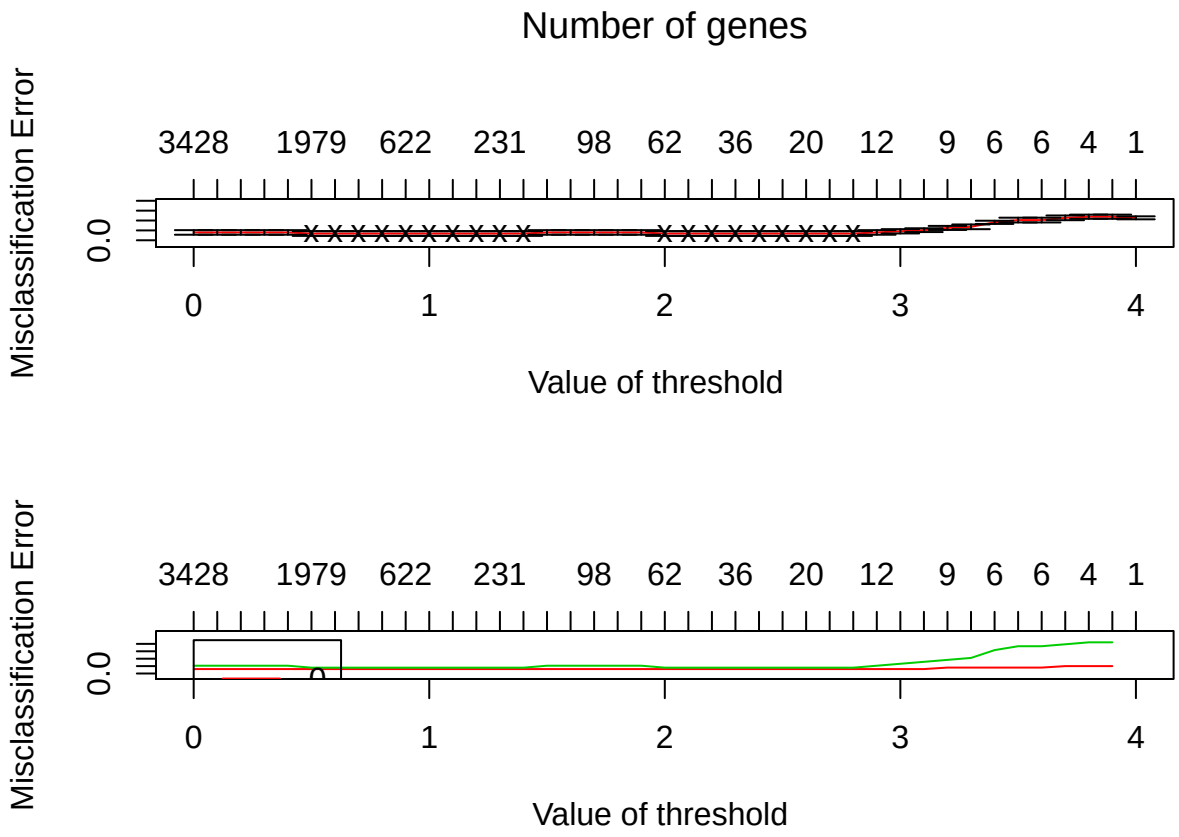
Assignment 1

1




0
1
papers
important
submission
due
published
position
call
dates
conference
candidates
topics
original
ready
camera
authors
pages
march
science
computer
paper


0
1



2

Elastic Net

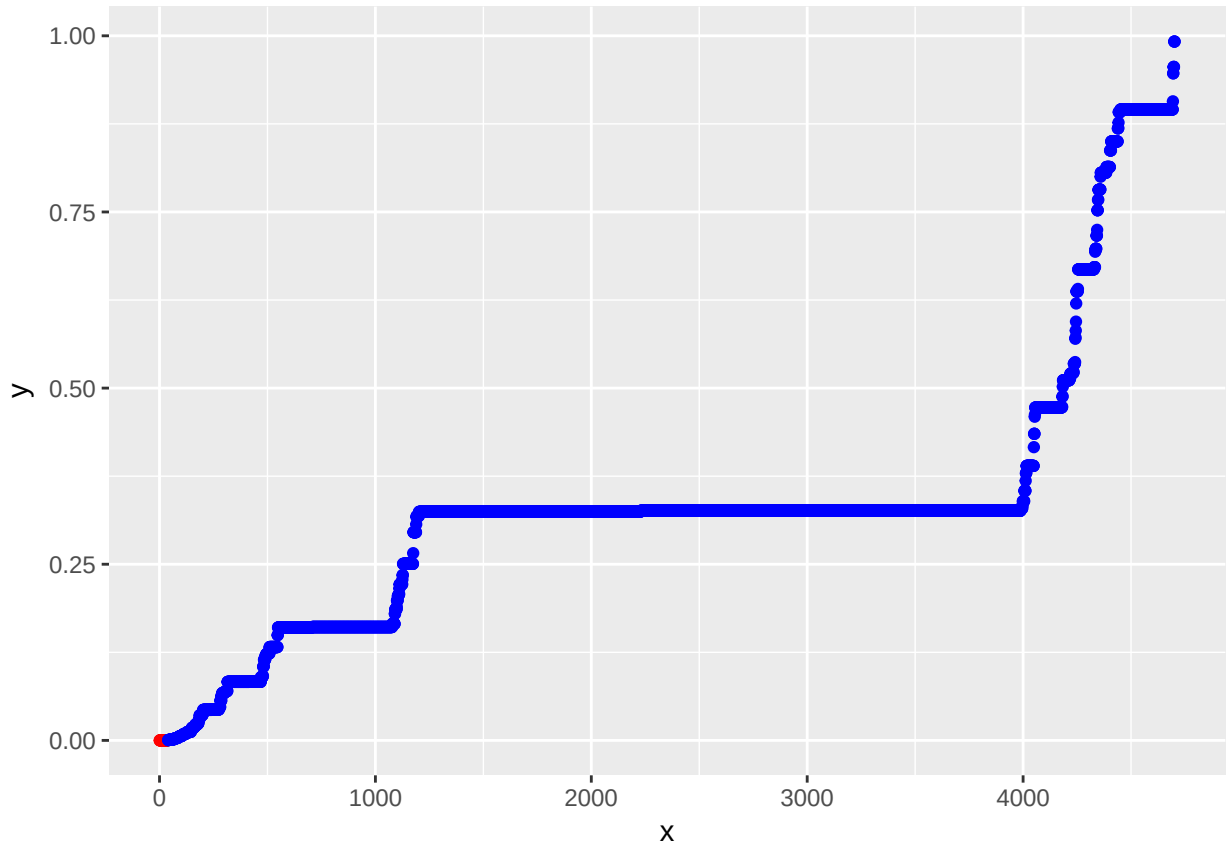
```
#> [1] 0.1311628
#> s35
#>  38
#> [1] 0.15
```

Support Vector Machine

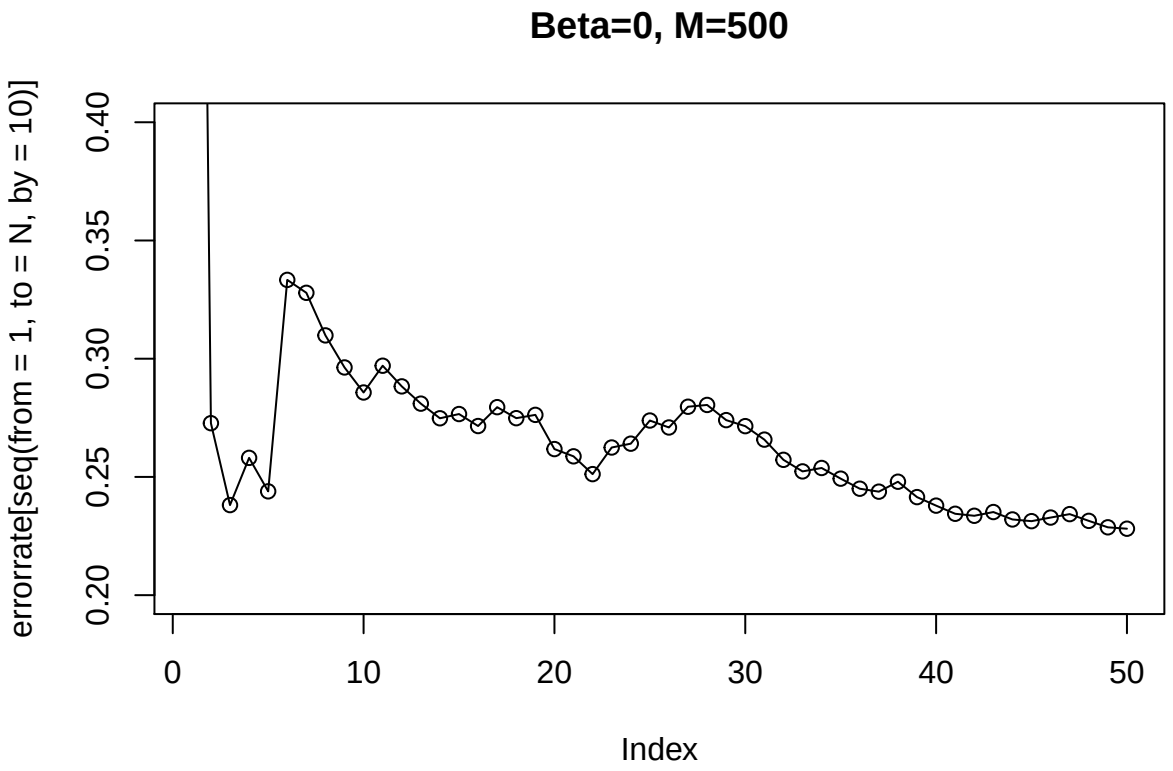
```
#>  Setting default kernel parameters
#> [1] 43
#> [1] 0.05
```

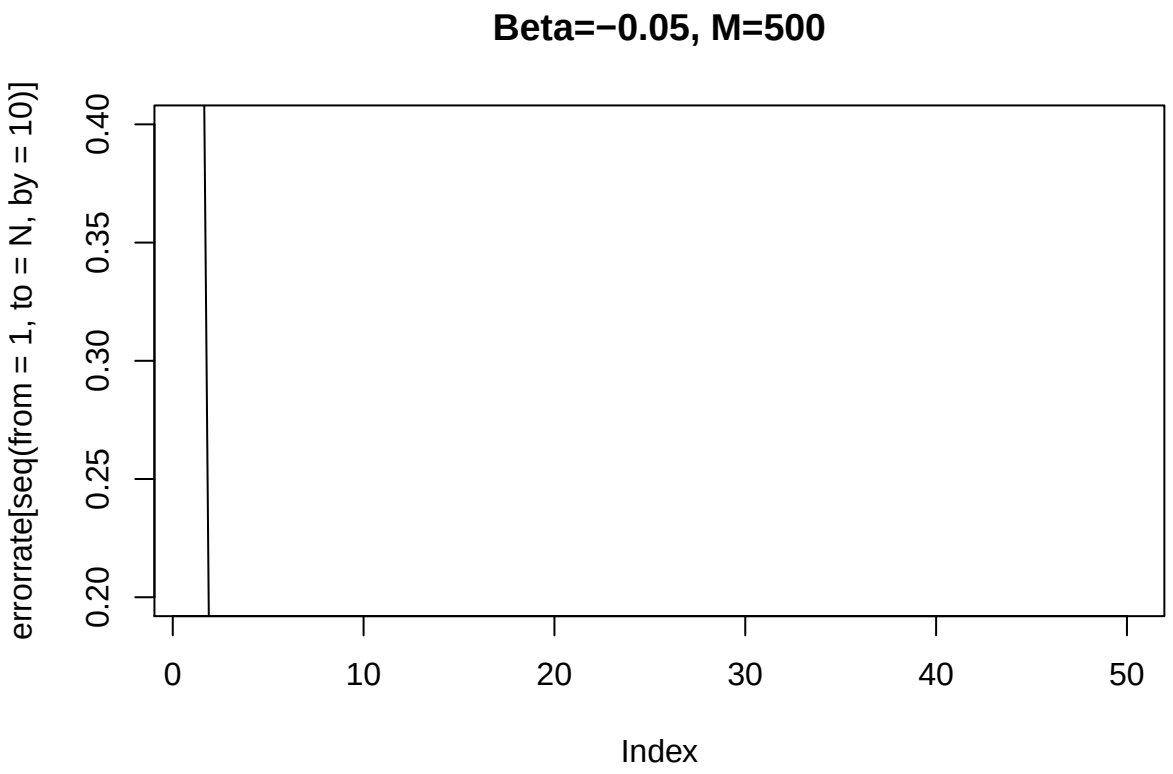
3

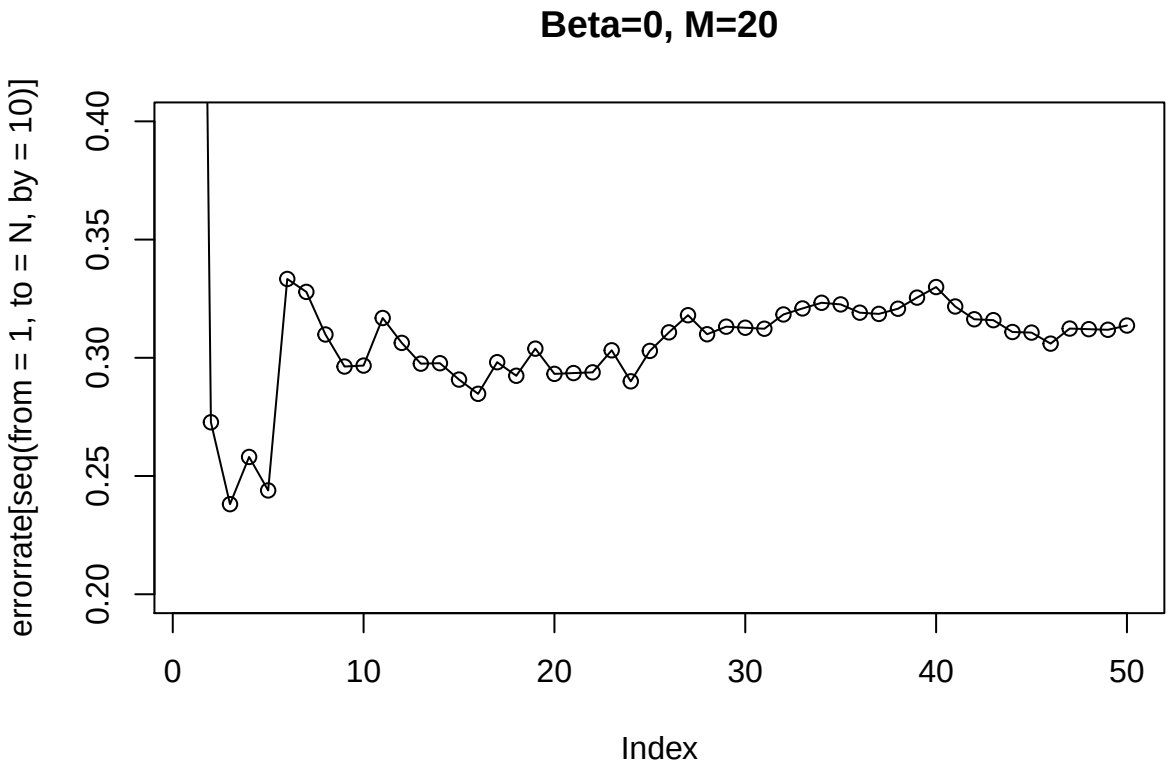
```
#>  [1] "papers"        "submission"    "position"      "published"     "important"     "call"
#> [32] "proceedings"   "apply"         "strong"        "international" "degree"        "excellent"
```

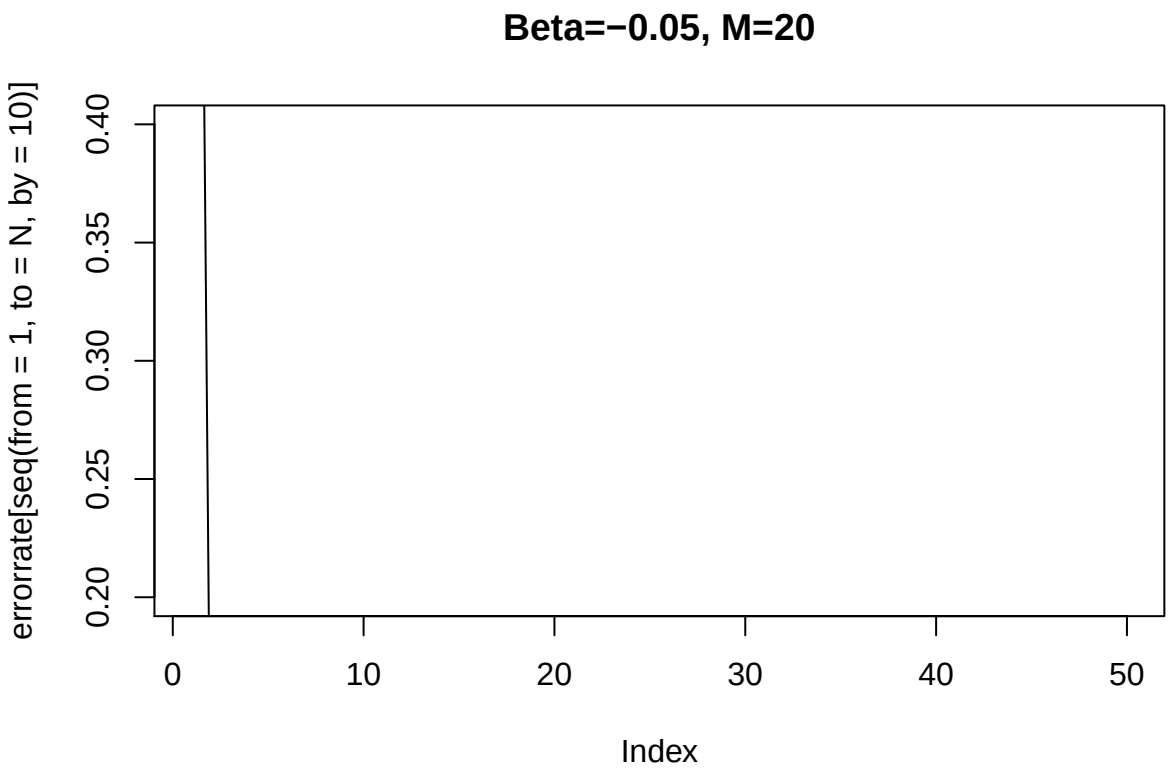

1.00
0.75
0.50
0.25
0.00
y
0
1000
2000
3000
4000
x

Assignment 2




Beta=−0.05, M=500
errorrate[seq(from = 1, to = N, by = 10)]
0.20
0.25
0.30
0.35
0.40
0
10
20
30
40
50
Index


Beta=0, M=20
errorrate[seq(from = 1, to = N, by = 10)]
0.20
0.25
0.30
0.35
0.40
0
10
20
30
40
50
Index


Beta=−0.05, M=20
errorrate[seq(from = 1, to = N, by = 10)]
0.20
0.25
0.30
0.35
0.40
0
10
20
30
40
50
Index

Appendix

Code for Assignment 1

```
library(pamr)
library(glmnet)
library(kernlab)
library(ggplot2)

data <- read.csv("../data/data.csv", sep=";", header=TRUE,
                 stringsAsFactors=FALSE, encoding="latin1")
rownames(data) <- 1:nrow(data)

set.seed(12345)
train_idx <- sample(nrow(data), size=floor(nrow(data) * 7 / 10))
train <- data[train_idx,]
test <- data[-train_idx,]

x <- t(train[, -ncol(data)])
y <- train[, ncol(data)]

x_test <- t(test[, -ncol(data)])
y_test <- test[, ncol(data)]
set.seed(12345)

nsc_data <- list(x=x, y=as.factor(y), geneid=as.character(1:nrow(x)), genenames=rownames(x))
model <- pamr.train(nsc_data, threshold=seq(0,4, 0.1))
cvmodel <- pamr.cv(model, nsc_data)

optimal_threshold <- cvmodel$threshold[which.min(cvmodel$error)]
optimal_size <- cvmodel$size[which.min(cvmodel$error)]

class_error <- 1 - (sum(pamr.predict(model, x_test,
                                     threshold=optimal_threshold) == y_test) /
                    length(y_test))

optimal_threshold
optimal_size
class_error

pamr.plotcen(model, nsc_data, threshold=1)
pamr.plotcen(model, nsc_data, threshold=2.5)
pamr.plotcen(model, nsc_data, threshold=optimal_threshold)


a <- pamr.listgenes(model, nsc_data, threshold=2.5)
cat(paste(colnames(data)[as.numeric(a[,1])], collapse='\n' ) )

a <- pamr.listgenes(model, nsc_data, threshold=optimal_threshold)
cat(paste(colnames(data)[as.numeric(a[,1])][1:10], collapse='\n' ) )

print(cvmodel)
pamr.plotcv(cvmodel)
```

```
set.seed(12345)

alpha <- 0.5
fit <- cv.glmnet(x=t(x), y=y, alpha=alpha, family="binomial")

optimal_lambda <- fit$lambda[which.min(fit$cvm)]
optimal_size <- fit$nzero[which.min(fit$cvm)]
class_error <- 1 - (sum(predict(fit, t(x_test), type="class") == y_test) / length(y_test))

optimal_lambda
optimal_size
class_error
set.seed(12345)

fit <- ksvm(x=t(x), y=y, kernel="vanilladot",
            type="C-svc", cross=10, scale=FALSE)

optimal_size <- fit@nSV
class_error <- 1 - (sum(predict(fit, t(x_test)) == y_test) / length(y_test))

optimal_size
class_error
benjamini_hochberg <- function(x, y, alpha) {
    pvalues <- apply(x, 2, function(feature) t.test(feature ~ y, alternative="two.sided")$p.value)
    m <- length(pvalues)

    sorted <- sort(pvalues)
    values <- 1:m * alpha / m

    L <- which.min(sorted < values) - 1
    mask <- sorted <= sorted[L]
    list(mask=mask, pvalues=sorted, features=colnames(x)[order(pvalues)][mask])
}

result <- benjamini_hochberg(x=data[,-ncol(data)], y=data[, ncol(data)], alpha=0.05)
result$features
ggplot() +
    geom_point(data=data.frame(x=1:length(result$features),
                               y=result$pvalues[result$mask]),
               aes(x=x, y=y), col="red") +
    geom_point(data=data.frame(x=((length(result$features) + 1):(ncol(data) -1)),
                               y=result$pvalues[!result$mask]),
               aes(x=x, y=y), col="blue")
```

Code for Assignment 2

```
set.seed(1234567890)
spam <- read.csv2("../data/spambase.csv")

ind <- sample(1:nrow(spam))
spam <- spam[ind,c(1:48,58)]
spam$Spam <- 2 * spam$Spam - 1
```

```
gaussian_k <- function(x, h) {
    ## Gaussian kernel
    exp(-(x / h)^2)
}

euclidean_d <- function(x, xi) {
    as.numeric(apply(x, 1, function(x) {
        sqrt(sum((x - xi)^2))
    }))
}

SVM <- function(sv, i) { # SVM on point i with support vectors sv
    ## Your code here
    ## Note that the labels in spambase.csv are 0/1 and SVMs need -1/+1. Then, use 2*label-1
    ## to convert from 0/1 to -1/+1
    ## Do not include the labels when computing the Euclidean distance between the point i
    ## and each of the support vectors. This is the distance to use in the kernel function
    ## You can use dist() to compute the Euclidean distance

    h <- 1
    b <- 0
    x <- sv[, -ncol(sv)]
    t <- sv[, ncol(sv)]

    predicted <- sum(t * gaussian_k(euclidean_d(x, i), h)) + b
    predicted
}

sv.least_important <- function(sv) {
    which.max(lapply(sv, function(m) {
        obs <- spam[m,]
        x <- obs[, -ncol(obs)]
        t <- obs$Spam
        y <- SVM(spam[sv,], x)
        h <- 1
        k <- gaussian_k(euclidean_d(x, x), h)
        t * (y - t * k)
    }))
}

run_BOSVM <- function(data, beta, M) {
    errors <- 1
    errorrate <- vector(length = N)
    errorrate[1] <- 1
    sv <- c(1)

    for(i in 2:N) {
        predicted <- SVM(data[sv,], data[i, -ncol(data)])

        if (data[i, "Spam"] * predicted < beta) {
            sv <- c(sv, i)
            errors <- errors + 1
```

```
            if (length(sv) > M) {
                sv <- sv[-sv.least_important(sv)]
            }
        }

        errorrate[i] <- errors / i
    }
    errorrate
}

errorrate <- run_BOSVM(spam, beta=0, M=500)
plot(errorrate[seq(from=1, to=N, by=10)], ylim=c(0.2,0.4), type="o",
     main="Beta=0, M=500")

errorrate <- run_BOSVM(spam, beta=-0.05, M=500)
plot(errorrate[seq(from=1, to=N, by=10)], ylim=c(0.2,0.4), type="o",
     main="Beta=-0.05, M=500")

errorrate <- run_BOSVM(spam, beta=0, M=20)
plot(errorrate[seq(from=1, to=N, by=10)], ylim=c(0.2,0.4), type="o",
     main="Beta=0, M=20")

errorrate <- run_BOSVM(spam, beta=-0.05, M=20)
plot(errorrate[seq(from=1, to=N, by=10)], ylim=c(0.2,0.4), type="o",
     main="Beta=-0.05, M=20")
```