# Introduction to Machine Learning

Lab 3 Block 2

*Rasmus Holm*

*2016-12-15*

## Contents

# Assignment 1

In this assignment I have used the data data set that contains 64 e-mails which were manually collected from DBWorld mailing list. Each e-mail consists of 4702 features, i.e. unique words in the e-mails, and I want to predict whether the e-mail is a conference e-mail or something else. In order to do so I have divided the data set into train and test sets corresponding to 70% respective 30% of the data.

## 1

Here I performed neastest shrunken centroid on the training set where the threshold was chosen by 10-fold cross-validation. Below are the results and we can see that the optimal threshold was 0.5 resulting in 1979 non-zero features and a classification error of 5%. We can see that words like *papers*, *submission*, *published*, and *conference* are important words in order to classify an e-mail as a conference e-mail which seems reasonable.

```
#> Threshold: 0.5
#> Size: 1979
#> Classification Error: 0.05
#> Top 10 features
#> papers
#> important
#> submission
#> due
#> published
#> position
#> call
#> conference
#> dates
#> candidates
```

Figure 1 is the resulting centroid plot that shows the contributions of the words to each class where a conference e-mail is a 1. We can see that the words have opposite contributions to the classes, i.e. if the contribution is high for class 1 then it is low for class 0. Unfortunately were many features non-zero making the plot difficult to interpret further.

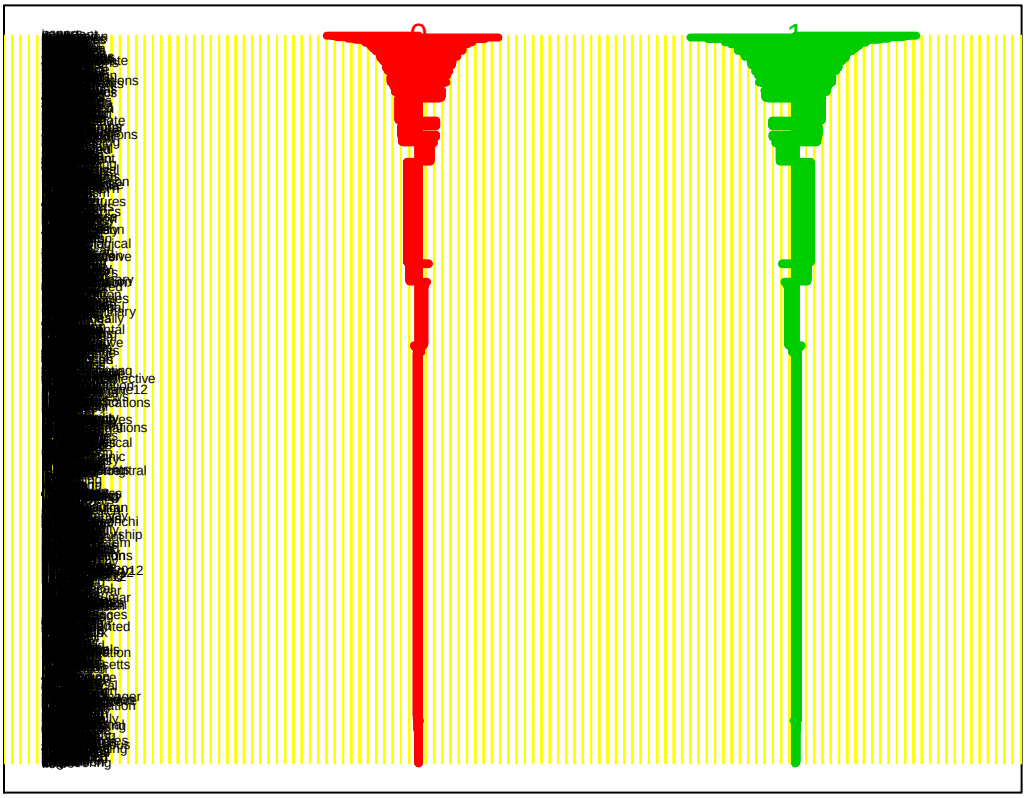


*Figure 1: Centroid plot.*

## 2

### Elastic Net

Elastic net is another method for feature extraction and I let cross-validation choose the penalty measurement. In this can it can be shown below that it picked *deviance* which results in 38 non-zero features and a misclassification rate of 15%.

```
#> Penalty Deviance
#> Lambda: 0.131162838159315
#> Size: 38
#> Classification Error: 0.15
```

### Support Vector Machine

Lastly I used the support vector machine and here the size correspond to the number of support vectors rather than features, i.e. all features are used but against a subset of training samples. The number of support vectors was 43 with a misclassification rate of 5% which compared to elastic net is very good.

```
#> Size: 43
#> Classification Error: 0.05
```

From the summary below we can see that nearest chrunken centroid and support vector machine are the best in terms of classification error. **TODO**: Which one is better? Fewer observations to check in support vector

but only 43 / 64 which is not that much. The number of features are still the same while in nsc the features are reduced by half. Is elastic net really that terrible considering the number of features?

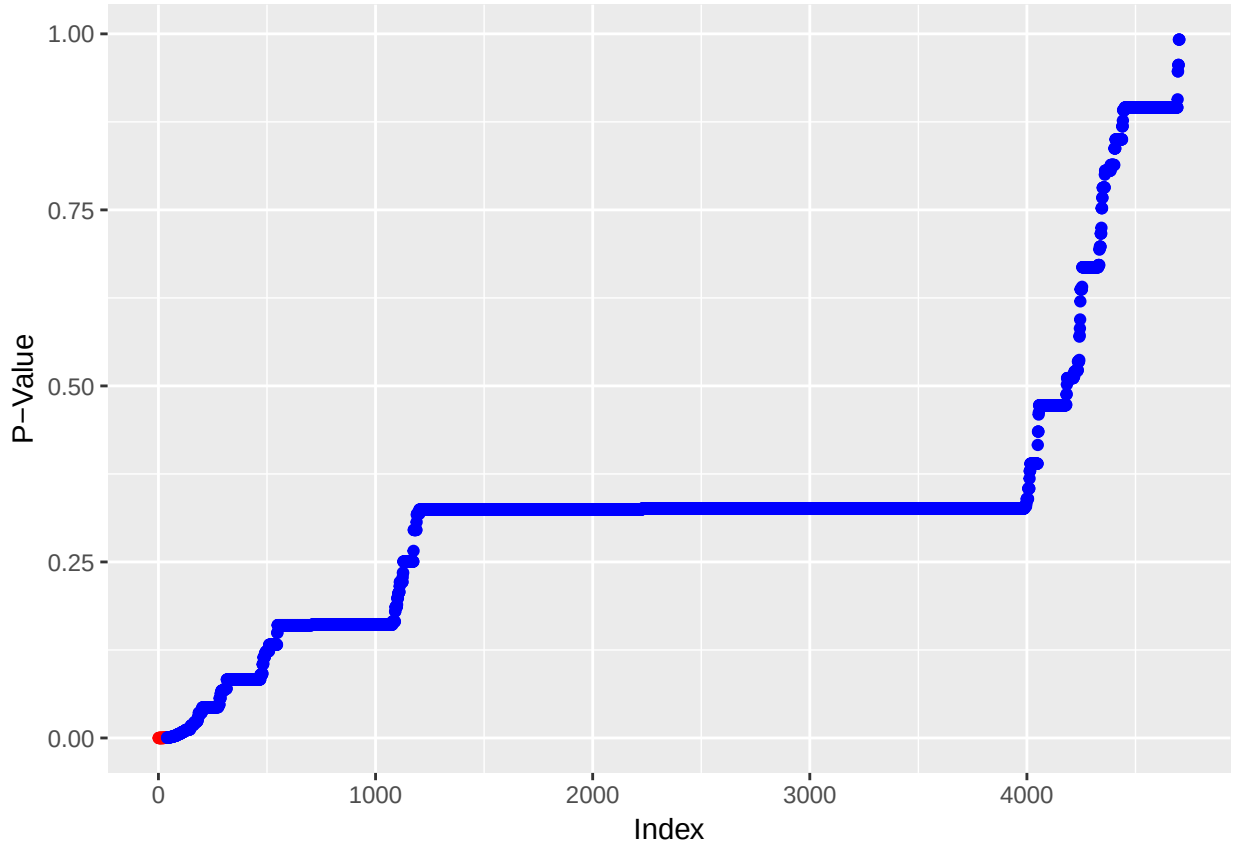
| | NSC | EN | SVM |
|---|---|---|---|
| Size | 1979 | 38 | 43 |
| Class. Err. | 5% | 15% | 5% |

## 3

```
#> [1] 39
```

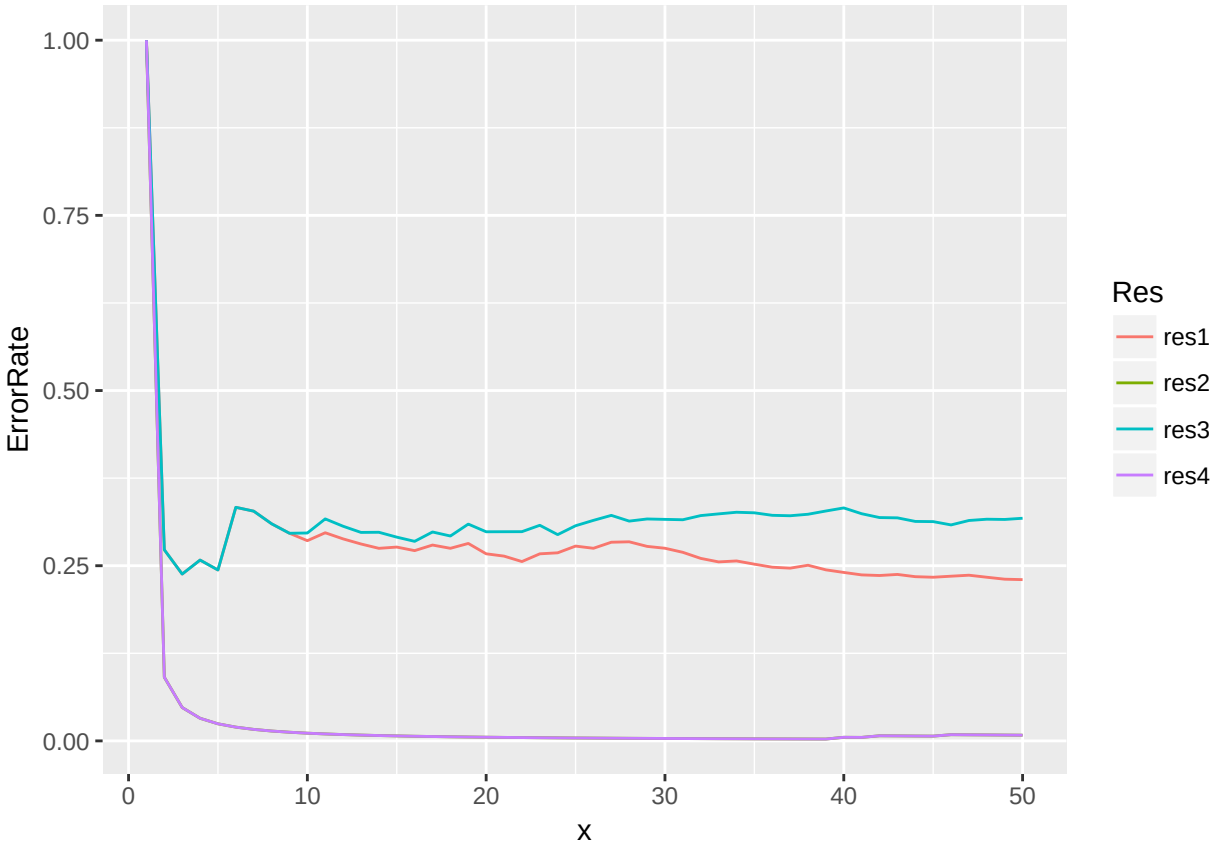
In total there were 39 rejected hypothesis by the Benjamini-Hochberg algorithm using $\alpha = 0.05$ and the ten most significant features found are quite similar to those found by the nearest chrunken centroid and make total sense as can be seen below.

```
#> Top 10 features
#> papers
#> submission
#> position
#> published
#> important
#> call
#> conference
#> candidates
#> dates
#> paper
```



## Assignment 2

```
#> Number of Support Vectors: 113
#> Number of Support Vectors: 4
#> Number of Support Vectors: 20
#> Number of Support Vectors: 4
```



# Appendix

## Code for Assignment 1

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

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)

nsc_optimal_threshold <- cvmodel$threshold[which.min(cvmodel$error)]
nsc_optimal_size <- cvmodel$size[which.min(cvmodel$error)]

nsc_class_error <- 1 - (sum(pamr.predict(model, x_test,
                                         threshold=nsc_optimal_threshold) == y_test) /
                    length(y_test))
genes <- pamr.listgenes(model, nsc_data, threshold=nsc_optimal_threshold)
cat(paste("Threshold:", nsc_optimal_threshold))
cat(paste("Size:", nsc_optimal_size))
cat(paste("Classification Error:", nsc_class_error))
cat("Top 10 features")
cat(paste(colnames(data)[as.numeric(genes[,1])][1:10], collapse='\n' ) )
pamr.plotcen(model, nsc_data, threshold=nsc_optimal_threshold)
## pamr.plotcv(cvmodel)

set.seed(12345)

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

```
en_optimal_lambda <- fit$lambda[which.min(fit$cvm)]
en_optimal_size <- fit$nzero[which.min(fit$cvm)]
penalty <- strsplit(fit$name, " ")[[1]][2]

en_class_error <- 1 - (sum(predict(fit, t(x_test), type="class") == y_test) /
                    length(y_test))
cat(paste("Penalty", penalty))
cat(paste("Lambda:", en_optimal_lambda))
cat(paste("Size:", en_optimal_size))
cat(paste("Classification Error:", en_class_error))
set.seed(12345)

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

svm_optimal_size <- fit@nSV
svm_class_error <- 1 - (sum(predict(fit, t(x_test)) == y_test) / length(y_test))
cat(paste("Size:", svm_optimal_size))
cat(paste("Classification Error:", svm_class_error))
table_data <- data.frame(NSC=c(nsc_optimal_size, percent(nsc_class_error)),
                         EN=c(en_optimal_size, percent(en_class_error)),
                         SVM=c(svm_optimal_size, percent(svm_class_error)),
                         row.names=c("Size", "Class. Err."))
kable(table_data, format="latex", format.args=list())

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)
length(result$features)
cat("Top 10 features")
cat(paste(result$features[1:10], collapse='\n' ) )
ggplot() +
    ylab("P-Value") + xlab("Index") +
    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

```
library(ggplot2)
library(reshape2)

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) {
    exp(-(x / h))
}

euclideansq_d <- function(x, xi) {
    x <- t(as.matrix(x))
    xi <- as.numeric(xi)
    colSums((x - xi)^2)
}

SVM <- function(sv, xi) {
    h <- 1
    b <- 0

    x <- sv[, -ncol(sv)]
    t <- sv[, ncol(sv)]

    k <- gaussian_k(euclideansq_d(x, xi), h)

    sum(t * k) + b
}

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

run_BOSVM <- function(data, beta, M, N) {
    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
    }
    list(errorrate=errorrate, sv=sv)
}

N <- 500
result1 <- run_BOSVM(data=spam, beta=0, M=500, N=N)
cat(paste("Number of Support Vectors:", length(result1$sv)))
result2 <- run_BOSVM(data=spam, beta=-0.05, M=500, N=N)
cat(paste("Number of Support Vectors:", length(result2$sv)))
result3 <- run_BOSVM(data=spam, beta=0, M=20, N=N)
cat(paste("Number of Support Vectors:", length(result3$sv)))
result4 <- run_BOSVM(data=spam, beta=-0.05, M=20, N=N)
cat(paste("Number of Support Vectors:", length(result4$sv)))
plot_data <- data.frame(
    x=1:length(seq(from=1, to=N, by=10)),
    res1=result1$errorrate[seq(from=1, to=N, by=10)],
    res2=result2$errorrate[seq(from=1, to=N, by=10)],
    res3=result3$errorrate[seq(from=1, to=N, by=10)],
    res4=result4$errorrate[seq(from=1, to=N, by=10)])

plot_data <- melt(plot_data, id="x", value.name="ErrorRate",
                  variable.name="Res")

ggplot(plot_data) +
    geom_line(aes(x=x, y=ErrorRate, color=Res))
```