Introduction to Machine Learning

Lab 3 Block 2

Anton Persson, Emil Klasson Svensson, Mattias Karlsson, Rasmus Holm

2016-12-20

Contents

Assignment 1

In this assignment we have used the data.csv 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 we want to predict whether the e-mail is a conference e-mail or something else. In order to do so we divided the data set into train and test sets corresponding to 70% (44 obs.) respective 30% (20 obs.) of the data.

1

In this first part we are supposed to use the nearest shrunken centroid method on the training set where the threshold is chosen by 10-fold cross-validation. Below are the results and we can see that the optimal threshold was 2.8 resulting in 12 non-zero features and a classification error of 10%. The words in the list of the top 10 most contributing ones are reasonable to discriminate conference e-mails from other e-mails.

```
#> Threshold: 2.8
#> Size: 12
#> Classification Error: 0.1
#> 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.

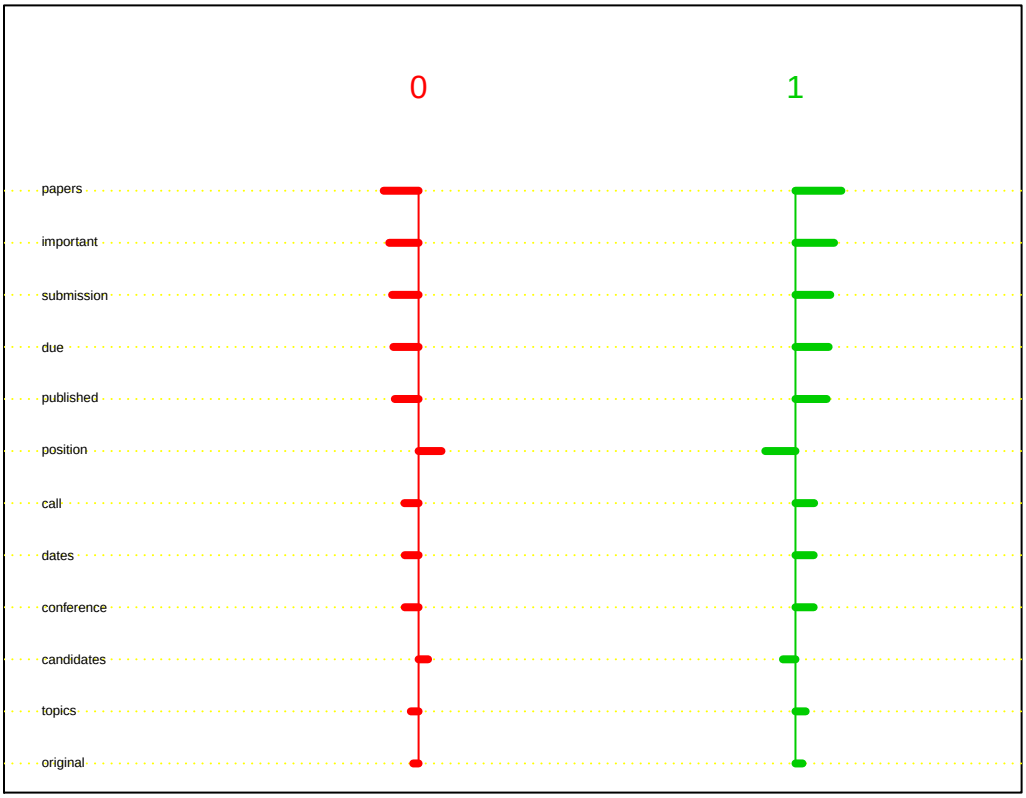


Figure 1: Centroid plot.

2

Elastic Net

Elastic net is another method for feature extraction and we have let cross-validation choose the penalty measurement for us. 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, we 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 a classification error point of view in the table below the support vector is better than both elastic net and nearest shrunken centroid, but in this case we only have 20 observations in the test set which means

+5% error rate is just an additional misclassification. Given the information we have we would prefer the nearest shrunken centroid since it reduces the feature space from 4702 to just 12 features with an additional misclassification compared to the support vector machine which uses all features. The elastic net have almost 3 times the number of features as nearest shrunken centroid with an additional misclassification so it is clearly a worse model both in terms of complexity and error rate.

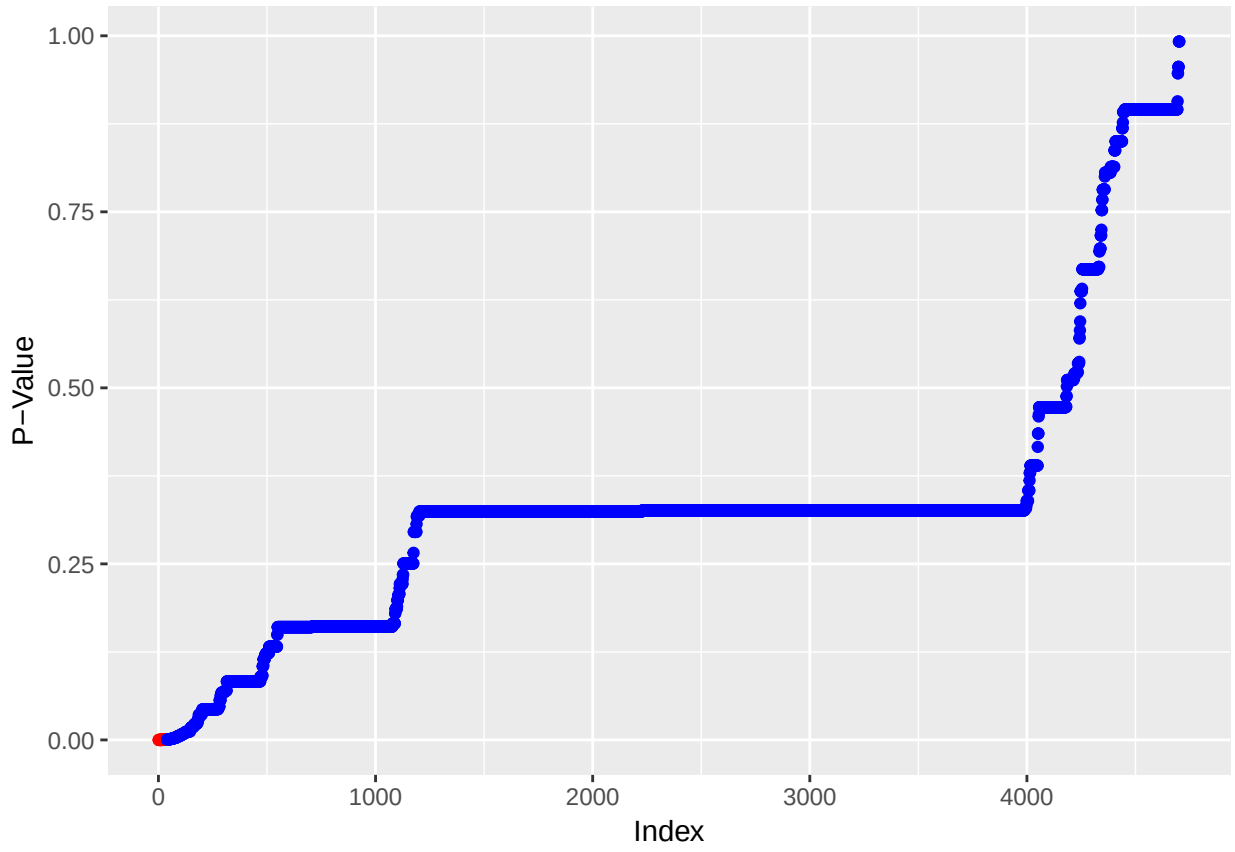
	NSC	EN	SVM
Features	12	38	4702
Class. Err.	10%	15%	5%

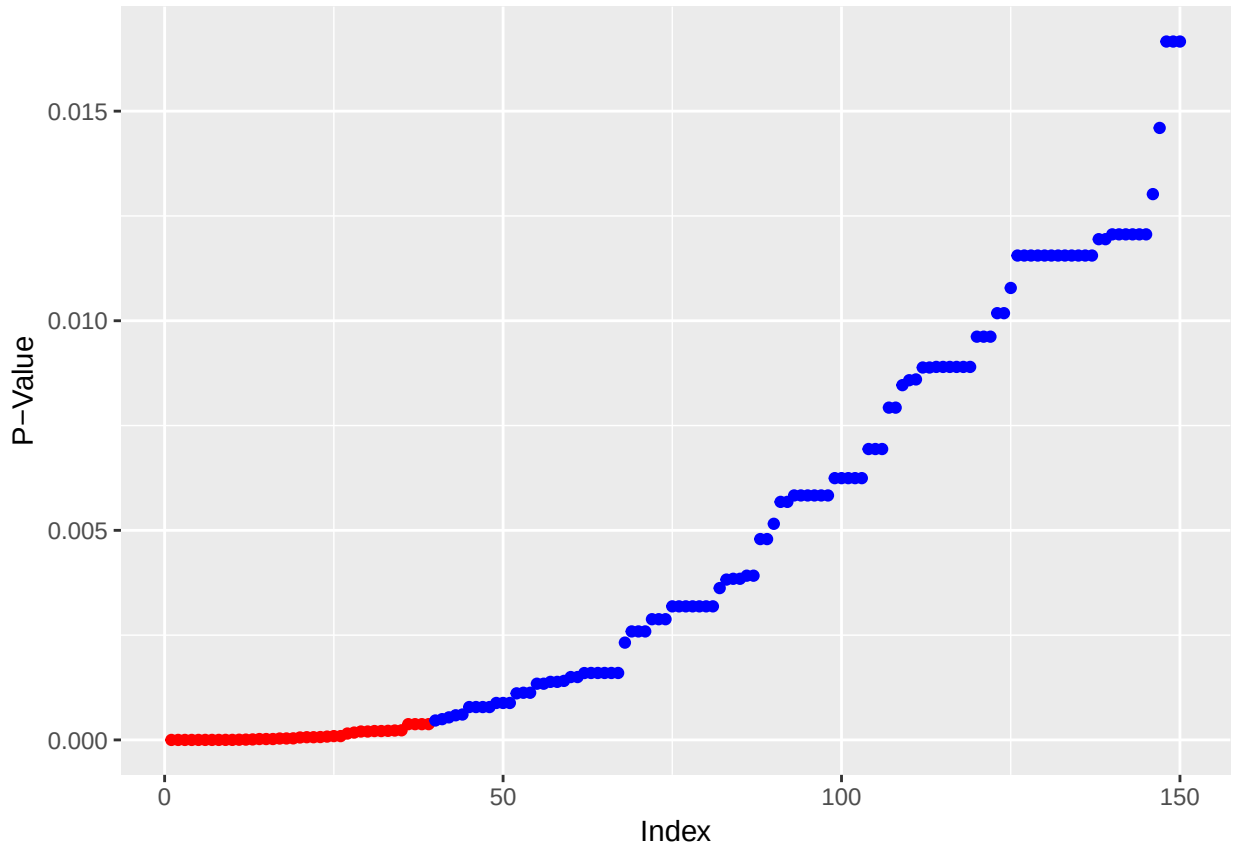
3

In total there were 39 rejected hypothesis by the Benjamini-Hochberg (BH) 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 seen below.

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

Below are plots of the p-values for each feature in sorted order and the second plot just shows the lowest 150 p-values. The red points are those features where the null hypotheses was rejected, i.e. those are statistically significant in explaining the *Conference* variable given $\alpha = 0.05$ according to the rules of the BH-method which put a stricter requirement on the most significant features.


1.00
0.75
0.50
0.25
0.00
P−Value
0
1000
2000
3000
4000
Index


0.015
0.010
0.005
0.000
P−Value
0
50
100
150
Index

Assignment 2

In this assignment we implemented a version of the budget online support vector machine (BOSVM) and it has parameters controlling the maximum number of support vectors that should be used in the prediction denoted M and the fault tolerance β. A faulty prediction means that the sign of the prediction does not correspond with the sign of the true label assuming a binary classification $\{-1, 1\}$. We ran the BOSVM for 500 iterations, i.e. samples, with four different settings which can be seen below with the corresponding number of final support vectors in the model.

```
#> Beta = 0, M = 500
#> Number of Support Vectors: 106

#> Beta = -0.05, M = 500
#> Number of Support Vectors: 48

#> Beta = 0, M = 20
#> Number of Support Vectors: 20

#> Beta = -0.05, M = 20
#> Number of Support Vectors: 20
```

Figure 2 shows how the error rate differ for the various settings. We can see that setting $\beta = 0$ is better in both cases which does make sense since we will adjust our model every misclassification and for the case where $M = 500$ it will start overfit the data. Using a $\beta < 0$ means we accept some misclassification without adjusting our model and this result in slightly higher error rate but with far fewer support vectors as we can see above when $M = 500$, 48 compared to 106.

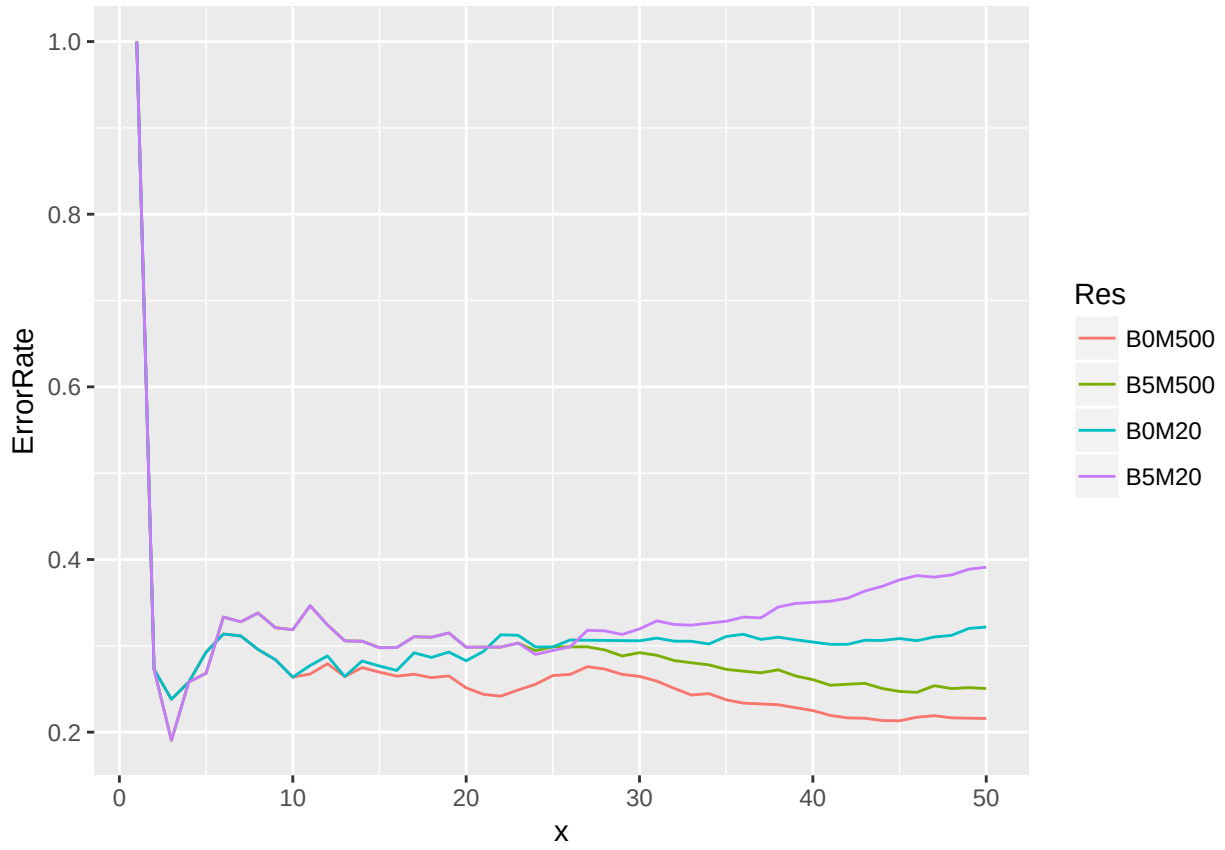


Figure 2: The error rate for different β and maximum number of support vectors, M. B0 means $\beta = 0$ and B5 means $\beta = -0.05$.

So what we can say is that when using $M = 500$ the β controls how much the model fits the data and therefore using $\beta = 0$ are more prone to overfitting than $\beta = -0.05$ and gets a lower error rate. BOSVM with $M = 20$ and $\beta = -0.05$ has a slightly smoother error curve than with $\beta = 0$ because it does not change the support vectors for every faulty prediction. This is due to stabilization and the set of support vectors are more general and have greater predictive power. One problem with using $M = 20$ and $\beta = 0$ is that we have set the total amount of support vectors to a low value and every misclassification adds a new support vector which means that the calculation for removing a current support vector is executed many times and it is the most expensive operation in the algorithm hence slows down the execution time significantly.

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 <- as.matrix(train[, -ncol(data)])
y <- train[, ncol(data)]

x_test <- as.matrix(test[, -ncol(data)])
y_test <- test[, ncol(data)]

set.seed(12345)

nsc.x <- t(x)
nsc.x_test = t(x_test)

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

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

nsc_class_error <- 1 - (sum(pamr.predict(model, nsc.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)

set.seed(12345)
```

```
alpha <- 0.5
fit <- cv.glmnet(x=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)]
en_penalty <- strsplit(fit$name, " ")[[1]][2]
en_class_error <- 1 - (sum(predict(fit, x_test, type="class") == y_test) /
                    length(y_test))
cat(paste("Penalty", en_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=x, y=y, kernel="vanilladot",
            type="C-svc", scale=FALSE)

svm_optimal_size <- fit@nSV
svm_class_error <- 1 - (sum(predict(fit, 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(ncol(data) - 1, percent(svm_class_error)),
                         row.names=c("Features", "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)
rejected <- 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 / (2 * 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(data, sv) {
    which.max(lapply(sv, function(m) {
        obs <- data[m,]
        x <- obs[, -ncol(obs)]
        t <- obs[, ncol(obs)]
        y <- SVM(data[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)])
        classification <- data[i, ncol(data)] * predicted

        if (classification <= beta) {
            sv <- c(sv, i)

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

        if (classification < 0) {
            errors <- errors + 1
        }

        errorrate[i] <- errors / i
    }
    list(errorrate=errorrate, sv=sv)
}

N <- 500
result1 <- run_BOSVM(data=spam, beta=0, M=500, N=N)
cat("Beta = 0, M = 500")
cat(paste("Number of Support Vectors:", length(result1$sv)))
result2 <- run_BOSVM(data=spam, beta=-0.05, M=500, N=N)
cat("Beta = -0.05, M = 500")
cat(paste("Number of Support Vectors:", length(result2$sv)))
result3 <- run_BOSVM(data=spam, beta=0, M=20, N=N)
cat("Beta = 0, M = 20")
cat(paste("Number of Support Vectors:", length(result3$sv)))
result4 <- run_BOSVM(data=spam, beta=-0.05, M=20, N=N)
cat("Beta = -0.05, M = 20")
cat(paste("Number of Support Vectors:", length(result4$sv)))
plot_data <- data.frame(
    x=1:length(seq(from=1, to=N, by=10)),
    B0M500=result1$errorrate[seq(from=1, to=N, by=10)],
    B5M500=result2$errorrate[seq(from=1, to=N, by=10)],
    B0M20=result3$errorrate[seq(from=1, to=N, by=10)],
    B5M20=result4$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))
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):150),
                               y=result$pvalues[!result$mask][1:(150 - rejected)]),
               aes(x=x, y=y), col="blue")
```

Contributions

We divided the work into two parts and discussed/compiled the results in pairs. Then we all discussed our findings together as a whole group and checked that everyone had similar/understood the results.