Lab3 block2 732A95

Anton Persson antpe404

16 december 2016

Assignment 1 High dimensional methods

The first assignment is about different methods to deal with wide data.

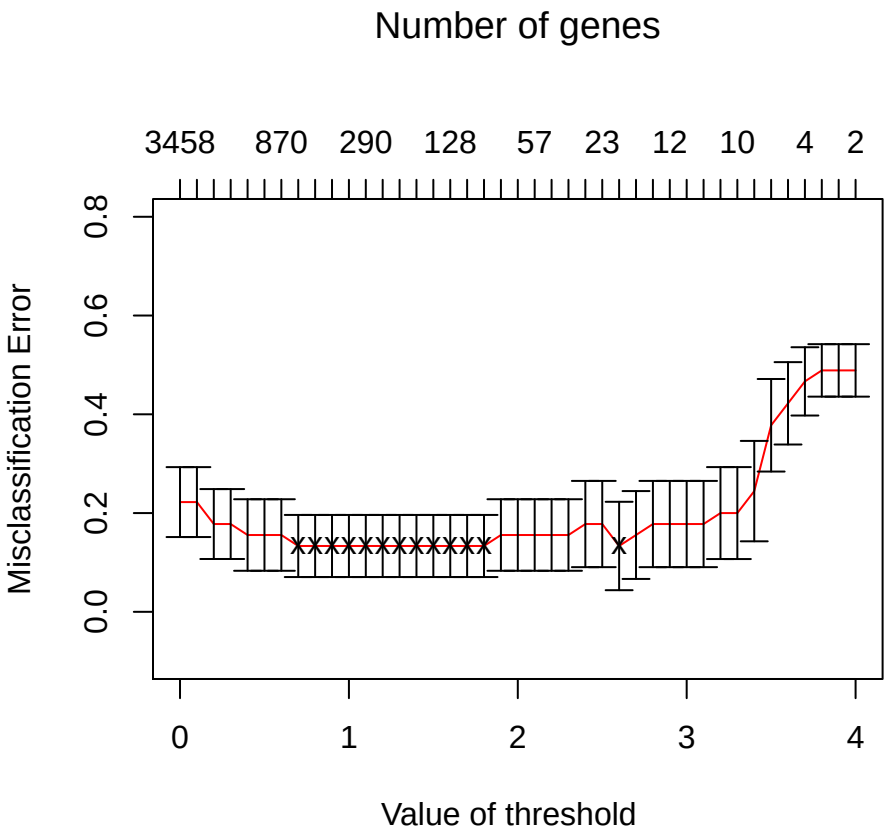
Assignment 1.1

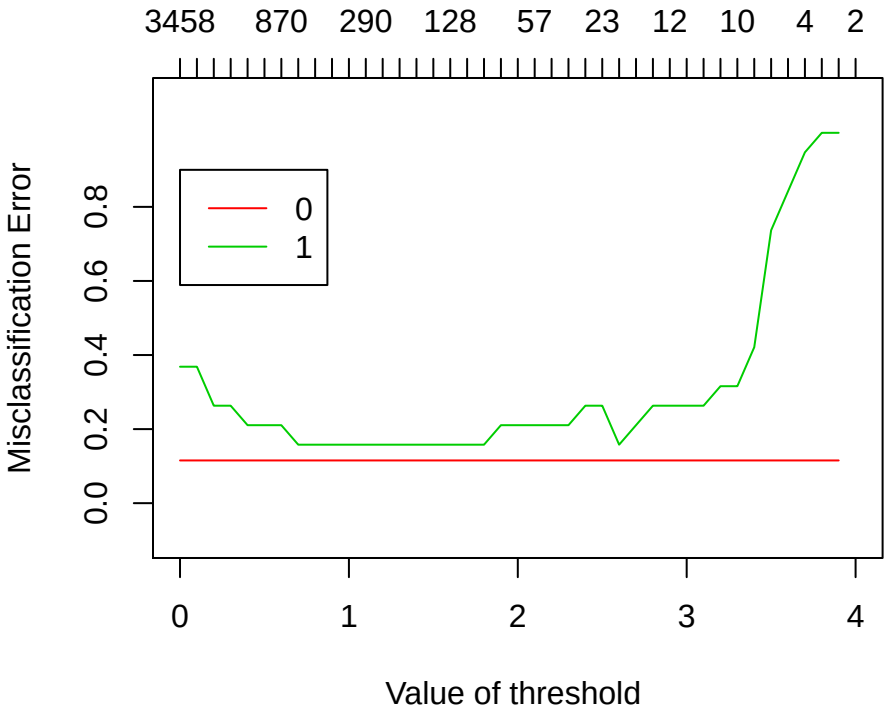
I divided the data into training and test according to the instruction. The nearest shrunken centroid classification was done and the required results are presented below.

```
## Call:
## pamr.cv(fit = model, data = mydata)
##    threshold nonzero errors
## 1  0.0       3458    10
## 2  0.1       3428    10
## 3  0.2       3110    8
## 4  0.3       3042    8
## 5  0.4       3025    7
## 6  0.5       1977    7
## 7  0.6        870    7
## 8  0.7        850    6
## 9  0.8        673    6
## 10 0.9        671    6
## 11 1.0        295    6
## 12 1.1        290    6
## 13 1.2        269    6
## 14 1.3        234    6
## 15 1.4        154    6
## 16 1.5        151    6
## 17 1.6        128    6
## 18 1.7        100    6
## 19 1.8         97    6
## 20 1.9         73    7
## 21 2.0         64    7
## 22 2.1         57    7
## 23 2.2         42    7
## 24 2.3         37    7
## 25 2.4         35    8
## 26 2.5         23    8
## 27 2.6         21    6
## 28 2.7         20    7
## 29 2.8         14    8
## 30 2.9         12    8
## 31 3.0         11    8
## 32 3.1         10    8
## 33 3.2         10    9
```

```
## 34 3.3         10    9
## 35 3.4          9    11
## 36 3.5          4    17
## 37 3.6          4    19
## 38 3.7          4    21
## 39 3.8          4    22
## 40 3.9          3    22
## 41 4.0          2    22
```

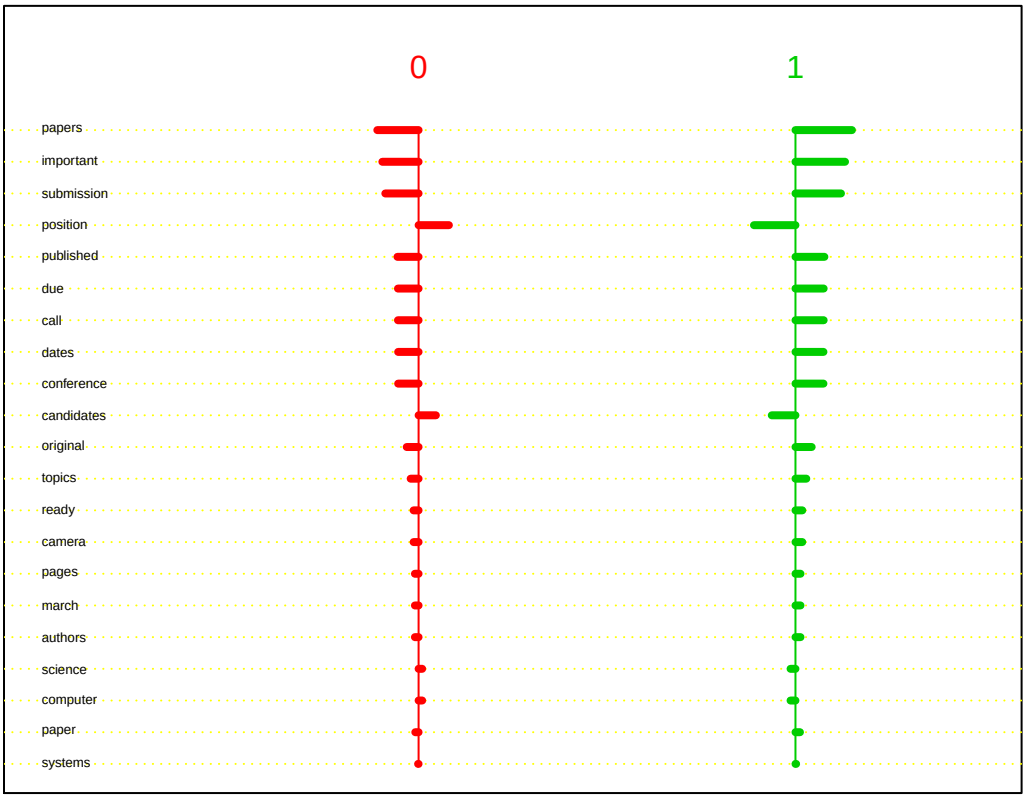
From the table above, I conclude that a threshold between 0.7-1.8 or 2.6 generates the lowest error. I choose 2.6 as threshold because of simplicity. This will generate 21 selected features, notated as *nonzero* in the output above. The figure below visualizes the table above. The plot on top in the figure below says that above mentioned thresholds minimizes the error.


Number of genes
3458 870 290 128 57 23 12 10 4 2
Misclassification Error
0.0 0.2 0.4 0.6 0.8
0 1 2 3 4
Value of threshold


3458 870 290 128 57 23 12 10 4 2
Misclassification Error
0.0 0.2 0.4 0.6 0.8
0
1
0 1 2 3 4
Value of threshold

The ten most important features for the model with 2.6 as threshold is visualized in a cendriod plot below.

```
## 1
```



The plot above lists the most contributing features and it's impact on the two levels of response, 0 and 1 in this case. 1 means that the observed mail was classified as a "announces of conference", and 0 means that the mail wasn't. Thus, if a word contribute highly to classify a mail as 1, it's low for class label 0. For example, the word *paper* contribute to define a mail as label 1, and therefore, if it appears in a mail it's less likely to get classfied as a 0. The default plot in pamr doesn't have a brilliant layout, why it's kind of hard to read which variables are actually chosen. For clarity I decided to list them in a more proper way. The ten most contributing features are thus listed below.

```
## papers
## important
## submission
## position
## published
## call
## due
## conference
## dates
## candidates
```

I'd say that the words are reasonable. I can see why you'd mention words like *paper*, *submission*, *candidates*, *published*, *dates* and *conference* in a mail about conferences. Finally, the test error is presented below.

```
## [1] 0.1052632
```

Assignment 1.2

In this assignment I'm supposed to compute the error rate and number of contributing features for two more methods, elastic net and support vector machine. I start off with elastic net.

Assignment 1.2a

The instructions defines the type of response and value of α for me. I do use the function cv.glmnet to decide penalty by cross validation. The selected penalty, the number of features and test error rate is presented below.

```
## $selected_penalty
##            deviance
## "Binomial Deviance"
##
## $test_error_rate_elastic
## [1] 0.1578947
##
## $number_of_features_elastic
## [1] 12
```

Assignment 1.2b

The test error and number of contributing features for a SVM with *vanilladot* kernel are presented below.

```
##  Setting default kernel parameters

## $test_error_rate_svm
## [1] 0.05263158
##
## $number_of_features_svm
## [1] 4702
```

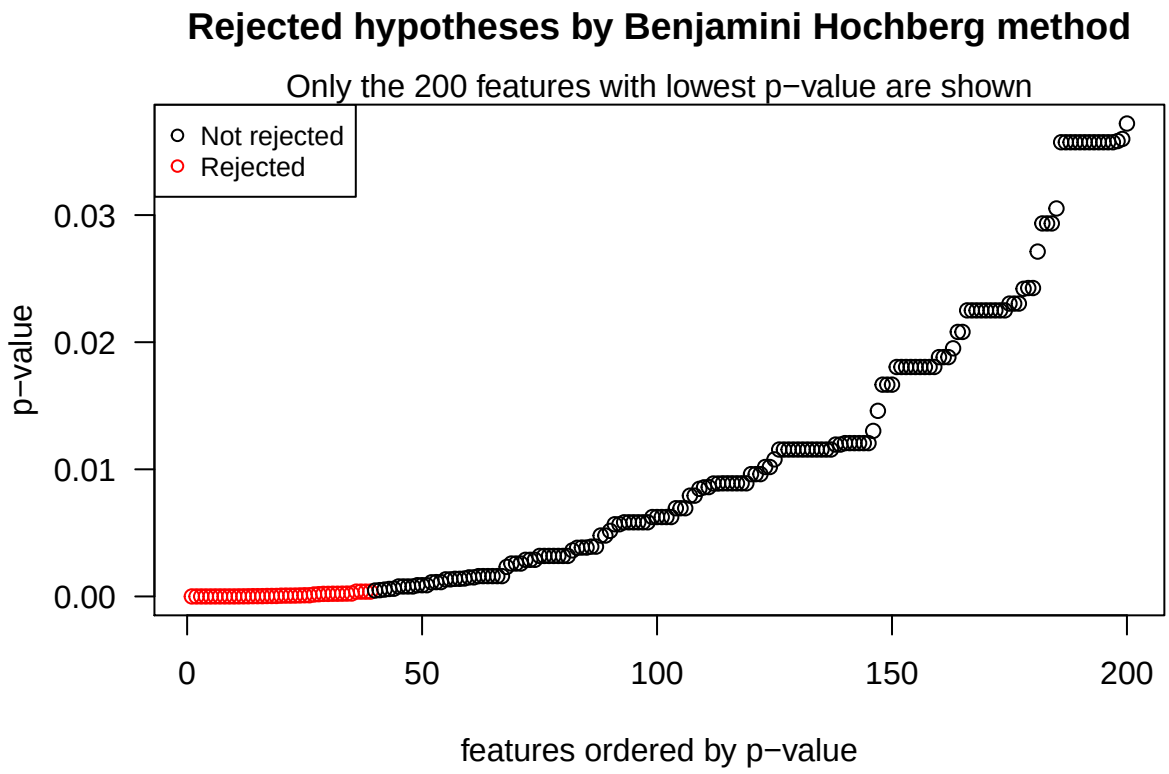
Finally I compare the three methods used above, nearest shrunken centroid method, elastic net and support vector machine, by arranging a table with all the results. Watch it below!

```
##                         NSC Elastic       SVM
## Number of features 21.0000 12.0000 4702.0000
## test_error_rate     0.1053  0.1579    0.0526
```

From the table I conclude that I'd probably choose the NSC method. NSC has one more missclassification than the SVM, but reduces the number of features from 4702 to 21. The SVM method has the lowest error rate, but uses all features. SVM doesn't reduce the number of features, but try to reduce the number observations as much as possible. In this case it only reduces the number of rows from 45 to 44 though, since I have 45 rows in my training set and 44 support vectors in my ksvm-modeln. That's not much of a reduction.

Assignment 1.3

In this assignment I implemented the Benjamini Hochberg method on the complete original data, not the training set only. To begin with, the result is visualized by the plot below.



Notice that in the plot above, only the first 200 features of the ordered data frame are plotted. That means that the features visualized in the plot only corresponds to the about 4 % of all features in the data. I did this because of off the 4702 features tested, only 39 gets rejected. Those 39 rejected features are thus the features that are most useful for describing the target *Conference*.

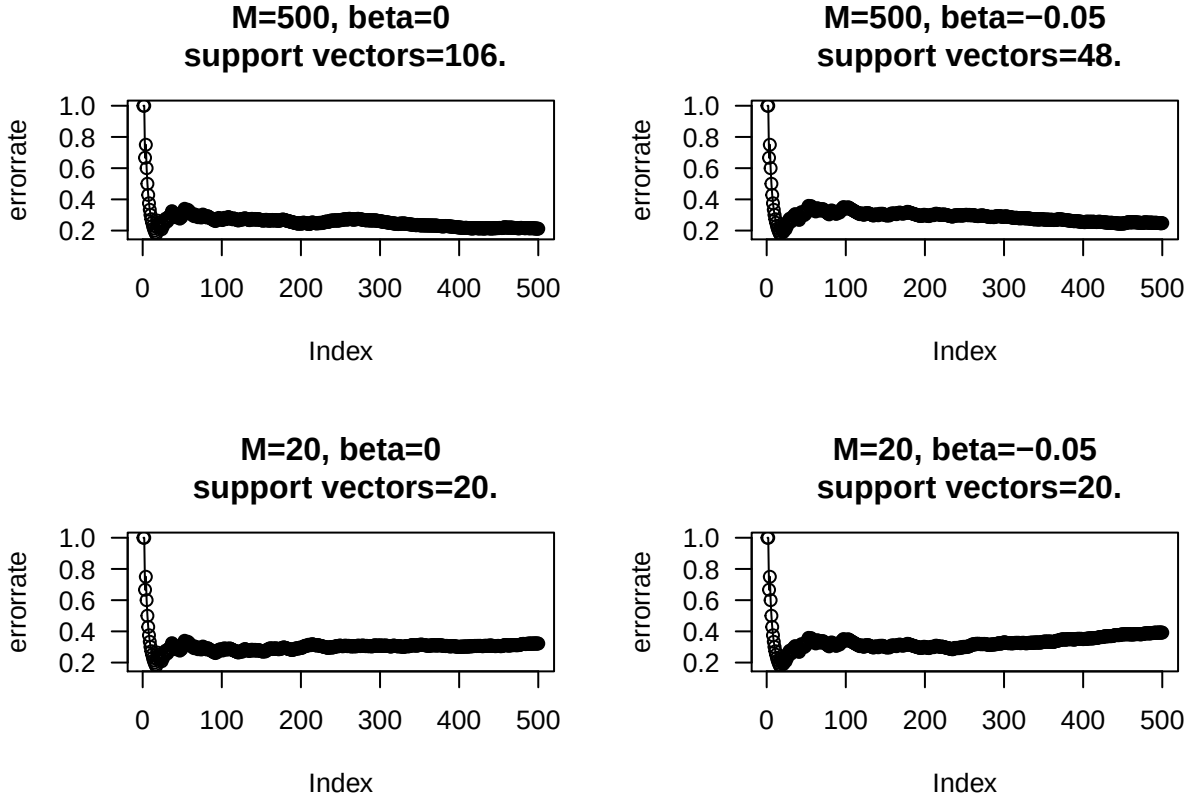
The 39 features that were rejected, i.e. gets chosen, are shown below. Notice that the words that have occured as important features in earlier methods are among the rejected features for the Benjamini Hochberg method as well.

```
## papers
## submission
## position
## published
## important
## call
## conference
## candidates
## dates
## paper
## topics
## limited
## candidate
```

```
## camera
## ready
## authors
## phd
## projects
## org
## chairs
## due
## original
## notification
## salary
## record
## skills
## held
## team
## pages
## workshop
## committee
## proceedings
## apply
## strong
## international
## degree
## excellent
## post
## presented
```

Assignment 2 Online learning

I used the pseudo code in the slides to implement the budget online SVM. I used parts of the template in the instruction as well. The input M defines the maximum number of support vectors, while the input β defines the fault tolerance. The four required plots of the error rates as a function of the number of training points are shown below.



The call with M=20, beta=0 is the slowest because it generates the most computation. The higher the β is the more iterations are true in step 5 of the pseudo code, thus the more computation needs to be done. That also applies for M=20 compared to M=500. When M=500, the $|S|$ (number of support vectors) never gets bigger than M when I run 500 training points. That means that the exclusion of the least important support vector in step eight in the pseodo code is never computed.

Using $\beta = -0.05$ means that I tolerate some missclassification without trying to improve the model. When $\beta = 0$ the model will be updated every time an error is made. Practically this means that I do get slighly worse error rates when $\beta = -.0.5$, but it reduces the number of support vectors a lot. The two plots on top in the figure above are run with $M = 500$, and from those two I conclude that the error rate is slightly better for the one where $\beta = 0$ but it also uses 106 support vectors compared to only 48 when $\beta = -0.05$.

As I understand it, $\beta = 0$ will always generates the best results, since it implies the most improvements of the model, but it may come with the risk of overfitting.

Appendix

The code used for solving this lab and to create this report is presented below.

```
#Assignment 1
data<-read.csv2("data/data.csv", header=T, sep=";")
#Wide data.
set.seed(12345)
data<-data[sample(1:nrow(data)), ] #Shufflar raderna
data$Conference<-as.factor(data$Conference)
train<-data[1:45, ]
test<-data[46:64, ]

#1.1

library(pamr)
which_col<-which(colnames(train)=="Conference")
x<-t(train[,-which_col]) #Alla utom Conf
y<-train[, which_col] #endast Conf
#head(as.character( 1:nrow(x) ))
mydata<-list(x=x,y=y,geneid=as.character(1:nrow(x)), genenames=rownames(x))
model<-pamr.train(mydata,threshold=seq(0,4, 0.1))

cv_model<-pamr.cv(model,mydata)
#cv_model$error#.13 lowest, fr 0.7-1.8 och 2.6

print(cv_model) #Here kan man se samma, 0.7-1.8 eller 2.6

pamr.plotcv(cv_model)

chosen_model<-pamr.train(mydata,threshold=2.6) #Väljer 2.6 för simplicity.
pamr.plotcen(chosen_model, mydata, threshold=2.6)

cv_vars<-pamr.listgenes(chosen_model,mydata,threshold=2.6)
#cv_vars[,1] #Variablerna
my_variables_index2<-as.numeric(cv_vars[,1])
my_variables2<-colnames(data)[my_variables_index2]

cat(paste(my_variables2[1:10], collapse="\n"))

x_test<-t(test[,-which_col]) #Alla utom Conf
y_test<-test[, which_col] #endast Conf
my_testdata<-list(x=x_test,y=y_test,
                  geneid=as.character(1:nrow(x)), genenames=rownames(x))

conf_mat_NSC<-table(preds=pamr.predict(chosen_model,
                  my_testdata$x,threshold = 2.6),truth=my_testdata$y)

number_of_features_NSC<-length(my_variables2)
test_error_rate_NSC<-((conf_mat_NSC[1,2]+conf_mat_NSC[2,1])/
  sum(conf_mat_NSC))
test_error_rate_NSC

#1.2a
```

```r
library(glmnet)

train_x<-as.matrix(train[,-which_col])
train_y<-as.factor(train[,which_col])
test_x<-as.matrix(test[, -which_col])
test_y<-as.factor(test[,which_col])

set.seed(12345)
elastic<-cv.glmnet(x=train_x, y=train_y, alpha=.5, family="binomial")
#plot(elastic)
selected_penalty_elastic<-elastic$name

elastic_coe<-coefficients(elastic)
elastic_coeff<-elastic_coe[,1]
elastic_coef<-elastic_coeff[elastic_coeff!=0] #Tar ut de som inte är 0.

elastic_coefficients<-as.data.frame(cbind(coef=as.numeric(elastic_coef),
                                          variable=names(elastic_coef)))
#12 variabler med coef skild fr 0.
fitted<-predict(elastic, newx = test_x,
      s = elastic$lambda.1se, type="class") #default s.


fitted<-as.numeric(fitted)

number_of_features_elastic<-nrow(elastic_coefficients)-1
conf_mat_elastic<-table(preds=fitted, truth=test_y)
test_error_rate_elastic<-((conf_mat_elastic[1,2]+conf_mat_elastic[2,1])/
  sum(conf_mat_elastic))

reslist<-list(selected_penalty=selected_penalty_elastic,
 test_error_rate_elastic=test_error_rate_elastic,
 number_of_features_elastic=number_of_features_elastic)

reslist

#1.2b
library(kernlab)
set.seed(12345)
my_svm<-ksvm(x=train_x, y=train_y, kernel="vanilladot",
             scale=FALSE, type="C-svc")

fitted_svm<-predict(my_svm, test_x)
conf_mat_svm<-table(preds=fitted_svm, truth=test_y)
#conf_mat_svm

test_error_rate_svm<-(conf_mat_svm[1,2]+conf_mat_svm[2,1])/sum(conf_mat_svm)

number_of_features_svm<-ncol(train_x)

reslist2<-list(test_error_rate_svm=test_error_rate_svm,
               number_of_features_svm=number_of_features_svm)
```

```
reslist2

#######Comparison, last thing in ass 1.2###############
comparison_table<-data.frame(cbind(rbind(number_of_features_NSC,test_error_rate_NSC),
                                   rbind(number_of_features_elastic, test_error_rate_elastic),
rbind(number_of_features_svm, test_error_rate_svm)))

colnames(comparison_table)<-c("NSC", "Elastic", "SVM")
rownames(comparison_table)<-c("Number of features", "test_error_rate")
#comparison_table[1,]<-round(comparison_table[1,], 0)
comparison_table<-round(comparison_table, 4)
comparison_table

#1.3

alpha<-0.05
pvalues<-data.frame(matrix(NA, nrow = ncol(data)-1, ncol = 2))
for (i in 1:(ncol(data)-1)){
  pvalues[i,2]<- t.test(data[,i]~data[, which_col], alternative="two.sided")$p.value
  pvalues[i,1]<-paste(colnames(data)[i])
}
colnames(pvalues)<-c("variable", "pvalue")
pvalues<-pvalues[order(pvalues$pvalue), ] #sortar dem på pvalue.

#plot(x=1:nrow(pvalues), y=pvalues$pvalue,
#xlab="features ordered by p-value", ylab="p-value", las=1,
#main="All features and their pvalues computed with t.test()")

pvalues$Loss<-NA

for (j in 1:nrow(pvalues)){
  pvalues[j,3]<-(alpha*j)/nrow(pvalues)
}

pvalues$selected<-"Not rejected"
pvalues[pvalues$pvalue<pvalues$Loss,4]<-"Rejected"
#Sätter de med lägre p än Loss till 1.
pvalues$selected<-as.factor(pvalues$selected)

plot(x=1:200, y=pvalues[1:200,2], col=pvalues$selected,
     xlab="features ordered by p-value",
     ylab="p-value", las=1,
     main="Rejected hypotheses by Benjamini Hochberg method")
legend('topleft', legend = levels(pvalues$selected),
       col = 1:2, cex = 0.8, pch = 1)
mtext("Only the 200 features with lowest p-value are shown")

selected_features_BH<-pvalues[pvalues$pvalue<pvalues$Loss, 1]
cat(paste(selected_features_BH, collapse = "\n"))

#Assignment 2
spam<-read.csv2("data/spambase_lab3bl2.csv", sep=";", header=T)
spam$Spam[spam$Spam==0]<--1 #Convertar för att underlätta SVM:s.
```

```
set.seed(1234567890)
ind <- sample(1:nrow(spam))
spam <- spam[ind,c(1:48,58)] #Shufflar och tar bort några cols.

#Defining gaussian kernel
gaussian_k <- function(x, h) { # Gaussian kernel
  ans<-exp(-(x**2)/(2*h**2))
  return(ans)
}

#defining step 8 in budget online SVM pseudo code
step8_func<-function(sv, h){
  b<-0
  res<-vector(length=length(sv))
  index<-1
  for(m in sv){
    x_m<-spam[m, -49]
    t_m<-spam[m, 49]
    dis<-as.matrix(dist(rbind(x_m,x_m))) #sista delen av step 8
    last_part<-sum(t_m*gaussian_k(x=dis[-1,1], h=h))
    #sista delen av step8, med dis mellan samma vector
    #Nedan ist för step4[m] eftersom uppdateras
    x_sv<-spam[sv, -49]
    t_sv<-spam[sv, 49]
    distance<-as.matrix(dist(rbind(x_m, x_sv), method="euclidean"))
    yxm<-sum(t_sv*gaussian_k(x=distance[-1,1], h=h)+b) #step4
    #yxm<-step4[m]#Motsvarar y(x_m) in step8, calculated in last loop
    res[index]<-t_m*(yxm-last_part)
    index<-index+1
  }
  #return(res)
  return(which.max(res)) #tar ut den minst viktiga, i.e. max.
}

#defining the whole function that takes M and beta as input.
SVM<-function(sv=c(1), H=1, b=0, N=500, M, beta){
  errors <- 1
  errorrate <- vector(length = N)
  errorrate[1] <- 1

  for(i in 2:N) {
    x_i<-spam[i, -49] #step3
    t_i<-spam[i, 49] #step3
    x_m<-spam[sv, -49] #x_m
    t_m<-spam[sv, 49]
    distance<-as.matrix(dist(rbind(x_i, x_m), method="euclidean"))
    yx_i<-sum(t_m*gaussian_k(x=distance[-1,1], h=H)+b) #step4

     if (t_i*yx_i<=beta){ #step5
      sv<- c(sv, i) #step6

      #ignoring step7 since always =1
      if (length(sv)>M){ #step 8
```

```
        sv <- sv[-step8_func(sv=sv, h=H)]
      }
     }
    if (t_i*yx_i<=0){
    errors<-errors+1
    }
    errorrate[i] <- errors / i
  }
  number_of_sv<-length(sv)
  plot(errorrate, main=paste("M=", M, ", beta=", beta,
  "\n support vectors=", number_of_sv,".", sep=""),
       type="o", las=1)
}

par(mfrow=c(2,2))
SVM(M=500, beta=0)
SVM(M=500, beta=-.05)
SVM(M=20, beta=0)
SVM(M=20, beta=-.05)
```