Different classification models on USPS data

Joaquim Lyrio - UNI:jc4637

2/26/2017

Support Vector Machines (SVM)

```r
# load necessary packages
library(e1071)
library(ggplot2)

# read input data
uspsdata <- read.table("../data/uspsdata.txt")
uspscl   <- read.table("../data/uspscl.txt")
```

First, let's randomly select 20% of the data and set it aside as our test set.

```r
# define number of folds
nFolds <- 5

# generate vector of random indexes
index  <- rep( seq( 1, nFolds), nrow(uspscl)/nFolds )
set.seed(234)
index  <- sample( index, replace = FALSE )

# subset test data
test.data <- uspsdata[ which( index == 5 ) , ]
test.cl   <- uspscl[ which( index == 5 ) , ]

# subset train data
rest.data <- uspsdata[ which( index != 5 ) , ]
rest.cl   <- uspscl[ which( index != 5 ) , ]
```

Model 1: Linear SVM with soft margin

Now, we have to train a linear SVM with soft margin. To find the optimal margin parameter, we will use cross-validation. For doing so, since we have already separated our test set, we just have to divide the remaining data into training set and validation set. We wil perform a 10-fold cross validation.

```r
# define number of folds
nFolds <- 10

# generate vector of random indexes
index  <- rep( seq( 1, nFolds), nrow(rest.data)/nFolds )
set.seed(321)
index  <- sample( index, replace = FALSE )

# define range of costs on which to iterate
range.cost <- c( 10^(-4), 0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 1, 5 )
```

```r
# initialize matrix to store cross-validated errors
matError <- matrix(NA, nrow = nFolds, ncol = length(range.cost) )

# for each fold
for( iFold in 1:nFolds )
{
  # separate data into train and validation sets
  train.data  <- rest.data[ which( index != iFold ) , ]
  train.cl    <- as.factor(rest.cl[ which( index != iFold ) ])

  validation.data  <- rest.data[ which( index == iFold ) , ]
  validation.cl    <- rest.cl[ which( index == iFold )  ]

  icol   <- 1
  for( i.cost in range.cost )
  {
    # fit SVM model
    fit.svm <- svm( x = train.data, y = train.cl, kernel = "linear", cost = i.cost )

    # predict in validation dataset
    class.pred <- predict( fit.svm, validation.data )

    # calculate error in validation data for the fit
    matError[ iFold, icol ] <- sum( class.pred != validation.cl )/length(class.pred)

    icol <- icol + 1

  }
}

colnames(matError) <- paste(range.cost)
rownames(matError) <- paste(1:nFolds)
```

Let's find the parameter which attains the smallest cross-validation error.

```r
which.min(apply(matError,2,mean))
```

```
## 0.001
##     2
```

As we can see, the margin parameter that minimizes the prediction error on the validation set is 0.001. Now that we have found the optimal parameter, we can train our model using the whole data (train + validation) and estimate the prediction error on the test set.

```r
# fit model
fit1.svm <- svm( x = rest.data, y = as.factor(rest.cl), kernel = "linear", cost = 0.001 )

# predict in test dataset
class.pred.1 <- predict( fit1.svm, test.data )

# calculate prediction error in test data
misc.rate.fit1 <- sum( class.pred.1 != test.cl )/length(class.pred.1)
```

This way, the misclassification rate for the model is given by:

```r
misc.rate.fit1
```

```
## [1] 0.025
```

or, 2.5%.

Model 2: SVM with soft margin and Kernel RBF

Now, let's train an SVM with soft margin and RBF Kernel. Now, in order to fit the best model, we have to use cross-validation to decide the optimal values of the bandwidth and soft-margin parameters.

```r
nFolds <- 10
index  <- rep( seq( 1, nFolds), nrow(rest.data)/nFolds )
set.seed(123)
index  <- sample( index, replace = FALSE )

range.bw   <- c( 0.0005, 0.005, 0.01, 0.015, 0.02, 0.025, 0.03 )
range.cost <- c( 0.1, 0.25, .5, 0.75, 0.9, 1, 1.2, 1.4 )

# initialize array to store cross-validated errors
error <- array(NA, dim = c( nFolds, length(range.cost), length(range.bw) ) )

#i.cost <- range.cost[1]; i.bw <- range.bw[1]; iFold <- 1;
for( iFold in 1:nFolds )
{
  # separate data into train and validation sets
  train.data  <- rest.data[ which( index != iFold ) , ]
  train.cl    <- as.factor(rest.cl[ which( index != iFold ) ])

  validation.data  <- rest.data[ which( index == iFold ) , ]
  validation.cl    <- rest.cl[ which( index == iFold )  ]

  for( i.c in 1:length(range.cost) )
  {
    i.cost <- range.cost[i.c]

    for( i.b in 1:length(range.bw))
    {
      i.bw <- range.bw[i.b]

      # fit SVM model
      fit.svm <- svm( x = train.data, y = train.cl, kernel = "radial",
                      gamma = i.bw, cost = i.cost )

      # predict in validation dataset
      class.pred <- predict( fit.svm, validation.data )

      # calculate error in validation data for the fit
      error[ iFold, i.c, i.b ] <- sum( class.pred != validation.cl )/length(class.pred)
    }
  }
}

# calculate average CV error
meanError <- matrix(0, length(range.cost), length(range.bw) )
for( iFold in 1:nFolds )
{
  meanError <- meanError + error[iFold,,]
}
meanError <- meanError/nFolds

rownames(meanError) <- paste(range.cost)
colnames(meanError) <- paste(range.bw)
```

Let's see the mean error for each combination of parameter:

```
meanError
```

```
##        5e-04   0.005    0.01   0.015    0.02   0.025    0.03
## 0.1  0.25000 0.10000 0.19375 0.35625 0.48125 0.49375 0.49375
## 0.25 0.03125 0.05625 0.10625 0.17500 0.30625 0.36875 0.46250
## 0.5  0.02500 0.02500 0.08125 0.11250 0.15000 0.24375 0.31250
## 0.75 0.02500 0.01250 0.05000 0.08750 0.11875 0.12500 0.20625
## 0.9  0.02500 0.01250 0.04375 0.07500 0.10625 0.11875 0.14375
## 1    0.02500 0.01250 0.04375 0.06875 0.10625 0.11875 0.11875
## 1.2  0.02500 0.01250 0.03750 0.05625 0.10000 0.11250 0.11250
## 1.4  0.02500 0.01250 0.03750 0.05625 0.10000 0.11250 0.11250
```

As we can see, the minimum error is obtained when the margin parameter $\gamma = 0.75$ and the bandwidth of the kernel is set to 0.005. Now that we have found the optimal parameters for γ and bandwidth, we can train our model using the whole data (train + validation) and estimate the prediction error on the test set.

```
# fit model
fit2.svm <- svm( x = rest.data, y = as.factor(rest.cl), kernel = "radial",
                      gamma = 0.005, cost = 0.75 )

# predict in test dataset
class.pred.2 <- predict( fit2.svm, test.data )

# calculate prediction error in test data
misc.rate.fit2 <- sum( class.pred.2 != test.cl )/length(class.pred.2)
```

And the misclassification rate for this model is given by:

```
misc.rate.fit2
```

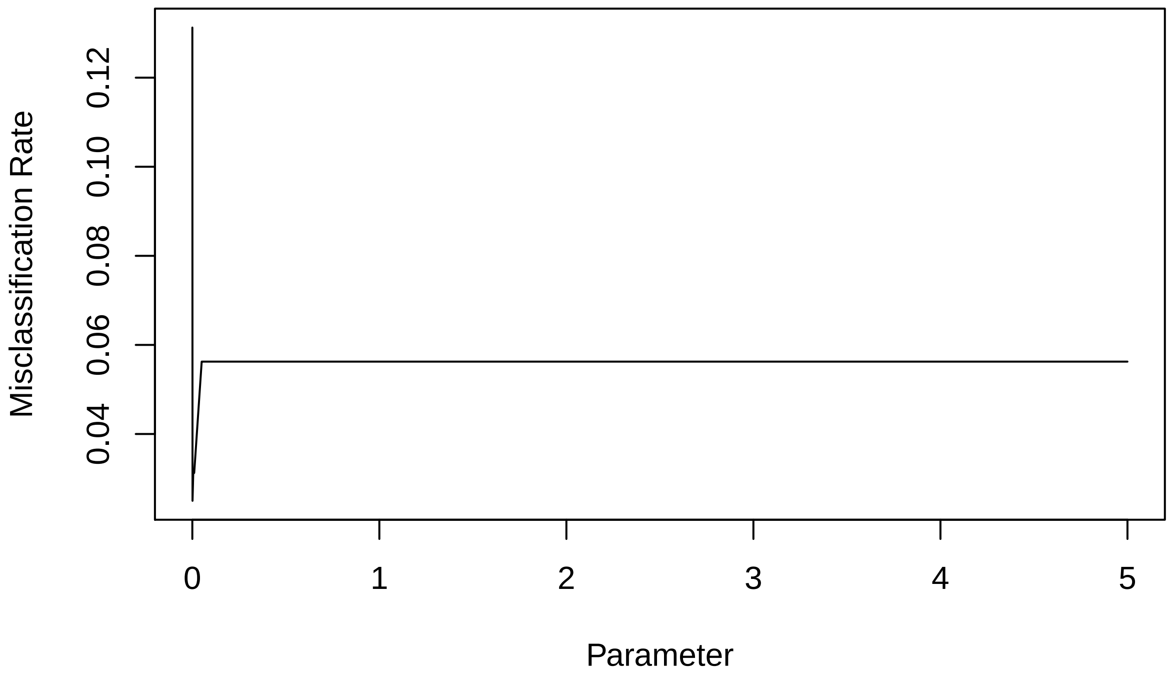
```
## [1] 0.025
```

or 2.5%.

Now, let's plot the cross-validation estimates of the misclassification rate as function of the different parameters:

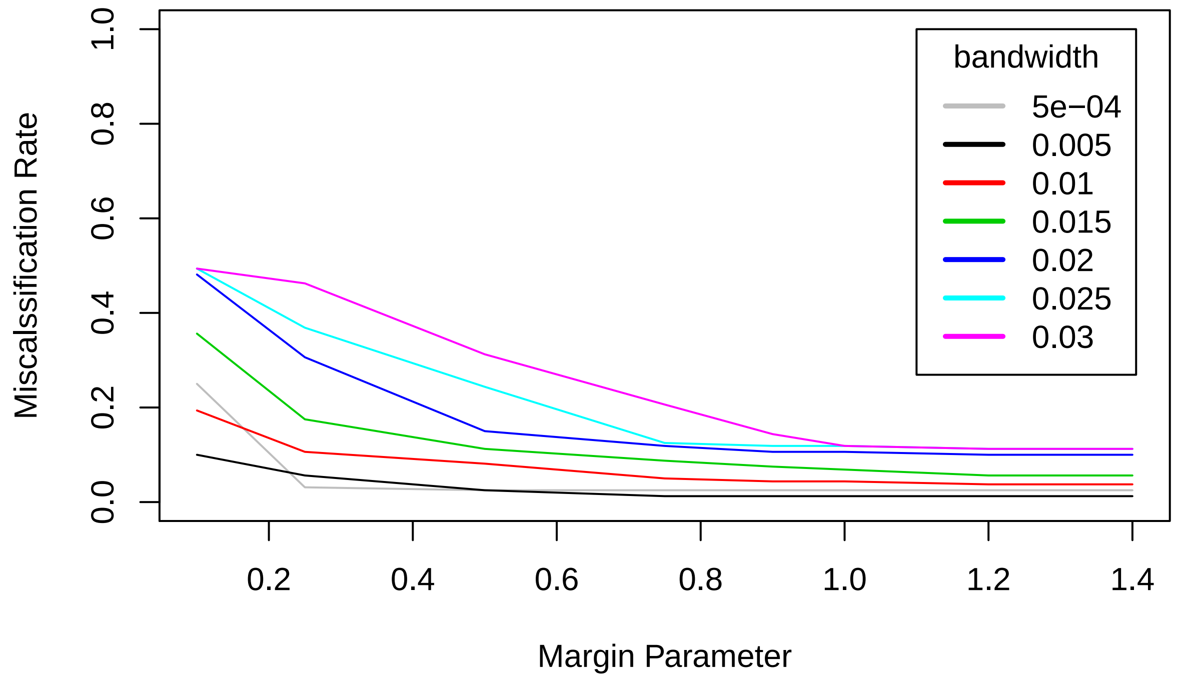
- margin parameter in the linear case.

```
linear.Error <- apply( matError, 2, mean)
linear.param <- as.numeric(colnames(matError))
plot( x = linear.param, y = linear.Error, type = "l",
      xlab = "Parameter", ylab = "Misclassification Rate")
```



- margin parameter and the kernel bandwidth in the non-linear case.

```
kern.cost <- as.numeric( rownames( meanError ) )
kern.bw   <- as.numeric( colnames( meanError ) )
plot( x = kern.cost, y = meanError[,1], col = 1+7, ylim = c(0,1), type = 'l',
      xlab = "Margin Parameter", ylab = "Miscalssification Rate")
for( icol in 2:ncol(meanError) )
{
  lines(  x = kern.cost, y = meanError[,icol], col = icol+7 )
}
legend( x = 1.1, y = 1, legend = colnames(meanError),
       lty = rep(1,ncol(meanError)), lwd = rep(2.5,ncol(meanError)),
       col = seq(8,ncol(meanError)+7,1 ), title = "bandwidth")
```



As we can note, both models attain the same misclassification rate on this data set. This way, if one had to choose between one of them, it would be better to use the linear SVM since this is a simpler model with fewer parameters.

adaBoost

First, let's load the functions that implement the adaBoost algorithm using decision stumps.

```
source("../lib/helpers.R")
```

Now, let's use the adaBoost model on the USPS data set.

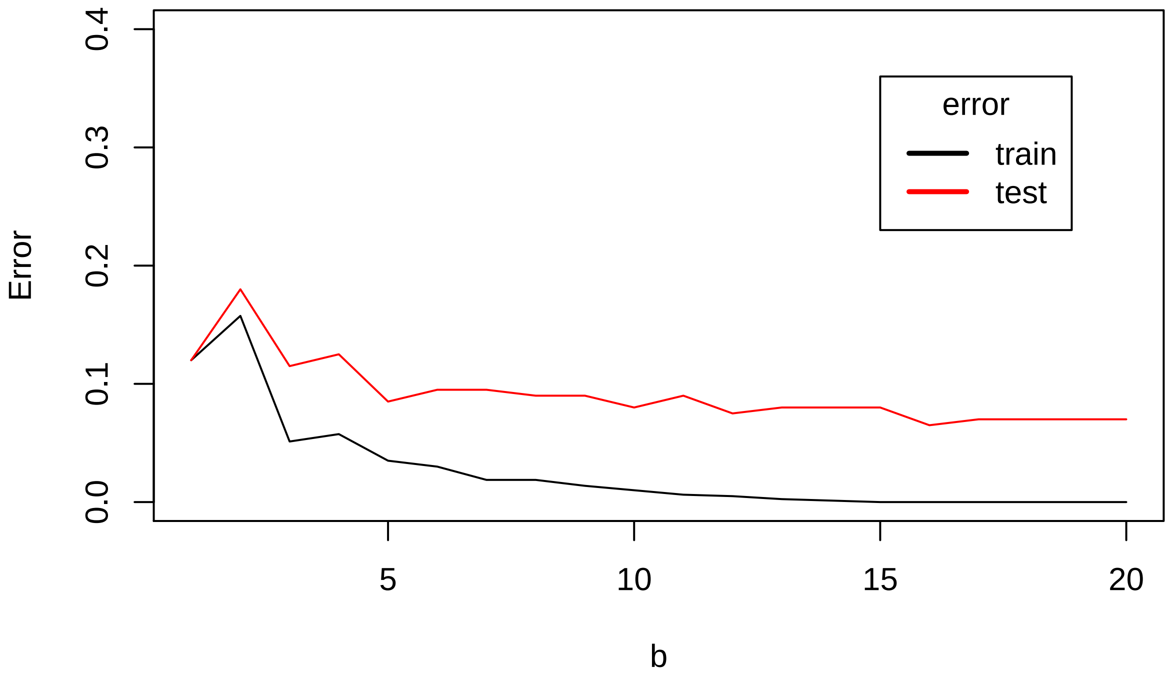
```
# set number of decision stumps
B <- 20

# call adaBoost function
boost.model <- adaBoost( X = uspsdata, y = uspscl[[1]],
                         B = B, w = rep(1/nrow(uspsdata), nrow(uspsdata)), nFolds = 5 )

# calculate average training error
avg.train.error <- apply( boost.model$error.train, 2, mean )

# calculate average prediction error
avg.test.error <- apply( boost.model$error.test, 2, mean )

plot( 1:B, avg.train.error, type = 'l', ylim = c(0,0.4), ylab = "Error", xlab = "b")
lines( 1:B, avg.test.error, type = 'l', col = 'red')
legend( x = B-5, y = 0.36, legend = c("train","test"),
       lty = c(1,1), lwd = c(2.5,2.5),
       col = c("black","red"), title = "error")
```


error
train
test
Error
0.0
0.1
0.2
0.3
0.4
5
10
15
20
b