Classifying Text articles into one of the 4 categories using Support Vector Machines

Suhail Shaikh

3/3/2020

loading the training and test data into data frames

```
require(Matrix)

## Loading required package: Matrix

# Change to the working directory where the data files are located.
# TODO: You should change the following ... to your working directory
setwd('E:\\Spring 2020\\IDS 575\\Assg2\\hw2')

# Read all individual lines in a text file.
# m = the number of training examples
dataFile <- file("articles.TRAIN", "r")
dataLines <- readLines(dataFile)

m <- length(dataLines)

close(dataFile)
#dataLines

# Split every string element by tokenizing space and colon.
dataTokens = strsplit(dataLines, "[: ]")

# Extract every first token from each line as a vector of numbers, which is
the class label.
Y = sapply(dataTokens, function(example) {as.numeric(x = example[1])})

# Extract the rest of tokens from each line as a list of matrices (one matrix
for each line)
# where each row consists of two columns: (feature number, its occurrences)
X_list = lapply(dataTokens, function(example) {n = length(example) - 1;
matrix(as.numeric(example[2:(n+1)]), ncol=2, byrow=T)})

# Add one column that indicates the example number at the left
X_list = mapply(cbind, x=1:length(X_list), y=X_list)

# Merge a list of different examples vertcially into a matrix
X_data = do.call('rbind', X_list)

library(Matrix)
```

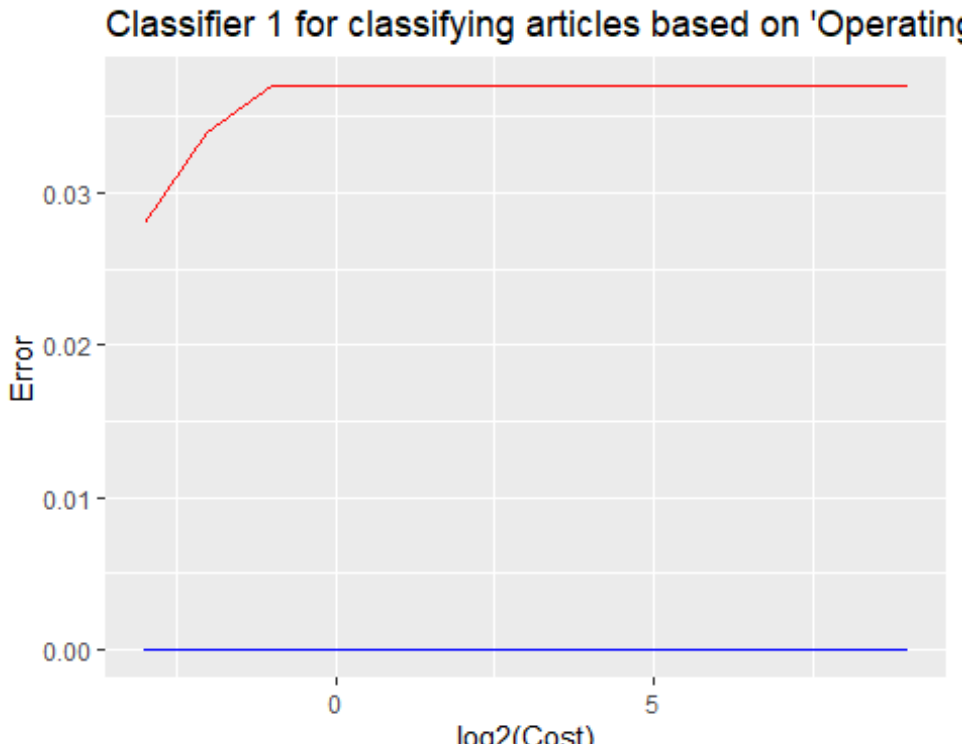
```
# Get a sparse data matrix X (rows: training exmaples, columns: # of
occurrences for each of features)
X = sparseMatrix(x=X_data[,3], i=X_data[,1], j=X_data[,2])
#Columns index represent word index from word.map. Row index represent
example number. Values represent frequency of word

#-----------------------------------------------------Test files
TestdataFile <- file("articles.TEST", "r")
dataLinesTest <- readLines(TestdataFile)

mtest <- length(dataLinesTest)

close(TestdataFile)
#dataLinesTest
# Split every string element by tokenizing space and colon.
dataTokensTest = strsplit(dataLinesTest, "[: ]")

# Extract every first token from each line as a vector of numbers, which is
the class label.
Y_Test = sapply(dataTokensTest, function(example) {as.numeric(x =
example[1])})

# Extract the rest of tokens from each line as a list of matrices (one matrix
for each line)
# where each row consists of two columns: (feature number, its occurrences)
X_list_Test = lapply(dataTokensTest, function(example) {n = length(example) -
1; matrix(as.numeric(example[2:(n+1)]), ncol=2, byrow=T)})

# Add one column that indicates the example number at the left
X_list_Test = mapply(cbind, x=1:length(X_list_Test), y=X_list_Test)

# Merge a list of different examples vertcially into a matrix
X_data_Test = do.call('rbind', X_list_Test)

# Get a sparse data matrix X (rows: training exmaples, columns: # of
occurrences for each of features)
X_Test = sparseMatrix(x=X_data_Test[,3], i=X_data_Test[,1],
j=X_data_Test[,2])
#Columns index represent word index from word.map. Row index represent
example number. Values represent frequency of word

Y1_Test= ifelse(Y_Test==1,1,-1)
Y2_Test= ifelse(Y_Test==2,1,-1)
Y3_Test= ifelse(Y_Test==3,1,-1)
Y4_Test= ifelse(Y_Test==4,1,-1)
#Importing test completed

#Selecting only words in Test on which model is trained
X_Test <-X_Test[,1:ncol(X)]
```

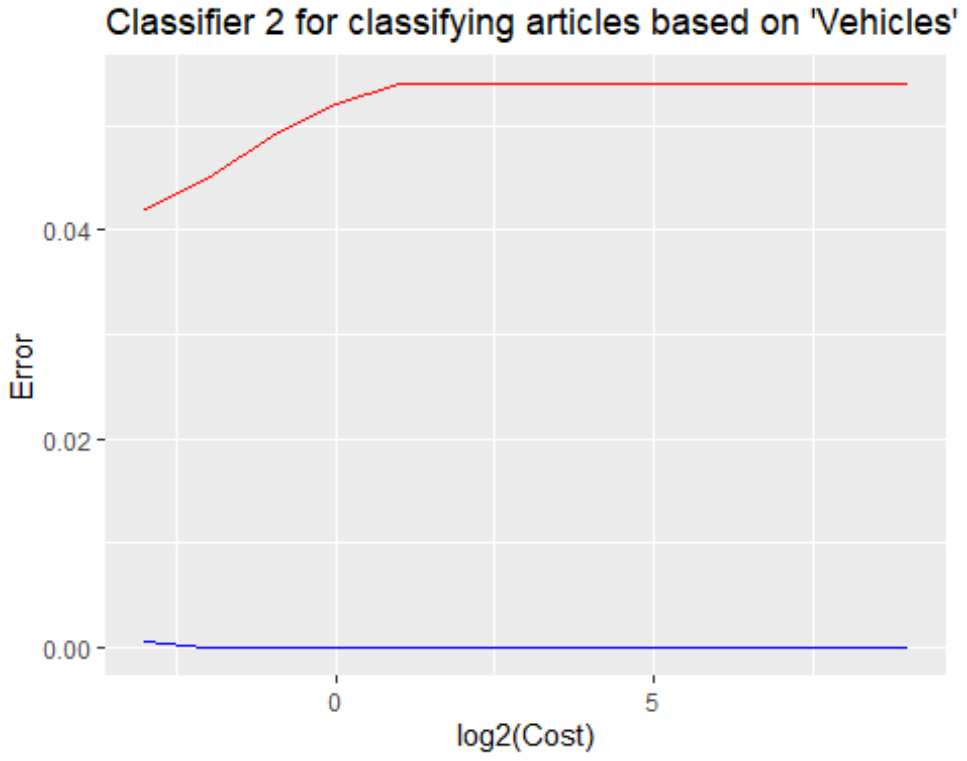
```
library(e1071)
Y1= ifelse(Y==1,1,-1)
Y2= ifelse(Y==2,1,-1)
Y3= ifelse(Y==3,1,-1)
Y4= ifelse(Y==4,1,-1)
```

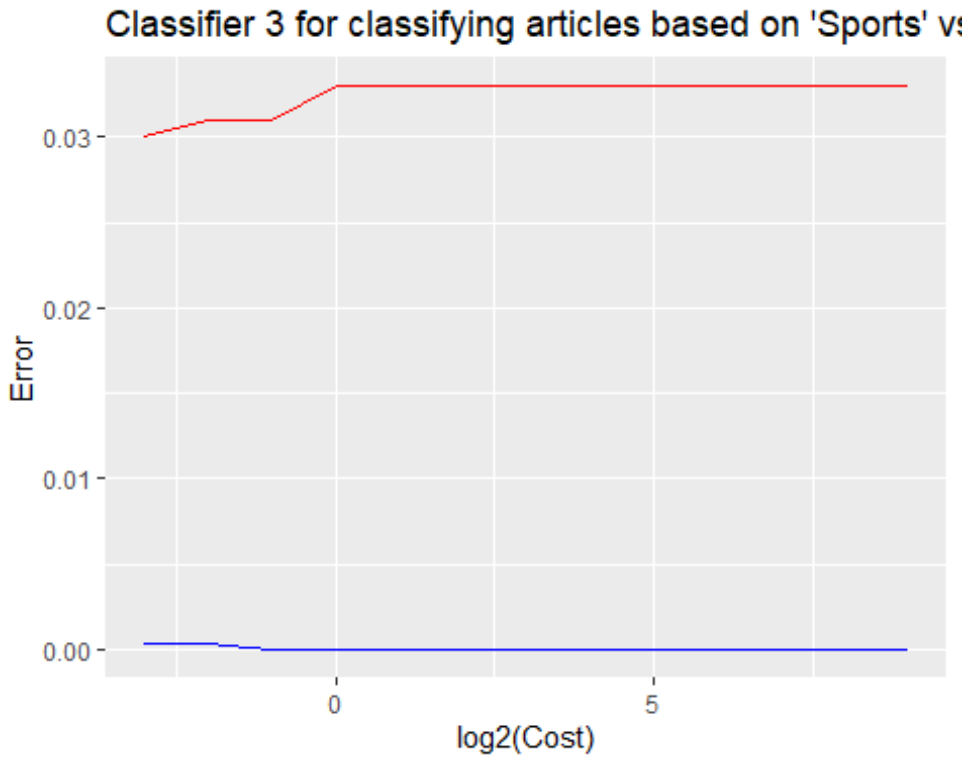
##Training four different (hard-margin) linear classifiers. As SVM classifies only binary labels, replaced the target class number to 1 and all others to -1 before calling the library function. For instance classify whether or not politics, using 1,000 articles about politics as positive samples and 3,000 others as negative samples for training. Once learned the four classifiers, the output label of each test example x is determined by the following formula: ##hw,b(x) = argmaxk∈{1,2,3,4}(w(k)Tx + b(k)) ##where w(k) is the learned weight vector, and b(k) is the biased term for the class k. ##If the prediction hw,b(x) is different from the ground-truth class, it yields an error. Reported the training and test errors of four classifiers, respectively.

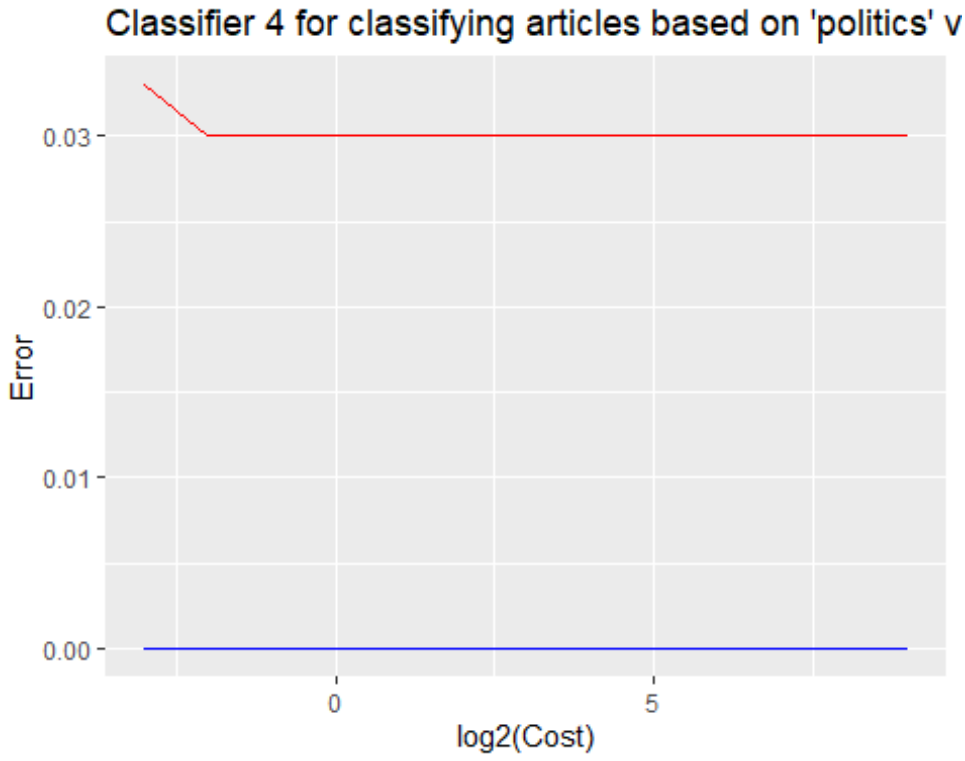
```
## Training_Error= 0
##  Testing_Error= 0.04166667

## Training_Error= 0
##  Testing_Error= 0.06666667

## Training_Error= 0
##  Testing_Error= 0.05291667

## Training_Error= 0
##  Testing_Error= 0.05916667

##       PredictingTestingArticle
## Y_Test   1   2   3   4
##      1 536  28  10  26
##      2  16 539  17  28
##      3   9  25 550  16
##      4   8  28  22 542

## Training_Error Combined Classifier= 0
##  Testing_Error Combined Classifier= 9.708333
```

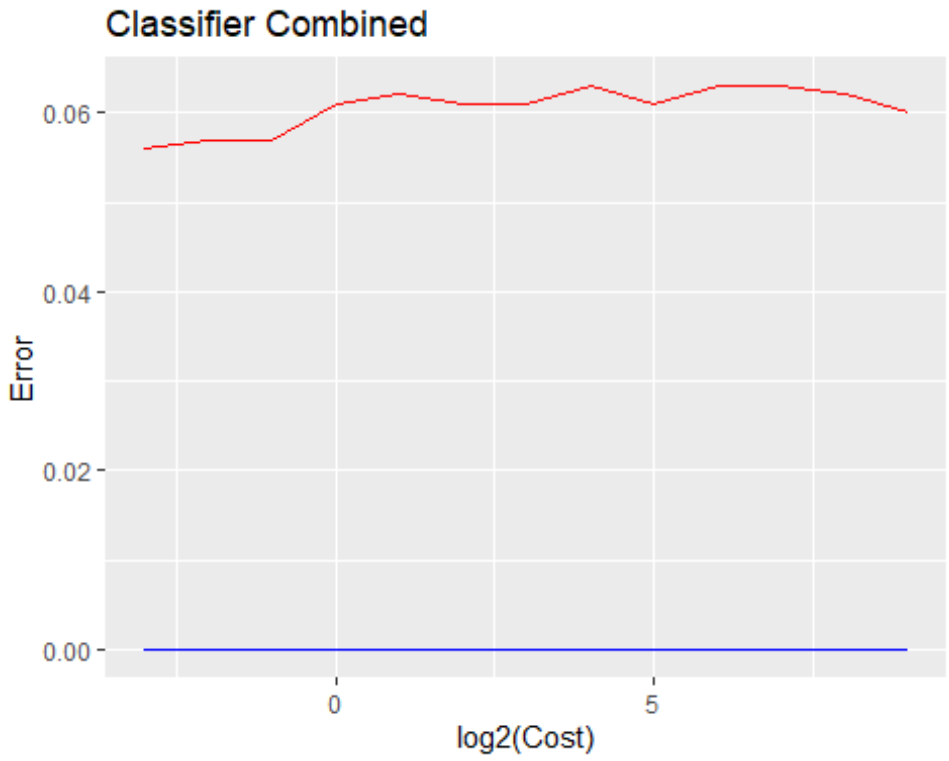
C) Training soft-margin linear classifiers with different Cost values from {0.125, 0.25, 0.5, 1, 2, 4, 8, … , 256, 512}. In order to pick the best Cost value, performed a hold-out validation:


Classifier 1 for classifying articles based on 'Operating
0.03
0.02
0.01
0.00
Error
0
5
log2(Cost)


Classifier 2 for classifying articles based on 'Vehicles'
0.04
0.02
0.00
Error
0
5
log2(Cost)


Classifier 3 for classifying articles based on 'Sports' vs
0.03
0.02
0.01
0.00
Error
0
5
log2(Cost)


Classifier 4 for classifying articles based on 'politics' v
0.03
0.02
0.01
0.00
Error
0
5
log2(Cost)



d)With the best Cost value chosen from part (b), learned four soft-margin classifiers again on the entire training set. Testing newly learned best classifiers on the test set where the output label is determined by the argmax class. Comparing the test error rates to hard-margin classifiers

```
## Testing_Error Classifier 1= 0.0375

## Testing_Error Classifier 2= 0.05291667

## Testing_Error Classifier 3= 0.03458333

## Testing_Error Classifier 4= 0.05875

##       PredictArticle_Best_C
## Y_Test   1   2   3   4
##      1 540  18  14  28
##      2  12 556  11  21
##      3   7  22 557  14
##      4  11  24  12 553

## Testing_Error Classifier 4= 8.083333
```

##e) Normalizing feature vectors so that the feature vectors of each example have unit length. For each example x = (x1,x2,…,xn), dividing every component into kxk2 so that kxk2 = 1/ Repeating the part (b) with normalized features and measuring the test error rates

again with newly picked C value. Comparing the new test error to previous test error from soft-margin classifier without normalization.

```
## Warning: package 'wordspace' was built under R version 3.6.2

##       predictingTestingArticle_Norm
## Y_Test   1   2   3   4
##      1 553  17  11  19
##      2  10 566   5  19
##      3   5   8 575  12
##      4   4  15   7 574

## Testing_Error Cpmbined Classifier = 5.5
```

f)What I have done so far is one way to extend binary SVM to a multiclass classifier, which is called 1-vs-all. There is another way of called 1-vs-1, where you are supposed to train all [4|2] binary classifiers distinguishing between every pair of classes. Then for each test example, picking the (not necessarily unique) class that achieves the highest votes.

```
## Warning: package 'modeest' was built under R version 3.6.2

##
## Attaching package: 'modeest'

## The following object is masked from 'package:e1071':
##
##     skewness

##       predictingTestingArticle_Norm_1vs1
## Y_Test   1   2   3
##      1 247 314  39
##      2   0 589  11
##      3   1 118 481
##      4   0 552  48

## Testing_Error of Combined Classifier of all 1vs1 classifiers and Hard
margin
##  45.125

##       predictingTestingArticle_Norm_1vs1
## Y_Test   1   2   3   4
##      1 530  23  10  37
##      2  32 476  27  65
##      3  28  26 515  31
##      4  16  42  32 510
```

```
## Testing_Error of Combined Classifier of all 1vs1 classifiers and  soft
margin on normalinezed data
##  15.375
```