R for Support Vector Machines
Dolores Romero Morales

The aim of this workshop is to perform Classification with R using a brief guide to R for Support Vector Machines and one dataset (spam.txt). All files can be found on the course directory. More information on the spam dataset can be found in https://archive.ics.uci.edu/ml/datasets/Spambase.

Task 0
Install and upload the corresponding R package

Note: To build Support Vector Machines we will use the R package 'e1071'. Follow the two steps below to complete the installation and uploading of the package:

```
install.packages('e1071',dependencies=TRUE)
library('e1071')
```

Once you are ready with the guide, use the dataset to perform the following steps using R:

Task 1
Get familiar with the spam dataset

Step 1. Read about the spam dataset in https://archive.ics.uci.edu/ml/datasets/Spambase.

Note: It is important to get familiar with the data you will be working with. It is important to know which are the explanatory variables (columns 1 to 57), and the class membership (column 58).

Step 2. Read the spam.txt file into a data frame and get the dimension of the dataset.
Answer:

```
myspam <- read.table(file.choose(),header=TRUE,stringsAsFactors=TRUE)
dim(myspam)
```

Note:
With file.choose() a new window will pop up and you will need to find the file.

Step 3. Get a summary report for all the explanatory variables and the class split, i.e., the number of observations in each class. Recall that the class membership is given by the last column.
Answer:

```
summary(myspam)
typeemailtable <- table(myspam$type)
typenonspamemail <- typeemailtable[1]/nrow(myspam)
typespamemail <- typeemailtable[2]/nrow(myspam)
```

Note1:
If you have a look at the summary given for columns 1 through 57, we can see the min and the max, the quartiles and the mean.
Note2:
If you have a look at the summary for column 58, we can see the number of spam, and the number of nonspam. With this you can calculate the class split, by calculating the percentage of spam versus nonspam. You can see that the class split is roughly speaking 60% of nonspam

and 40% of spam emails.

Step 4. Reshuffle the dataset and take a subsample of 500 observations. Call this the *minispam*. Call the remaining observations *testspam*.

Answer:

```
set.seed(1)
mymyspam <- myspam[sample(nrow(myspam)),]
minispam <- mymyspam[1:500,]
testspam <- mymyspam[501: nrow(mymyspam),]
```

Note1:
It is usual to take a smaller sample to tune the parameters of SVM. Note that the first step reshuffles the rows of the dataset, and the second step takes just 500 of the observations.
Note2:
Prior to the function that reshuffles the dataset, you may want to set the seed for the random generator using the function set.seed(). If you do so, then you are ensuring that you can reproduce in the future the same reshuffling.

Step 5. Get a summary report for all the explanatory variables and the class split, i.e., the number of observations in each class for both *minispam* and *testspam*.

Answer: As in Step 3.

Task 2
Classification with linear Support Vector Machine

Step 6. Tune the Support Vector Machine for values of the tradeoff parameter equal to 0.001, 0.01, 0.1, 1, 10, 100, 1000 using *minispam*. Report the best value of the parameter found.

Answer:

```
set.seed(1000)
tunedmodellinear <- tune.svm(type~.,data = minispam, cost=10^(-3:3), kernel= "linear")
bestcost <- tunedmodellinear$best.parameters[[1]]
```

Note1:
The class labels are ‘spam/nonspam’. Please note that the package e1071 gets ‘confused’ if the label given by a numeric column. In that case it will return an SVM-regression as opposed to an SVM-classification model. To avoid any confusion, make sure that your class labels are not numeric.
Note2:
Recall that the e1071 package calls ‘cost’ to the tradeoff parameter C.
Note3:
This function reshuffles the dataset, and you may want to set the seed for the random generator using the function set.seed(). As I mention above, if you do so, then you are ensuring that you can reproduce results again in the future.
Note4:
It is not been asked in the question, but it would be good if you got a summary of the tuning process that you have just performed. For that, you can call, as usual, the summary function. Please see the output of the summary function below. In particular, it reports

- the best value of cost found in the chosen grid: 0.1
- the corresponding best 10-fold cross validation error: 0.08

```
summary(tunedmodellinear)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.1

- best performance: 0.08

- Detailed performance results:
   cost error dispersion
1 1e-03 0.206 0.07244922
2 1e-02 0.094 0.06931410
3 1e-01 0.080 0.05249339
4 1e+00 0.088 0.05181162
5 1e+01 0.092 0.04131182
6 1e+02 0.092 0.03675746
7 1e+03 0.094 0.04221637
```

Step 7. Use the best value of the parameter found in the previous step to build a model in *minispam*. Test the model in *testspam* and report the classification accuracy.

Answer:

```
finalmodellinear <- svm(type~.,data=minispam,cost= bestcost, kernel="linear")
predictionlinear <- predict(finalmodellinear,testspam[,-58])
classificationtable <- table(pred= predictionlinear, testspam[,58])
acctestfinalmodellinear <- sum(diag(classificationtable))/sum(classificationtable)
```

Note1:
acctestfinalmodellinear is roughly 90%.

Note2:
When you use the linear kernel you can retrieve the coefficient of the explanatory variables, as well as the independent term b.

```
myomega <- t(finalmodellinear$coefs) %*% finalmodellinear$SV
myb <- -finalmodellinear$rho
```

The hyperplane is written as myomega[1,1]*x1 + myomega[1,2]*x2 … + myb = 0

It is important you realize this cannot be done for the RBF kernel, but only for the linear kernel you can.

Task 3
Classification with RBF Support Vector Machine

Step 8. Tune the Support Vector Machine with the RBF kernel, for values of the tradeoff parameter equal to 1,10,100 and values of the RBF parameter equal to 0.25,0.5,1,2,4 using *minispam*. Report the best value found for the parameters.

Answer:

```
set.seed(9999)
tunedmodelRBF <- tune.svm(type~.,data = minispam,gamma=2^(-2:2),cost=10^(0:2))
bestgamma <- tunedmodelRBF$best.parameters[[1]]
bestcost <- tunedmodelRBF$best.parameters[[2]]
```

Note1:
For the purpose of getting quick results in class, we have tried out only a few values of the parameters. It is important that you understand that we SHOULD have taken a wider range of values of the parameters (cost and gamma) in the model. By taking a wider range, you are doing a more thorough parameter tuning, and therefore, a better training of your final SVM model with the RBF kernel.

Note2:
It is not been asked in the question, but it would be good if you got a summary of the tuning process that you have just performed. For that, you can call, as usual, the summary function. Please see the output of the summary function below. In particular, it reports

- the best value of cost found in the chosen grid: 10
- the best value of gamma found in the chosen grid: 0.25
- the corresponding best 10-fold cross validation error: 0.252

```
summary(tunedmodelRBF)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 gamma cost
  0.25   10

- best performance: 0.252

- Detailed performance results:
   gamma cost error dispersion
1   0.25    1 0.268 0.07130529
2   0.50    1 0.310 0.06749486
3   1.00    1 0.330 0.06683313
4   2.00    1 0.336 0.06380526
5   4.00    1 0.334 0.06669999
6   0.25   10 0.252 0.07130529
7   0.50   10 0.302 0.06356099
8   1.00   10 0.324 0.07042727
9   2.00   10 0.336 0.06380526
10  4.00   10 0.336 0.06785606
11  0.25  100 0.258 0.06957011
12  0.50  100 0.298 0.05921711
13  1.00  100 0.326 0.06801961
14  2.00  100 0.338 0.06142746
15  4.00  100 0.336 0.06785606
```

Step 9. Use the best value of the parameters found in the previous step to build a model in *minispam*. Test the model in *testspam* and report the classification accuracy.

Answer:

```
finalmodelRBF <- svm(type~.,data=minispam,gamma=bestgamma,cost=bestcost)
predictionRBF <- predict(finalmodelRBF,testspam[,-58])
classificationtable<-table(pred= predictionRBF, testspam[,58])
acctestfinalmodelRBF <- sum(diag(classificationtable))/sum(classificationtable)
```

Note1:
acctestfinalmodelRBF is roughly 74.5%.

Note2:
This accuracy is quite disappointing, and much lower than then one we obtained in Task 2 for the Linear SVM. I would recommend you enlarge the set of value you try out for the parameters. See below

```
set.seed(9999)
tunedmodelRBF <- tune.svm(type~.,data = minispam,gamma=2^(-6:6),cost=2^(-6:6))
bestgamma <- tunedmodelRBF$best.parameters[[1]]
bestcost <- tunedmodelRBF$best.parameters[[2]]
finalmodelRBF <- svm(type~.,data=minispam,gamma=bestgamma,cost=bestcost)
predictionRBF <- predict(finalmodelRBF,testspam[,-58])
classificationtable<-table(pred= predictionRBF, testspam[,58])
acctestfinalmodelRBF <- sum(diag(classificationtable))/sum(classificationtable)
```

For this
acctestfinalmodelRBF is roughly 89.5%.