Intro to Data Science - Lab 9

Week 9 – Supervised Data Mining

```
# Enter your name here: Rebecca Candee
```

Please include nice comments.

Instructions:

Run the necessary code on your own instance of R-Studio.

Attribution statement: (choose only one and delete the rest)

```
# 1. I did this lab assignment by myself, with help from the book and the professor.
```

Supervised data mining/machine learning is the most prevalent form of data mining as it allows for the prediction of new cases in the future. For example, when credit card companies are trying to detect fraud, they will create a supervised model by training it on fraud data that they already have. Then they will deploy the
model into the field: As new input data arrives the model predicts whether it seems fraudulent and flags those transactions where that probability is high.

In these exercises we will work with a built-in data set called **GermanCredit**. This data set is in the **“caret”** package so we will need that and the **“kernlab”** package to be installed and “libraried” before running the following:

```
data("GermanCredit")
subCredit <- GermanCredit[,1:10]
str(subCredit)
```

```
install.packages("caret")
install.packages("kernlab")

     Installing package into ‘/usr/local/lib/R/site-library’
     (as ‘lib’ is unspecified)

     also installing the dependencies ‘listenv’, ‘parallelly’, ‘future’, ‘globals’, ‘


     Installing package into ‘/usr/local/lib/R/site-library’
     (as ‘lib’ is unspecified)
```

```
library(caret)
library(kernlab)

     Loading required package: ggplot2

     Loading required package: lattice

     Warning message in system("timedatectl", intern = TRUE):
     “running command 'timedatectl' had status 1”

     Attaching package: ‘kernlab’


     The following object is masked from ‘package:ggplot2’:

         alpha
```

```
data("GermanCredit")
subCredit <- GermanCredit[,1:10]
str(subCredit)

     'data.frame':   1000 obs. of  10 variables:
      $ Duration                 : int  6 48 12 42 24 36 24 36 12 30 ...
      $ Amount                   : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 
      $ InstallmentRatePercentage: int  4 2 2 2 3 2 3 2 2 4 ...
      $ ResidenceDuration        : int  4 2 3 4 4 4 4 2 4 2 ...
      $ Age                      : int  67 22 49 45 53 35 53 35 61 28 ...
      $ NumberExistingCredits    : int  2 1 1 1 2 1 1 1 1 2 ...
      $ NumberPeopleMaintenance  : int  1 1 2 2 2 2 1 1 1 1 ...
      $ Telephone                : num  0 1 1 1 1 0 1 0 1 1 ...
      $ ForeignWorker            : num  1 1 1 1 1 1 1 1 1 1 ...
      $ Class                    : Factor w/ 2 levels "Bad","Good": 2 1 2 2 1 2 2 2 2
```

1. Examine the data structure that **str()** reveals. Also use the **help()** command to learn more about the **GermanCredit** data set. Summarize what you see in a comment.

```
str(subCredit)
help("GermanCredit")
#The data has two different classes for the credit- good and bad.
```

```
'data.frame':	1000 obs. of  10 variables:
 $ Duration                 : int  6 48 12 42 24 36 24 36 12 30 ...
 $ Amount                   : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 
 $ InstallmentRatePercentage: int  4 2 2 2 3 2 3 2 2 4 ...
 $ ResidenceDuration        : int  4 2 3 4 4 4 4 2 4 2 ...
 $ Age                      : int  67 22 49 45 53 35 53 35 61 28 ...
 $ NumberExistingCredits    : int  2 1 1 1 2 1 1 1 1 2 ...
 $ NumberPeopleMaintenance  : int  1 1 2 2 2 2 1 1 1 1 ...
 $ Telephone                : num  0 1 1 1 1 0 1 0 1 1 ...
 $ ForeignWorker            : num  1 1 1 1 1 1 1 1 1 1 ...
 $ Class                    : Factor w/ 2 levels "Bad","Good": 2 1 2 2 1 2 2 2 2
```

2. Use the **createDataPartition()** function to generate a list of cases to include in the training data. This function is conveniently provided by caret and allows one to directly control the number of training cases. It also ensures that the training cases are balanced with respect to the outcome variable. Try this:

```
trainList <- createDataPartition(y=subCredit$Class,p=.40,list=FALSE)
```

```
trainList <- createDataPartition(y=subCredit$Class,p=.40,list=FALSE)
```

3. Examine the contents of **trainList** to make sure that it is a list of case numbers. With **p=0.40**, it should have 400 case numbers in it.

```
summary(trainList)
#It has a list of 400 case numbers
```

4. What is **trainList**? What do the elements in **trainList** represent? Which attribute is balanced in the **trainList** dataset?

```
#the elements in trainList represent 400 random rows of good and bad credit from the s
```

5. Use **trainList** and the square brackets notation to create a training data set called **“trainSet”** from the **subCredit** data frame. Look at the structure of trainSet to make sure it has all of the same variables as **subCredit**. The **trainSet** structure should be a data frame with **400 rows and 10 columns**.

```
trainSet <- subCredit[trainList, ]
```

6. Use **trainList** and the square brackets notation to create a testing data set called **“testSet”** from the subCredit data frame. The **testSet** structure should be a data frame with **600 rows and 10 columns** and should be a completely different set of cases than **trainSet**.

```
testSet <- subCredit[-trainList, ]
```

7. Create and interpret boxplots of all the predictor variables in relation to the outcome variable (**Class**).

```
boxplot(Age ~ Class, data = subCredit)
boxplot(Duration ~ Class, data = subCredit)
```

8. Train a support vector machine with the **ksvm()** function from the **kernlab** package. Make sure that you have installed and libraried the **kernlab** package. Have the **cost** be 5, and have **ksvm** do 3 **cross validations** (Hint: `try prob.model = TRUE`)

```
svmOutput<-ksvm(Class ~ ., data=trainSet, kernel="rbfdot", kpar="automatic", cost=5, c
svmOutput
```

9. Examine the ksvm output object. In particular, look at the **cross-validation error** for an initial indication of model quality. Add a comment that gives your opinion on whether this is a good model.

```
#This is a good model
```

10. Predict the training cases using the **predict()** command

```
svmPredict<-predict(svmOutput, testSet,type="response")
svmPredict
```

11. Examine the predicted out object with **str( )**. Then, calculate a **confusion matrix** using the **table()** function.

```
str(svmPredict)
```

```
 Factor w/ 2 levels "Bad","Good": 2 2 2 2 2 2 2 2 2 2 ...
```

```
table(testSet$Class,svmPredict)
```

12. Interpret the confusion matrix and in particular calculate the overall **accuracy** of the model. The **diag( )** command can be applied to the results of the table command you ran in the previous step. You can also use **sum( )** to get the total of all four cells.

```
sum(diag(table(svmPredict,testSet$Class)))/sum(table(svmPredict,testSet$Class))
```

13. Check you calculation with the **confusionMatrix()** function in the **caret** package.

```
confusionMatrix(svmPredict,testSet$Class)
```

✓ 0s completed at 3:14 PM