Intro to Data Science - Lab 9

Week 9 - Supervised Data Mining

```
# Enter your name here: Tao Pang
```

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.
#remove.packages('ggplot2')
#install.packages('caret')
#install.packages('kernlab')
library(caret)
```

```
## 载入需要的程辑包：ggplot2
```

```
## 载入需要的程辑包：lattice
```

```
library(kernlab)
```

```
##
## 载入程辑包：'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

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)
```


```
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 52
34 ...
##  $ 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
...
```

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(GermanCredit)
```

```
## 'data.frame':    1000 obs. of  62 variables:
##  $ Duration                                : int  6 48 12 42 24 36 24 36 12 30 ...
##  $ Amount                                  : int  1169 5951 2096 7882 4870 9055 2835
6948 3059 5234 ...
##  $ 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 ...
##  $ CheckingAccountStatus.lt.0              : num  1 0 0 1 1 0 0 0 0 0 ...
##  $ CheckingAccountStatus.0.to.200          : num  0 1 0 0 0 0 0 1 0 1 ...
##  $ CheckingAccountStatus.gt.200            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ CheckingAccountStatus.none              : num  0 0 1 0 0 1 1 0 1 0 ...
##  $ CreditHistory.NoCredit.AllPaid          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ CreditHistory.ThisBank.AllPaid          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ CreditHistory.PaidDuly                  : num  0 1 0 1 0 1 1 1 1 0 ...
##  $ CreditHistory.Delay                     : num  0 0 0 0 1 0 0 0 0 0 ...
##  $ CreditHistory.Critical                  : num  1 0 1 0 0 0 0 0 0 1 ...
##  $ Purpose.NewCar                          : num  0 0 0 0 1 0 0 0 0 1 ...
##  $ Purpose.UsedCar                         : num  0 0 0 0 0 0 0 1 0 0 ...
##  $ Purpose.Furniture.Equipment             : num  0 0 0 1 0 0 1 0 0 0 ...
##  $ Purpose.Radio.Television                : num  1 1 0 0 0 0 0 0 1 0 ...
##  $ Purpose.DomesticAppliance               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Purpose.Repairs                         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Purpose.Education                       : num  0 0 1 0 0 1 0 0 0 0 ...
##  $ Purpose.Vacation                        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Purpose.Retraining                      : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Purpose.Business                        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Purpose.Other                           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ SavingsAccountBonds.lt.100              : num  0 1 1 1 1 0 0 1 0 1 ...
##  $ SavingsAccountBonds.100.to.500          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ SavingsAccountBonds.500.to.1000         : num  0 0 0 0 0 0 1 0 0 0 ...
##  $ SavingsAccountBonds.gt.1000             : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ SavingsAccountBonds.Unknown             : num  1 0 0 0 0 1 0 0 0 0 ...
##  $ EmploymentDuration.lt.1                 : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ EmploymentDuration.1.to.4               : num  0 1 0 0 1 1 0 1 0 0 ...
##  $ EmploymentDuration.4.to.7               : num  0 0 1 1 0 0 0 0 1 0 ...
##  $ EmploymentDuration.gt.7                 : num  1 0 0 0 0 0 1 0 0 0 ...
##  $ EmploymentDuration.Unemployed           : num  0 0 0 0 0 0 0 0 0 1 ...
##  $ Personal.Male.Divorced.Seperated        : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ Personal.Female.NotSingle               : num  0 1 0 0 0 0 0 0 0 0 ...
##  $ Personal.Male.Single                    : num  1 0 1 1 1 1 1 1 0 0 ...
##  $ Personal.Male.Married.Widowed           : num  0 0 0 0 0 0 0 0 0 1 ...
##  $ Personal.Female.Single                  : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ OtherDebtorsGuarantors.None             : num  1 1 1 0 1 1 1 1 1 1 ...
##  $ OtherDebtorsGuarantors.CoApplicant      : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ OtherDebtorsGuarantors.Guarantor        : num  0 0 0 1 0 0 0 0 0 0 ...
##  $ Property.RealEstate                     : num  1 1 1 0 0 0 0 0 1 0 ...
##  $ Property.Insurance                      : num  0 0 0 1 0 0 1 0 0 0 ...
##  $ Property.CarOther                       : num  0 0 0 0 0 0 0 1 0 1 ...
##  $ Property.Unknown                        : num  0 0 0 0 1 1 0 0 0 0 ...
```

```
##  $ OtherInstallmentPlans.Bank            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ OtherInstallmentPlans.Stores          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ OtherInstallmentPlans.None            : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Housing.Rent                          : num  0 0 0 0 0 0 0 1 0 0 ...
##  $ Housing.Own                           : num  1 1 1 0 0 0 1 0 1 1 ...
##  $ Housing.ForFree                       : num  0 0 0 1 1 1 0 0 0 0 ...
##  $ Job.UnemployedUnskilled               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Job.UnskilledResident                 : num  0 0 1 0 0 1 0 0 1 0 ...
##  $ Job.SkilledEmployee                   : num  1 1 0 1 1 0 1 0 0 0 ...
##  $ Job.Management.SelfEmp.HighlyQualified: num  0 0 0 0 0 0 0 1 0 1 ...
```

```
help(GermanCredit)
```

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.

```
str(trainList)
```

```
##  int [1:400, 1] 5 8 10 14 19 23 25 27 29 30 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr "Resample1"
```

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

```
#trainlist is a dataset
#elements in trainlist represent the good or bad class
#the class attribute is balanced in the trainlist dataset
```

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**).

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`)

```
ksvm_model <- ksvm(data = trainSet, Class ~ ., C = 5, cross = 3, prob.model = TRUE)
```

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.

```
ksvm_model
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 5
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.0973556772475811
##
## Number of Support Vectors : 271
##
## Objective Function Value : -904.0987
## Training error : 0.19
## Cross validation error : 0.299966
## Probability model included.
```

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

```
pridiction <-predict(ksvm_model,testSet)
```

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

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

```
##
## pridiction Bad Good
##       Bad   44   38
##       Good 136  382
```

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

```
## [1] 600
```

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