Computer lab 7 in MMS075, March 5, 2020

1. This exercise shows how to compute training MSE, how to estimate test MSE by cross-validation and how test error can be used for variable selection. We will work with the Advertising dataset which can be downloaded from http://faculty.marshall.usc.edu/gareth-james/ISL/data.html and imported in RStudio as usual. After downloading it, we remove the index variable X in the first column so that it won't be distracting:
   Advertising=Advertising[,-1]

   We attach the dataset so that we don't need to write "**Advertising$**" when referring to its variables:
   attach(Advertising)

   Let us first consider a simple linear regression model, with TV as the only predictor. We would usually define this model using the **lm** function. However, it will be important later that we define the model using the **glm** function this time. We will check that **glm** gives the same results as **lm** for linear regression models, by looking at the model summaries:
   AdModel1lm=lm(sales~TV)
   AdModel1=glm(sales~TV)
   summary(AdModel1lm)
   summary(AdModel1)

   We now compute the training mean squared error for this model, which is the mean of the squared differences between the predictions and the observations for the training set:
   Predictions1=predict(AdModel1,data=Advertising)
   PredictionErrors1=sales-Predictions1
   SquaredErrors1=PredictionErrors1^2
   mean(SquaredErrors1)

   Alternatively, one can compute the training MSE in one line:
   mean((sales-predict(AdModel1,data=Advertising))^2)

   We have now computed training MSE for one specific model. However, we may want to consider other models as well, with one or more predictors. Furthermore, we have seen before that interaction terms between **TV** and **radio** may also be of interest. We now look at all possible models predicting sales, including those with an interaction term between **TV** and **radio**, and compute training MSE for each one of them. We first define the models:
   AdModel1=glm(sales~TV)
   AdModel2=glm(sales~radio)
   AdModel3=glm(sales~newspaper)
   AdModel4=glm(sales~TV+radio)
   AdModel5=glm(sales~TV+newspaper)
   AdModel6=glm(sales~radio+newspaper)
   AdModel7=glm(sales~TV+radio+newspaper)
   AdModel8=glm(sales~TV*radio)
   AdModel9=glm(sales~TV*radio+newspaper)

It will be convenient to store the results in a variable called **TrainingMSE**. We create first a variable with 0 values and then fill it with meaningful values:
TrainingMSE=rep(0,9)
TrainingMSE[1]=mean((sales-predict(AdModel1,data=Advertising))^2)
TrainingMSE[2]=mean((sales-predict(AdModel2,data=Advertising))^2)
TrainingMSE[3]=mean((sales-predict(AdModel3,data=Advertising))^2)
TrainingMSE[4]=mean((sales-predict(AdModel4,data=Advertising))^2)
TrainingMSE[5]=mean((sales-predict(AdModel5,data=Advertising))^2)
TrainingMSE[6]=mean((sales-predict(AdModel6,data=Advertising))^2)
TrainingMSE[7]=mean((sales-predict(AdModel7,data=Advertising))^2)
TrainingMSE[8]=mean((sales-predict(AdModel8,data=Advertising))^2)
TrainingMSE[9]=mean((sales-predict(AdModel9,data=Advertising))^2)

We can now check the results and check which model gives the lowest training MSE value:
TrainingMSE
which.min(TrainingMSE)

However, instead of a model with low training MSE, we rather want a model with low test MSE! We do not have new data points to test our models with, hence we can only estimate test MSE. Below we compute the estimates for test MSE using Leave-One-Out Cross-Validation (LOOCV) and 5-fold cross-validation.

Doing LOOCV is easy in R, using the **cv.glm** function in the **boot** library. However, this function can only take models created with **glm** as an argument; that is why it was important to define the models using **glm** instead of **lm** at the beginning.
library(boot)
LOOCVAdModel1=cv.glm(AdModel1,data=Advertising)
LOOCVTestMSE1=LOOCVAdModel1$delta[1]

You might want to understand why the last command line was necessary. The **cv.glm** function will return a list, which has an argument called **delta**, which contains two numbers, and the first such number is the test error estimate discussed in the lecture. The other arguments are only needed for those who want to have a very detailed understanding of the cross-validation process and results – for our purposes, checking only the first number in delta is perfectly sufficient.

Doing 5-fold cross-validation is equally simple. Remember that in this case, the results depend somewhat on how the 5 folds are defined; therefore, we do it twice to see the difference. The selection of folds is a random process and we set seed for the random number generator to ensure that our results will be reproducible:
set.seed(1)
CV5AdModel1Seed1=cv.glm(AdModel1,data=Advertising,K=5)
CV5TestMSE1Seed1=CV5AdModel1Seed1$delta[1]

set.seed(2)
CV5AdModel1Seed2=cv.glm(AdModel1,data=Advertising,K=5)
CV5TestMSE1Seed2=CV5AdModel1Seed2$delta[1]

Compare the three estimated test errors for **AdModel1** with each other (i.e. the estimate from LOOCV and the two estimates from 10-fold CV using different seeds) and also with the corresponding training error!

Set the seed of the random number generator to 1 for reproducibility, compute test error estimates for each of the 9 models considered using 10-fold cross-validation and store the results in a variable called TestMSE. Which model gives the lowest estimated test MSE? Is this in line with our earlier conclusions during the course that **newspaper** is not a significant predictor, but it is important to include the interaction term between **TV** and **radio**?

If you believe that it would go fast, repeat this procedure after setting a different seed, e.g. **set.seed(2)**, and compute also the LOOCV-based test error estimates for all 9 models, and check which model is best according to the different test error estimates. However, if you feel that this would take too much time, proceed to the next exercise instead to ensure that you will have enough time for the other exercises as well.

2. As indicated in Computer lab 6, we consider parts of Exercise 11 in Section 4.7 of [ISL](ISL) (pages 171-172), which considers models to predict whether the mileage per gallon value of a car is above or below the median **mpg** value, based on the **Auto** dataset in the **ISLR** library.

   The first step is to ensure access to the data by loading **ISLR** and easy reference to the variables by attaching the **Auto** dataset:
   library(ISLR)
   attach(Auto)

   We need to determine the median value for **mpg** which can be done by writing **median(mpg)**. As we want to refer to this value later, it is a good idea to create a variable that we will call **MedMPG** that contains this value:
   MedMPG=median(mpg)

   The next step is to create the requested binary variable **mpg01** taking value 1 for a car if its **mpg** value is above the median **mpg** and 0 otherwise. This can be very easily done using the **ifelse** function that is very similar to "IF" in Excel: a logical statement is specified in its first argument (i.e. something is stated that can be true or false), the second argument specifies what happens if the statement is true, and the third argument specifies what happens if the statement is false. In this case, we need a comparison for each row in **Auto** between the **mpg** value and **MedMPG**, and if the value is greater, we insert a 1, otherwise 0 to the appropriate place of the **mpg01** vector. All this is done by the following command line:
   mpg01=ifelse(mpg>MedMPG,1,0)

   Note: There is also an alternative way to define **mpg01** in two steps, as follows: we first create a vector that contains only 0's and has the same length as **mpg** in the **Auto** dataset. In the second step, we set value 1 at those indices that correspond to cars having **mpg** value above **MedMPG**. We need two command lines for this:
   mpg01=rep(0,length(mpg))
   mpg01[mpg>MedMPG]=1

   Next, we add the newly defined vector to the Auto dataset and look at the data:

Auto$mpg01=mpg01
View(Auto)
Checking the resulting data set helps to see whether **mpg01** is indeed taking the appropriate values. For example, rows 1-14 have mpg values under 20 while the median value **MedMPG** is 22.75; therefore, the **mpg01** value in these rows should be 0. Conversely, rows 19-24 have **mpg** values above 24, hence these are all above the median value and should have an **mpg01** value of 1.

For part b), asking us to create various plots for identifying potentially relevant predictors of **mpg01**, we can first create scatter plots for all pairs of variables in the Auto data frame:
pairs(Auto)

Looking at the bottom row shows scatter plots with values of other variables on the x-axis and **mpg01** on the y-axis. Which variables that make a separation between 0 and 1 values of **mpg01**? Intuitively, those variables that make a separation should be considered as predictors of **mpg01**.

3. The variables identified above may potentially be useful in predicting **mpg01**. Are there any others? We can check that by looking at the box plots of the other variables versus **mpg01**. We can first try to use the same command as we used in Computer lab 6 for creating the box plots; for example, for the **year** variable:
   plot(year~mpg01)

   Interestingly, this command does not produce a box plot here but rather a scatter plot. Why is that? Because R treats **mpg01** as a numerical variable that happened to have 0 and 1 values (but, for example, its next value could be 0.2 or any other number), rather than a categorical variable that can only have these two values! To get the usual box plots that are generated for categorical variables, we need to help R to know that it should treat **mpg01** as a categorical variable, or, in other words, as a factor. This can be done by using the **as.factor** function, and the following command creates the desired box plots:
   plot(year~as.factor(mpg01))

   Add some axis labels and possibly some colors to this box plot to make it look nicer!

   We can also check how this is shown when viewing the data frame. We add a new variable **Largempg** to **Auto** that contains the same values as **mpg01** and is treated as a factor:
   Auto$Largempg=as.factor(mpg01)
   View(Auto)

   Do you see any difference between how the values for **mpg01** and **Largempg** are displayed?

   Note also that being a factor was an issue only because **mpg01** was defined as **1** or **0** depending on the value of **mpg**; had it been defined to take **"Yes"** or **"No"** instead, R would immediately have recognized it as a categorical variable.

4. After this detour, we can try to create the box plots for the other variables as follows:
   plot(cylinders~as.factor(mpg01))
   plot(year~as.factor(mpg01))

plot(origin~as.factor(mpg01))
plot(name~as.factor(mpg01))

The last plot using name as a predictor is showing too many names values to be useful and does not show evidence that the **name** would be a good predictor of **mpg01**. For the plot with **origin**, it is worth noting that **origin** is a categorical value that is coded as 1, 2 or 3, so we get a more meaningful plot if we ask R to treat both **mpg01** and **origin** as factors:
plot(as.factor(origin)~as.factor(mpg01))

Based on all plots created in this exercise, it is reasonable to believe that each variable except for **name** may help in predicting **mpg01** values. We can check this by fitting single-variable logistic regression models with response **mpg01**, and looking at the summaries of these models; for example, we can consider the following command:
summary(glm(mpg01~year,family="binomial",data=Auto))

Note, however, that several of the predictors are strongly correlated with each other. For example, the scatter plots generated by the **pairs(Auto)** command suggest correlation between **horsepower** and **displacement** or **horsepower** and **weight**. Therefore, some variables may appear important in predicting **mpg01** only because they are correlated with a predictor that may indeed be important. Therefore, for a better understanding of how to predict **mpg01**, we need to consider multivariate models. Furthermore, as we noted that higher degrees of **horsepower** were relevant for predicting the value of **mpg**, we may expect that higher degrees of this variable should be considered as predictors of **mpg01** as well.

5. As requested in part c), we split the data into a training set and a test set. For this, we check how many observations there are by checking the length of the response vector:
length(mpg01)

We have seen that **mpg01** contains 392 elements that we need to split. This could be done in a non-random way as follows:
train=1:196
test=197:392

However, this is not so good, for many reasons. For example, it could happen that the database is ordered in some way, for example by the **mpg** value; in that case, the first half would contain all 0 values for mpg01 while the second half would contain essentially only 1's. It is better to split the set randomly, as shown below. As usual, we set a seed for the random number generator to ensure that we don't get different outputs each time when we run the code:
set.seed(1)
train=sample(392,196)
test=setdiff(1:392,train)

The second line chooses 196 random numbers from the set of all integers from 1 to 392. The **setdiff** function in the last line takes the difference of all integers from 1 to 392 and the train set, i.e. all numbers from 1 to 392 that are not included in **train**. This line is not essential – when fitting models in further steps, it would be enough to use **train** for the training set and **-train** to denote elements of the dataset that are not in **train**.

6. We proceed to part f) of the exercise, asking us to fit a logistic regression model on observations in the training set using a good predictor identified in part b) and quantify the test error for the validation set. For example, we can take **year** whose box plot showed a clear difference between the **year** values of those cars with **mpg** below the median and those with **mpg** above the median. Generally, one could include many more variables in this model, but including **year** suffices for the demonstration of computing test error.

   We fit the model using the subset argument of **glm**:
   YearModelTrain=glm(mpg01~year,family="binomial",data=Auto,subset=train)

   We can then make the predictions using the predict function, and classify those with predicted probability >50% to have value 1:
   YearModelTrain=glm(mpg01~year,family="binomial",data=Auto,subset=train)
   Probs=predict(YearModelTrain,type="response",data=Auto)[test]
   mpg01Prediction=rep(0,196)
   mpg01Prediction[Probs>0.5]=1

   Finally, we can look at a comparison with the mpg01 values in the test set to get an overview of the correctness of predictions:
   table(mpg01Prediction,mpg01[test])

   The values in the diagonal indicate correct classifications, while the other classifications are incorrect. The test error is therefore the sum of the values outside the diagonal divided by the number of observations in the test set. What can you conclude about this predictor based on this result?

7. For feedback related to this specific class, talk to me or use www.menti.com with the code 45 41 36. Also, please fill the course survey once it becomes available!