USASK Data Science Bootcamp T6: Statistical Machine Learning Linear and Logistic Regression R Tutorial

TA: Kyle Gardiner Tutors: Lina Li and Jing Wang

June 13, 2023. (2:00pm-2:30pm)

This document will outline how to fit linear and logistic regression models using R. For both models, we will split the data into train and test datasets to assess the model accuracy.

Firstly, we will install/load in the necessary packages for this tutorial.

```
rm(list=ls(all=TRUE)) # removes objects from the environment to start fresh

library(ISLR2) # contains the dataset used in linear regression section of this tutorial
library(caret) # used to create confusionMatrix for logistic regression
```

We will start with linear regression.

Linear Regression

Linear regression is used for when the outcome variable of interest is continuous.

(a) Loading in the data

```
data(Auto) # adds the "Auto" dataset to the environment

?Auto # Prints dataframe information in the 'help' console

str(Auto) # Displays the structure of the "Auto" dataset
```

```
## 'data.frame':    392 obs. of  9 variables:
##  $ mpg         : num  18 15 18 16 17 15 14 14 14 15 ...
##  $ cylinders   : int  8 8 8 8 8 8 8 8 8 8 ...
##  $ displacement: num  307 350 318 304 302 429 454 440 455 390 ...
##  $ horsepower  : int  130 165 150 150 140 198 220 215 225 190 ...
##  $ weight      : int  3504 3693 3436 3433 3449 4341 4354 4312 4425 3850 ...
##  $ acceleration: num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
##  $ year        : int  70 70 70 70 70 70 70 70 70 70 ...
##  $ origin      : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ name        : Factor w/ 304 levels "amc ambassador brougham",..: 49 36 231 14 161 141 54 223 241 1
##  - attr(*, "na.action")= 'omit' Named int [1:5] 33 127 331 337 355
##   ..- attr(*, "names")= chr [1:5] "33" "127" "331" "337" ...
```

```r
dim(Auto) # Displays the dimensions of the "Auto" dataset (rows x columns)
```

```
## [1] 392   9
```

```r
sum(is.na(Auto)) # checking for missing values that might need to be dealt with before data analysis
```

```
## [1] 0
```

(*b*) **Splitting the Data**
This section will review how to split the data into train and test data sets. We will use the training set to build the linear model. Then we will use the testing set to test how accurate the built model is.

We will split the data as: 80% for training and 20% for testing.

```r
set.seed(1) # set the seed for reproducibility
id<-sample(nrow(Auto),round(nrow(Auto)*0.80)) # Used to identify indexes that fall in train or test
auto.train.data<-Auto[id,] # 'auto.train.data' contains 80% of the original data
auto.test.data<-Auto[-id,] # 'auto.test.data' contains the remaining 20% of the original data
```

(*c*) **Linear Regression Model Building**
We will use 'auto.train.data' to fit the linear regression model. *mpg* (miles per gallon) is the response variable. Use *cylinders*, *displacement*, *horsepower*, *weight*, *acceleration*, *year*, and *origin* as the predictors for this model.

```r
# In R, we can fit a linear regression model using the 'lm()' function
# Make sure that any categorical variables need to be specified using the 'as.factor()' function
linear.model<-lm(mpg~ as.factor(cylinders) + displacement + horsepower + weight + acceleration +
                   as.factor(year) + as.factor(origin), data=auto.train.data)

summary(linear.model) # prints out summary of predictors and their coefficients for this linear model
```

```
##
## Call:
## lm(formula = mpg ~ as.factor(cylinders) + displacement + horsepower +
##     weight + acceleration + as.factor(year) + as.factor(origin),
##     data = auto.train.data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.9637 -1.5392 -0.0802  1.4544 10.7398
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)           30.0944859  2.6104319  11.529  < 2e-16 ***
```

```
## as.factor(cylinders)4  6.9608329  1.5943309   4.366 1.76e-05 ***
## as.factor(cylinders)5  6.0564815  2.4281832   2.494 0.013177 *
## as.factor(cylinders)6  4.8619145  1.8173501   2.675 0.007889 **
## as.factor(cylinders)8  7.0815723  2.1406285   3.308 0.001057 **
## displacement           0.0067361  0.0078259   0.861 0.390086
## horsepower            -0.0370761  0.0147278  -2.517 0.012359 *
## weight                -0.0051102  0.0006985  -7.316 2.49e-12 ***
## acceleration           0.0555042  0.0977940   0.568 0.570770
## as.factor(year)71      1.4174123  0.9397865   1.508 0.132581
## as.factor(year)72     -0.1207293  0.9466633  -0.128 0.898608
## as.factor(year)73     -0.2575799  0.8379578  -0.307 0.758766
## as.factor(year)74      1.4350127  0.9906843   1.449 0.148552
## as.factor(year)75      0.9843405  0.9730993   1.012 0.312593
## as.factor(year)76      1.1588366  0.9291463   1.247 0.213325
## as.factor(year)77      3.3842986  0.9257411   3.656 0.000304 ***
## as.factor(year)78      3.4256811  0.8886095   3.855 0.000142 ***
## as.factor(year)79      4.9296670  0.9496559   5.191 3.93e-07 ***
## as.factor(year)80      9.9836522  1.0011764   9.972  < 2e-16 ***
## as.factor(year)81      6.7940001  0.9810495   6.925 2.79e-11 ***
## as.factor(year)82      8.2557964  0.9596681   8.603 4.92e-16 ***
## as.factor(origin)2     2.0030827  0.5969291   3.356 0.000897 ***
## as.factor(origin)3     1.9209942  0.5726702   3.354 0.000901 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.91 on 291 degrees of freedom
## Multiple R-squared:  0.8732, Adjusted R-squared:  0.8636
## F-statistic:  91.1 on 22 and 291 DF,  p-value: < 2.2e-16
```

(*d*) **Assessing Model Accuracy**

This section will focus on determining how well the built model fits the actual data. Using 'auto.test.data', we will predict the values using the built model. We will then compare the predicted values to the true values to determine accuracy. Measuring the difference between predicted values and true values is Mean Squared Error.

```
# Use the 'predict()' function with the model and 'auto.test.data' to get predicted values
predicted.values<-predict(linear.model, auto.test.data)

# we can now compare these predicted values to the true values of 'mpg'
mean((auto.test.data$mpg-predicted.values)^2)
```

```
## [1] 7.696648
```

When using *cylinders*, *displacement*, *horsepower*, *weight*, *acceleration*, *year*, and *origin* as predictors to build the linear model, we conclude the model to have a MSE of 7.696648.

Now we will explore how to employ logistic regression in R.

Logistic Regression

Logistic regression is used then the outcome variable of interest is binary (ex. 0 and 1, yes and no, affected and unaffected, good and bad, etc.)

(*a*) Loading in the data

```
data(Default) # adds the "Default" dataset to the environment

?Default # Prints dataframe information in the 'help' console

str(Default) # Displays the structure of the "Default" dataset
```

```
## 'data.frame':    10000 obs. of  4 variables:
##  $ default: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ student: Factor w/ 2 levels "No","Yes": 1 2 1 1 1 2 1 2 1 1 ...
##  $ balance: num  730 817 1074 529 786 ...
##  $ income : num  44362 12106 31767 35704 38463 ...
```

```
dim(Default) # Displays the dimensions of the "Default" dataset (rows x columns)
```

```
## [1] 10000     4
```

```
sum(is.na(Default)) # checking for missing values that might need to be dealt with before data analysis
```

```
## [1] 0
```

(*b*) Recoding Outcome Variable

For logistic regression, you often have to recode the outcome variables as 0 and 1 to use in the R functions. In this case, the *default* variable is the outcome of interest and its currently coded as "Yes" or "No". We will recode them as "Yes"= 1 and "No"= 0.

```
Default$default<-ifelse(Default$default=="Yes",1,0)
                  # this replaces the 'default" variable with 0 and 1
                  # if it was previously "Yes", its now 1, otherwise its now 0
```

(*c*) **Splitting the Data**
Again, this section will split the 'Default' data into train and test data set. Same as above, we will split the data 80/20

```
set.seed(1) # Not necessary to state twice but I like to do it

# Used to identify indexes that fall in train or test
id<-sample(nrow(Default),round(nrow(Default)*0.80))

debt.train.data<-Default[id,] # 'debt.train.data' contains 80% of the original data
debt.test.data<-Default[-id,] # 'debt.test.data' contains the remaining 20% of the original data
```

(*d*) **Logistic Regression Model Building**
We will use 'debt.train.data' to fit the logistic regression model. *default* is the response variable. Use *student*, *balance*, and *income* as the predictors for this model.

```
# Instead of using the 'lm()' function like linear regression above, we will use the 'glm()' function.
# NOTE: We don't need to classify the categorical variables 'as.factor()'
# since it was already done in the 'Default' dataset
logistic.model<-glm(default~., family="binomial", data=debt.train.data)
# IMPORTANT: need to specify 'family="binomial"' to make sure the 'glm()' function performs
# logistic regression. Otherwise, it will assume linear regression.


# prints out summary of predictors and their coefficients for this logistic model
summary(logistic.model)
```

```
##
## Call:
## glm(formula = default ~ ., family = "binomial", data = debt.train.data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4736  -0.1389  -0.0543  -0.0202   3.5171
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.094e+01  5.517e-01 -19.834  < 2e-16 ***
## studentYes  -7.282e-01  2.683e-01  -2.714  0.00664 **
## balance      5.713e-03  2.585e-04  22.100  < 2e-16 ***
## income       5.877e-06  9.172e-06   0.641  0.52170
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2313.6  on 7999  degrees of freedom
## Residual deviance: 1231.9  on 7996  degrees of freedom
## AIC: 1239.9
```

```
##
## Number of Fisher Scoring iterations: 8
```

(e) Assessing Model Accuracy

Same as above for linear regression, this section will focus on determining how well the built model fits the actual data. Using 'debt.test.data' we will predict the values using the built model then we will compare the predicted values to the true values to determine accuracy. However, for logistic regression, we often use a confusion matrix to identify how well the model can predict the 0 or 1 cases.

```
# Use the 'predict()' function with the model and 'debt.test.data' to get predicted probabilities
# NOTE: The prediction for logistic regression returns the predicted probabilities.
#Then we classify those probabilities into 0 or 1 based on a set threshold
logistic.predicted.values<-predict(logistic.model, debt.test.data , type="response")
# need to specify 'type="response"' to get predicted probabilities

# converting probabilities into 0 or 1  labels. That is, predicted probabilities >0.5 get assigned 1,
# otherwise, they are assigned 0
debt.test.data$pred.outcome<-ifelse(logistic.predicted.values>0.5,1,0)

# creating a confusion matrix to determine model accuracy
confusionMatrix(as.factor(debt.test.data$default), as.factor(debt.test.data$pred.outcome))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1928    2
##          1   53   17
##
##                Accuracy : 0.9725
##                  95% CI : (0.9644, 0.9792)
##     No Information Rate : 0.9905
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3726
##
##  Mcnemar's Test P-Value : 1.562e-11
##
##             Sensitivity : 0.9732
##             Specificity : 0.8947
##          Pos Pred Value : 0.9990
##          Neg Pred Value : 0.2429
##              Prevalence : 0.9905
##          Detection Rate : 0.9640
##    Detection Prevalence : 0.9650
##       Balanced Accuracy : 0.9340
##
##        'Positive' Class : 0
##
```

```
# need to specify 'as.factor()' to use in the 'confusionMatrix()' function
```

Here we can see that the accuracy of the model is 0.9725. Or in other words, the model is accurately predicting the outcome of whether or not a customer defaulted on their debt 97.25% of the time

This concludes the brief introduction of how to utilize linear and logistic regression in R.