Learning Outcomes: Logistic Regression

Logistic regression is a method for fitting a regression curve, *y = f(x)*, when y is a categorical variable. The typical use of this model is predicting *y* given a set of predictors *x*. The predictors can be continuous, categorical or a mix of both.

The categorical variable *y*, in general, can assume different values. In the simplest case scenario *y* is binary meaning that it can assume either the value 1 or 0. A classic example used in machine learning is email classification: given a set of attributes for each email such as number of words, links and pictures, the algorithm should decide whether the email is spam (1) or not (0). In this case we call the model **"binomial logistic regression"**, since the variable to predict is binary, however, logistic regression can also be used to predict a dependent variable which can assume more than 2 values. In this case we call the model "**multinomial logistic regression**". A typical example for instance, would be classifying films between "Entertaining", "borderline" or "boring".

Logistic regression implementation in R

R makes it very easy to fit a logistic regression model. The function to be called is glm() and the fitting process is not so different from the one used in linear regression.

After completing the following exercises, you should be able to understand and perform below tasks.

1. Building classification model using logistic regression technique.
2. Validating the model results.
3. Evaluation of error metrics.
4. Applying the model on un-seen data.
   i. Split the data into train and test (validation) datasets.
   ii. Comparing the error metrics.
5. Interpretation of the results.
6. Studying the ROC curve.

Problem Statement-1:

Find whether a customer is defaulter or not from the “**Default**” dataset available with the “**ISLR**” package in R.

Objective: Build a classification model to predict whether the given customer is a defaulter or not based on other known factors.

Logistic Regression Model:

Steps to carry out the regression.

1. Import and summarize data.

2. Build a simple logistic regression model - using 1 independent variable.

3. Split the data into training and testing datasets.

4. Fit a multiple variable logistic regression model - Using all significant variables.

5. Predict values on training and testing datasets.

6. Use confusion matrix and ROC to check the model performance.

```
install.packages("ISLR")
install.packages("MASS")
library(ISLR)
library(MASS)
```

1. Import Data.
attach the dataset "Default".

```
	attach(Default)
```

Understand the structure of the data.

```
	?Default
	summary(Default)
	str(Default)
```

Look for missing values in the data. No missing values found.

```
	sapply(Default,function(x) sum(is.na(x)))
```

Unique set of values in each columns.

```
	sapply(Default, function(x) length(unique(x)))
```

2. Build a simple model using "income" attribute.
The only variable income has very small estimate.
The significance of this variable also indicates that very small 0.471
So this variable doesn't help us in building the model.

```
	simple.model <- glm(default~income, data = Default, family = binomial())

	summary(simple.model)
```

We stored our output in simple.model
By specifying type = response, We instruct R to response with the probabilities using the specified model.

```
	simple.model.pred.prob <- predict(simple.model, type = "response")
```

Visualize the model

```
	plot(simple.model.pred.prob, Churned)
```

Confirms that our model wasn't very good - If our model was good all we could see that non-defaulters cluster on the left side and defaulters cluster on the right side.
Most of the defaulters are lying in a very low probability range also

Non defaulters are spread out. - Model wasn't able to find them very good.

```
plot(simple.model.pred.prob, jitter(as.numeric(Churned)))
```

Build Comprehensive model.

3. Split data into training and validation datasets.

```
set.seed(123)
```

list of random row-ids from the customer_Data dataset.

```
sub_seq <- sample(nrow(Default), floor(nrow(Default) * 0.7))

training_data <- Default[sub_seq,]

validation_data <- Default[-sub_seq,]
```

Verify training and validation datasets.

Default No : 6754 : Yes – 246

```
summary(training_data)
```

Default No : 2913 : Yes – 87

```
summary(validation_data)
```

4. Build a more comprehensive model

Student and balance are significant. Income is in-significant.

```
model.new <- glm(default~., data = training_data, family = binomial())

summary(model.new)
```

Explore the model by dropping the insignificant variables with-out hurting the model.

Perform step wise regressing using stepAIC function.

```
model.step <- stepAIC(model.new)

summary(model.step)
```

Use model.step to predict the churned or not for training and validation datasets.

```
training_data$pred.prob     <- predict(model.step, type = "response")

validation_data$pred.prob   <- predict(model.step, type = "response",
newdata = validation_data)
```

```
plot(training_data$pred.prob, jitter(as.numeric(training_data$Churned)))

plot(validation_data$pred.prob, jitter(as.numeric(validation_data$Churned)))
```

Confusion Matrix
Convert the probability values to 1 and 0 -
0.5 - Arbitrary choice guided by chart - all the zeroes are highly concentrated at the left - Take a cut-off point as 0.05 to get a quick estimate of how the model is performing -

```
training_data$class   <- as.factor(ifelse(training_data$pred.prob > 0.5, 1, 0))

validation_data$class <- as.factor(ifelse(validation_data$pred.prob > 0.5, 1, 0))

conf.train <- table(training_data$default, training_data$class)

conf.valid <- table(validation_data$default, validation_data$class)
```

Plot ROC curve
The ROC curve is used to visualize the performance of the model.
The area under this curve - represents how good is the model is,
the closer the curve to 100% true positive rate.

```
library(ROCR)

train_pred <- prediction(training_data$pred.prob, training_data$Churned)

train_perf <- performance(train_pred, measure = "tpr", x.measure = "fpr")

plot(train_perf)

validation_pred <- prediction(validation_data$pred.prob,

validation_data$Churned)

validation_pref <- performance(validation_pred, measure = "tpr", x.measure =
"fpr")

plot(validation_pref)
```

```
auc.train <- performance(train_pred,"auc")
auc_train <- as.numeric(auc.train@y.values)
auc_train

auc.test <- performance(validation_pred, "auc")
auc_test <- as.numeric(auc.test@y.values)
auc_test
```

Compute error metrics precision, accuracy and recall

For training data.

Calculate Accuracy Percentage

What percent of your predictions were correct? TN + TP / TOTAL

```
accuracy <- (sum(diag(conf.train))/sum(conf.train))*100

cat("accuracy is --->", accuracy)
```

Calculate Recall Percentage

What percent of the positive cases did you catch? TP / FN + TP

```
recall=(conf.train[2,2]/(conf.train[2,1]+conf.train[2,2]))*100

cat("recall is --->", recall)
```

Calculate Precission Percentage

What percent of positive predictions were correct? TP / TP + FP

```
precision=(conf.train[2,2])/(conf.train[1,2]+conf.train[2,2])*100

cat("precision is --->", precision)
```

Compute error metrics precision, accuracy and recall

For validation data.

Calculate Accuracy Percentage

What percent of your predictions were correct? TN + TP / TOTAL

```
accuracy <- (sum(diag(conf.valid))/sum(conf.valid))*100

cat("accuracy is --->", accuracy)
```

Calculate Recall Percentage

What percent of the positive cases did you catch? TP / FN + TP

```
recall=(conf.valid[2,2]/(conf.valid[2,1]+conf.valid[2,2]))*100

cat("recall is --->", recall)
```

Calculate Precission Percentage

What percent of positive predictions were correct? TP / TP + FP

```
precision=(conf.valid[2,2])/(conf.valid[1,2]+conf.valid[2,2])*100

cat("precision is --->", precision)
```

Problem Statement-2:

We will be working on the Titanic (train) dataset.

The dataset (train) is a collection of data about some of the passengers (891 to be precise), and the goal is to predict the survival (either 1 if the passenger survived or 0 if they did not) based on some features such as the class of service, the sex, the age etc. As you can see, we are going to use both categorical and continuous variables.

Import data

When working with a real dataset we need to take into account the fact that some data might be missing or corrupted, therefore we need to prepare the dataset for our analysis.
As a first step we load the csv data using the read.csv() function.
Make sure that the parameter na.strings is equal to c("") so that each missing value is coded as a NA. This will help us in the next steps.

```
training_Data <- read.csv('train.csv', header=T, na.strings=c(""))
```

Understand the structure of the data.

```
summary(training_Data)

str(training_Data)
```

Check for missing values and look how many unique values there are for each variable - using the sapply() function which applies the function passed as argument to each column of the dataframe.

```
sapply(training_Data,function(x) sum(is.na(x)))
```

Unique set of values in each columns.

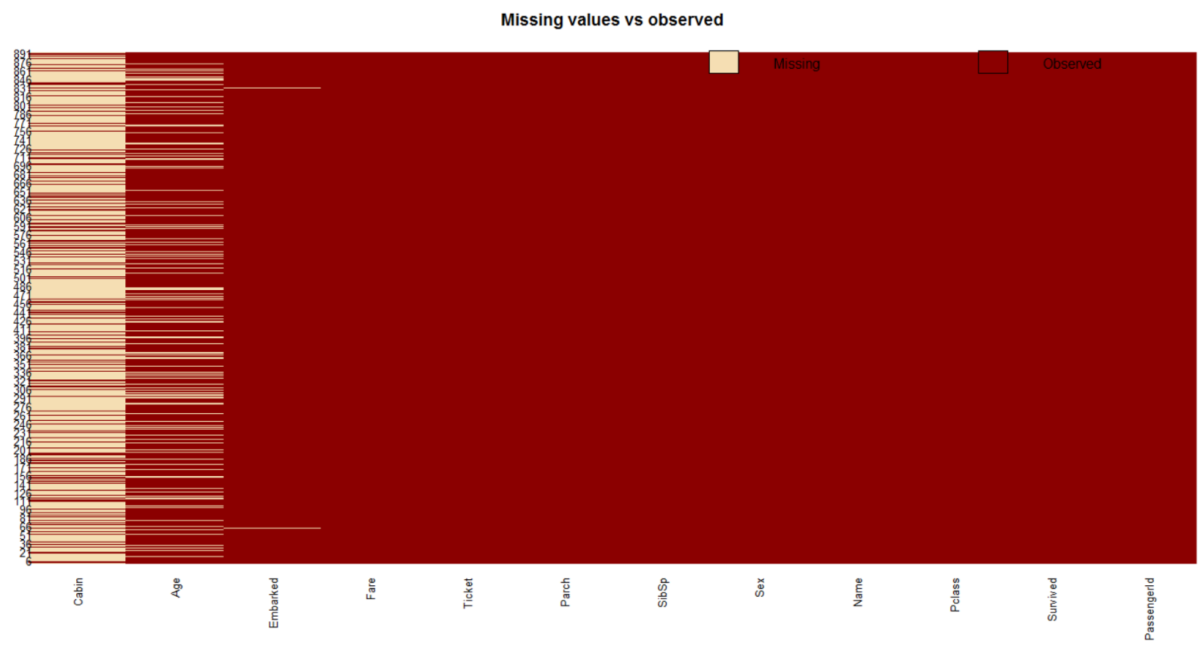
```
sapply(training_Data, function(x) length(unique(x)))
```

Visualization on the missing values might be helpful:
The Amelia package has a special plotting function missmap() that will plot your dataset and highlight missing values:

```
#install.packages("Amelia")

library(Amelia)

missmap(training_Data, main = "Missing values vs observed")
```



The variable cabin has too many missing values, better not to use it. We will also drop PassengerId since it is only an index and Ticket. Using the subset() function we subset the original dataset selecting the relevant columns only.

```
model_Data <-   subset(training_Data,select=c(2,3,5,6,7,8,10,12))
```

The variable cabin has too many missing values, better not to use it.
We will also drop PassengerId since it is only an index and Ticket.
Using the subset() function we subset the original dataset selecting the relevant columns only.

```
model_Data <-   subset(training_Data,select=c(2,3,5,6,7,8,10,12))
```

Taking care of the missing values

Fill the other missing values. R can easily deal with them when fitting a generalized linear model by setting a parameter inside the fitting function. However, preferable to replace the NAs "by hand", when is possible. There are different ways to do this, a typical approach is to replace the missing values with the average, the median or the mode of the existing one. We'll be using the average.

```
model_Data$Age[is.na(model_Data$Age)] <- mean(model_Data$Age,na.rm=T)
```

As far as categorical variables are concerned, using the read.table() or read.csv() by default will encode the categorical variables as factors. A factor is how R deals categorical variables. We can check the encoding using the following lines of code

```
is.factor(model_Data$Sex)
is.factor(model_Data$Embarked)
is.factor(model_Data$Survived)
model_Data$Survived <- as.factor(as.character(model_Data$Survived))
```

For the missing values in Embarked, since there are only two, we will discard those two rows (we could also have replaced the missing values with the mode and keep the datapoints).

```
model_Data <- model_Data[!is.na(model_Data$Embarked),]
rownames(model_Data) <- NULL
```

Model Building

We split the data into two chunks: training and testing set.
The training set will be used to fit our model which we will be testing over the testing set.

```
set.seed(123)
```

list of random row-ids from the customer_Data dataset.

```
sub_Seq         <- sample(nrow(model_Data), floor(nrow(model_Data) * 0.7))
training_data   <- model_Data[sub_Seq,]
validation_data <- model_Data[-sub_Seq,]
```

Verify training and validation datasets.
Survived Yes - 237 : No - 385

```
summary(training_data)
```

Survived Yes - 103 : No - 164

```
summary(validation_data)
```

Build the model

```
model <- glm(Survived~., data = training_data, family = binomial())
summary(model)
```

Explore the model by dropping the insignificant variables with-out hurting the model.
Perform step wise regressing using stepAIC function.

```
model.step <- stepAIC(model)
summary(model.step)
```

Use model.step to predict the default or not for training and validation datasets.

```
training_data$pred.prob <- predict(model.step, type = "response")
```

```
validation_data$pred.prob <- predict(model.step, type = "response", newdata
= validation_data)

plot(training_data$pred.prob, jitter(as.numeric(training_data$Survived)))

plot(validation_data$pred.prob, jitter(as.numeric(validation_data$Survived)))
```

Confusion Matrix
Convert the probabilty values to 1 and 0 -
0.5 - Arbitrary choice guided by chart -
Take a cut-off point as 0.5 to get a quick estimate of how the model is performing -

```
training_data$class <- ifelse(training_data$pred.prob > 0.5, 1, 0)
validation_data$class <- ifelse(validation_data$pred.prob > 0.5, 1, 0)

conf.train <- table(training_data$Survived, training_data$class)
conf.valid <- table(validation_data$Survived, validation_data$class)
```

Plot ROC curve
The ROC curve is used to visualize the performance of the model.
The area under this curve - represents how good is the model is,
the closer the curve to 100% true positive rate.

```
library(ROCR)
train_pred <- prediction(training_data$pred.prob, training_data$Survived)
train_perf <- performance(train_pred, measure = "tpr", x.measure = "fpr")
plot(train_perf)

validation_pred <- prediction(validation_data$pred.prob,
validation_data$Survived)
validation_pref <- performance(validation_pred, measure = "tpr", x.measure =
"fpr")
plot(validation_pref)

auc.train <- performance(train_pred,"auc")
auc_train <- as.numeric(auc.train@y.values)
auc_train
```

```
auc.test <- performance(validation_pred, "auc")
auc_test <- as.numeric(auc.test@y.values)
auc_test
```

Compute error metrics precision, accuracy and recall
For training data.
Calculate Accuracy Percentage
What percent of your predictions were correct? TN + TP / TOTAL

```
accuracy <- (sum(diag(conf.train))/sum(conf.train))*100
cat("accuracy is --->", accuracy)
```

Calculate Recall Percentage
What percent of the positive cases did you catch? TP / FN + TP

```
recall=(conf.train[2,2]/(conf.train[2,1]+conf.train[2,2]))*100
cat("recall is --->", recall)
```

Calculate Precission Percentage
What percent of positive predictions were correct? TP / TP + FP

```
precision=(conf.train[2,2])/(conf.train[1,2]+conf.train[2,2])*100
cat("precision is --->", precision)
```

Compute error metrics precision, accuracy and recall
For validation data.
Calculate Accuracy Percentage
What percent of your predictions were correct? TN + TP / TOTAL

```
accuracy <- (sum(diag(conf.valid))/sum(conf.valid))*100
cat("accuracy is --->", accuracy)
```

Calculate Recall Percentage
What percent of the positive cases did you catch? TP / FN + TP

```
recall=(conf.valid[2,2]/(conf.valid[2,1]+conf.valid[2,2]))*100
cat("recall is --->", recall)
```

Calculate Precission Percentage
What percent of positive predictions were correct? TP / TP + FP

```
precision=(conf.valid[2,2])/(conf.valid[1,2]+conf.valid[2,2])*100
cat("precision is --->", precision)
```

Problem Statement-3:

A large child education toy company which sells edutainment tablets and gaming systems both online and in retail stores wanted to analyze the customer data. They are operating from last few years and maintaining all transactional information data. The given data '**CustomerData.csv**' is a sample of customer level data extracted and processed for the analysis from various set of transactional files.

Objective: Build a classification model to predict whether the given customer will churn or not churn based on other known factors.

Logistic Regression Model:

Steps to carry out the regression.

1. Import and summarize data.

2. Build a simple logistic regression model - using 1 independent variable.

3. Split the data into training and testing datasets.

4. Fit a multiple variable logistic regression model - Using all significant variables.

5. Predict values on training and testing datasets.

6. Use confusion matrix and ROC to check the model performance.

library(MASS)

1. Import Data.

```
customer_Data <- read.csv(file = "CustomerData.csv", header = T)
```

Understand the structure of the data.

```
str(customer_Data)
```

Look for missing values in the CustomerData dataset. No missing values found.

```
sapply(customer_Data, function(x) sum(is.na(x)))
```

Unique set of values in each columns.

```
sapply(customer_Data, function(x) length(unique(x)))
```

Pre-Process Data.
Drop the attribute "CustomerID".

```
customer_Data <- customer_Data[,-c(1)]
```

Convert "City" as factor variable.

```
customer_Data$City <- as.factor(as.character(customer_Data$City))
```

Target attribute is : Churned. Convert this as factor variable.

```
customer_Data$Churned <- as.factor(as.character(customer_Data$Churned))
```

2. Build a simple model using "FrequencyOFPlay" attribute.
The only variable frequency of play has very small estimate.
So this variable doesn't help us in building the model.

```
simple.model <- glm(Churned~FrequencyOFPlay, data = customer_Data,
family = binomial())
summary(simple.model)
```

We stored our output in simple.model
By specifying type = response, We instruct R to response with the probabilities using the specified model.

```
simple.model.pred.prob <- predict(simple.model, type = "response")
```

Visualize the model

```
plot(simple.model.pred.prob, Churned)
```

Confirms that our model wasn't very good -
If our model was good all we could see that non-Churned cluster on the one side and Churned on the other side

Most of the non-churned are lying in a very high probability range also
Model wasn't able to find them very good.
For a better plot apply jitter function.

```
plot(simple.model.pred.prob, jitter(as.numeric(Churned)))
```

Build Comprehensive model.
3. Split data into training and validation datasets.

```
set.seed(123)
```

list of random row-ids from the customer_Data dataset.

```
sub_Seq        <- sample(nrow(customer_Data), floor(nrow(customer_Data) *
0.7))

training_data   <- customer_Data[sub_Seq,]

validation_data <- customer_Data[-sub_Seq,]
```

Verify training and validation datasets.
Churned Yes - 1141 : No - 1105

```
summary(training_data)
```

Churned Yes - 498 : No - 465

```
summary(validation_data)
```

4. Build a more comprehensive model

```
model.new <- glm(Churned~., data = training_data, family = binomial())

summary(model.new)
```

Explore the model by dropping the insignificant variables with-out hurting the model.
Perform step wise regressing using stepAIC function.

```
model.step <- stepAIC(model.new)

summary(model.step)
```

Use model.step to predict the churned or not for training and validation datasets.

```
training_data$pred.prob     <- predict(model.step, type = "response")

validation_data$pred.prob   <- predict(model.step, type = "response",
newdata = validation_data)

plot(training_data$pred.prob, jitter(as.numeric(training_data$Churned)))

plot(validation_data$pred.prob, jitter(as.numeric(validation_data$Churned)))
```

Confusion Matrix

Convert the probability values to 1 and 0 -
0.5 - Arbitrary choice guided by chart -

```
training_data$class   <- as.factor(ifelse(training_data$pred.prob > 0.5, 1, 0))

validation_data$class <- as.factor(ifelse(validation_data$pred.prob > 0.5, 1, 0))

conf.train <- table(training_data$Churned, training_data$class)

conf.valid <- table(validation_data$Churned, validation_data$class)
```

Plot ROC curve

The ROC curve is used to visualize the performance of the model.
The area under this curve - represents how good is the model is,
the closer the curve to 100% true positive rate.

```
library(ROCR)

train_pred <- prediction(training_data$pred.prob, training_data$Churned)

train_perf <- performance(train_pred, measure = "tpr", x.measure = "fpr")

plot(train_perf)

validation_pred <- prediction(validation_data$pred.prob,

validation_data$Churned)
```

```
validation_pref <- performance(validation_pred, measure = "tpr", x.measure =
"fpr")

plot(validation_pref)

auc.train <- performance(train_pred,"auc")
auc_train <- as.numeric(auc.train@y.values)
auc_train

auc.test <- performance(validation_pred, "auc")
auc_test <- as.numeric(auc.test@y.values)
auc_test
```

Compute error metrics precision, accuracy and recall
For training data.
Calculate Accuracy Percentage
What percent of your predictions were correct? TN + TP / TOTAL

```
accuracy <- (sum(diag(conf.train))/sum(conf.train))*100

cat("accuracy is --->", accuracy)
```

Calculate Recall Percentage
What percent of the positive cases did you catch? TP / FN + TP

```
recall=(conf.train[2,2]/(conf.train[2,1]+conf.train[2,2]))*100

cat("recall is --->", recall)
```

Calculate Precission Percentage
What percent of positive predictions were correct? TP / TP + FP

```
precision=(conf.train[2,2])/(conf.train[1,2]+conf.train[2,2])*100

cat("precision is --->", precision)
```

Compute error metrics precision, accuracy and recall

For validation data.

Calculate Accuracy Percentage

What percent of your predictions were correct? TN + TP / TOTAL

```
accuracy <- (sum(diag(conf.valid))/sum(conf.valid))*100

cat("accuracy is --->", accuracy)
```

Calculate Recall Percentage

What percent of the positive cases did you catch? TP / FN + TP

```
recall=(conf.valid[2,2]/(conf.valid[2,1]+conf.valid[2,2]))*100

cat("recall is --->", recall)
```

Calculate Precission Percentage

What percent of positive predictions were correct? TP / TP + FP

```
precision=(conf.valid[2,2])/(conf.valid[1,2]+conf.valid[2,2])*100

cat("precision is --->", precision)
```