Objective:

In this session, you will learn to build logistic regression model, validate your model and interpret the results, and see the performance using evaluation metrics. We will also see how to look for evidences which can guide us in building a better model.

Key takeaways:

- Reading and pre-processing supervised dataset.
- Building logistic regression model using glm()
- Interpreting the results
- Model Evaluation Methodologies
    - Confusion Matrix
        - Accuracy, Recall and Precision
    - Receiver Operating Characteristic(ROC) Curve
- Tuning the model

Problem Statement:

The "Bank.txt" file consists of the data related to direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, to access if the product (bank term deposit) would be (or not) subscribed. The data and attribute description are provided. The classification goal is to predict if the client will subscribe (yes/no) a term deposit (variable y).

Reading and pre-processing the data

Set the working directory

Make sure the dataset is in your current working directory. Else you can change your working directory using the "setwd(path)" function OR setwd(choose.dir()) OR use ctrl+ shift +H

Import the data into R

The data given to you is in a .txt file, where values on each line of the dataset are separated by ";". Read in the data using the "read.csv()" function

```
bank_data <- read.csv(file = "bank.txt", header = T,sep = ";")
```

Understand the data and preform required pre-processing steps

- Structure and summary of the data

Use the str() function to get the dimensions and types of attributes in the dataset and use summary() function to understand the distribution of variables

```
str(bank_data); summary(bank_data)
```

- Handling missing values
  - Check the number of missing values in the data frame

```
sum(is.na(bank_data)

#No missing values in the data
```

Note: Since all the data types of the variables were found to be appropriate, there is no need for data type conversion; Also, there are no missing values

Splitting the data into Train and Validation

Split the data 70/30 into train and validation sets by using → Stratified Sampling. We can do that manually or by using the createDataPartition() function from the caret package

Manually

```
# set.seed(123)

# rows = seq(1,nrow(Bank),1)

# set.seed(123)

# trainRows = sample(rows,(70*nrow(Bank))/100)

# train = Bank[trainRows,]

# val = Bank[-trainRows,]

#str(train_data)
```

Using caret package

```
# Note: caret stands for Classification and Regression Training

# set.seed(123)

# train_rows <- createDataPartition(bank_data$y, p = 0.7, list = F)

# The argument "y" to the function is the response variable

# The argument "p" is the percentage of data that goes to training

# The argument "list" should be input a boolean (T or F). Remember to put list = F, else
the output is going to be a list and your data can't be subsetted with it

# train_data <- bank_data[train_rows, ]

# val_data <- bank_data[-train_rows, ]

#str(train_data)
```

Building a basic Logistic Regression Model

Model Object

- Use the glm() function to build a basic model; Build a model using all the variables, excluding the response variable, in the dataset

```
log_reg <- glm(y~., data = train_data, family = binomial)
```

- Get the summary of the model and understand the output

```
summary(log_reg)
```

Predicting on train data

```
prob_ train <- predict(LogReg, type="response")

# By default if no data set is mentioned, training data is used

#prob_val <- predict(LogReg, val, type="response") # Predicting on validation data
```

Note: The predict() function on the "glm" object of "binomial" family gives a probability score between 0 and 1, NOT the original levels (0 and 1) of the response variable

Choosing a cutoff point

- As predict() function on the "glm" gives a probability score between 0 and 1, we must first choose a cutoff point for getting to the original levels of the response variables
- To choose the cutoff point we will use the train data, as validation data should not be used to make any decisions regarding the model

Manually

Manually choose the threshold; Here, we take it as 0.5

```
# pred_class <- ifelse(prob > 0.5, 1, 0)

# table(train$outcome,pred_class)
```

Using ROC curve

```
#library(ROCR)

pred <- prediction(prob_train, train_data$y)
```

Note: The prediction object takes the probability scores and the original levels for the data as input; The prediction object contains a list of predictions (probability scores), original class labels, #cutoffs, #false positives, true positives, true negatives, false negatives, No. of positive #predictions and No. of #negative predictions corresponding to these cutoffs. Class distribution #in the dataset.

- Extract performance measures (True Positive Rate and False Positive Rate) using the "performance()" function from the ROCR package
- The performance() function from the ROCR package helps us extract metrics such as True positive rate, False positive rate etc. from the prediction object, we created above.
- Two measures (y-axis = tpr, x-axis = fpr) are extracted

```
perf <- performance(pred, measure="tpr", x.measure="fpr")

Plot the ROC curve using the extracted performance measures (TPR and FPR)

plot(perf, col=rainbow(10), colorize=T, print.cutoffs.at=seq(0,1,0.05))
```

- Extract the AUC score of the ROC curve and store it in a variable named "auc"
- Use the performance() function on the prediction object created above using the ROCR package, to extract the AUC score

```
perf_auc <- performance(pred, measure="auc")
```

- Access the auc score from the performance object

```
auc <- perf_auc@y.values[[1]]

print(auc)
```

Deciding on Cutoff Value

Based on the trade-off between TPR and FPR, depending on the business domain, a call on the cutoff has to be made. Here, a cutoff of 0.1 can be chosen

Predictions on val data

After choosing a cutoff value of 0.1, let's predict the class labels on the validation data using our model

```
prob_val <- predict(log_reg_step, val_data, type = "response")

preds_val <- ifelse(prob_val > 0.1, "yes", "no")

table(preds_val)
```

Evaluation Metrics for classification

Manual Computation using Confusion Matrix

Create a confusion matrix using the table() function

```
conf_matrix <- table(val_data$y, preds_val)

print(conf_matrix)

specificity <- conf_matrix[1, 1]/sum(conf_matrix[1, ])

sensitivity <- conf_matrix[2, 2]/sum(conf_matrix[2, ])

accuracy <- sum(diag(conf_matrix))/sum(conf_matrix)
```

Automated Computation through Caret

Evaluation metrics for classification can be accessed through the "confusionMatrix()" function from the caret package

```
library(caret)

# Using the argument "Positive", we can get the evaluation metrics according to our positive
#referene level

confusionMatrix(preds_val, val_data$y, positive = "yes")
```

Tuning the model

Check the model summary to check for any insignificant variables

```
summary(log_reg_step)
```

Use vif to find any multi-collinearity

```
library(car)

log_reg_step_vif = vif(log_reg_step)

log_reg_step_vif
```

Improve the model using stepAIC

```
library(MASS)

log_reg_step = stepAIC(log_reg, direction = "both")
```