# Class Example – Supervised Learning

Consider the Breast Cancer dataset (BreastCancer.csv). You would like to model and predict if a given specimen is benign or malignant, based on the other cell features present in the data.

### Import and include packages

In the cell below, install and include all necessary packages

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

### Import the necessary data

In the cell below, import the necessary dataset(s)

```
bc <- read.csv("BreastCancer.csv")
head(bc)
```

```
##   X Cl.thickness Cell.size Cell.shape Marg.adhesion Epith.c.size Bare.nuclei
## 1 1            5         1          1             1            2           1
## 2 2            5         4          4             5            7          10
## 3 3            3         1          1             1            2           2
## 4 4            6         8          8             1            3           4
## 5 5            4         1          1             3            2           1
## 6 6            8        10         10             8            7          10
##   Bl.cromatin Normal.nucleoli Mitoses     Class
## 1           3               1       1    benign
## 2           3               2       1    benign
## 3           3               1       1    benign
## 4           3               7       1    benign
## 5           3               1       1    benign
## 6           9               7       1 malignant
```

### Step 1: Dummy Variables

- Response variable is a binary qualitative variable.
- These variables need to be represented numerically in R to make analysis possible.
- Dummy variables can be used as a numerical “stand-in” for qualitative variables.
- Typically, a dummy variable takes on the value of 0 or 1 to indicate the absence or presence of a factor.
- Define a dummy variable using the ifelse( ) function.

- Convert variable type for new dummy variable from numeric to factor.

For this example, - Indicate a benign response with a 0 and a malignant response with a 1.

```
bc$Class <- ifelse(bc$Class == "benign", 0, 1)
#bc$Class <- ifelse(bc$Class == "malignant", 1, 0)
head(bc)
```

```
##   X Cl.thickness Cell.size Cell.shape Marg.adhesion Epith.c.size Bare.nuclei
## 1 1            5         1          1             1            2           1
## 2 2            5         4          4             5            7          10
## 3 3            3         1          1             1            2           2
## 4 4            6         8          8             1            3           4
## 5 5            4         1          1             3            2           1
## 6 6            8        10         10             8            7          10
##   Bl.cromatin Normal.nucleoli Mitoses Class
## 1           3               1       1     0
## 2           3               2       1     0
## 3           3               1       1     0
## 4           3               7       1     0
## 5           3               1       1     0
## 6           9               7       1     1
```

```
class(bc$Class)
```

```
## [1] "numeric"
```

```
bc$Class <- factor(bc$Class, levels = c(0,1))
head(bc$Class)
```

```
## [1] 0 0 0 0 0 1
## Levels: 0 1
```

### Step 2: Train/Test Split

- Randomly split the data into a training and test set.

For this example,

- Randomly split the data into a training and test set using a 80/20% split.
- Use a seed value of 404.

```
set.seed(404)
split <- round(nrow(bc)*0.80)
train_ind <- sample(1:nrow(bc),
                   split,
                   replace = FALSE)
trainData <- bc[train_ind,]
testData <- bc[-train_ind,]
head(trainData)
```

```
##       X Cl.thickness Cell.size Cell.shape Marg.adhesion Epith.c.size
## 441 456           10         2          2             1            2
## 536 551            3         1          1             1            2
## 570 585            5         1          1             6            3
## 39   40            2         5          3             3            6
## 552 567            3         1          2             1            2
## 109 111            1         3          1             2            2
##     Bare.nuclei Bl.cromatin Normal.nucleoli Mitoses Class
## 441           6           1               1       2     1
## 536           1           2               1       1     0
## 570           1           1               1       1     0
## 39            7           7               5       1     1
## 552           1           3               1       1     0
## 109           2           5               3       2     0
```

### Step 3: Fit the Logistic Regression Model

- Use the glm( ) function in R to fit a logistic regression model. The general syntax for this model is,
  - glm(y~., data = my_dataset, family = 'binomial')
  - glm(y~x1+x2+x3, data = my_dataset, family = 'binomial')

For this example, - Build a logistic regression model with Class being our response variable and the remaining variables as predictors.

```
logistic_mod <- glm(Class ~ ., data = trainData, family = "binomial")
summary(logistic_mod)
```

```
##
## Call:
## glm(formula = Class ~ ., family = "binomial", data = trainData)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.3735  -0.0835  -0.0378   0.0207   2.0224
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -10.608439   1.697683  -6.249 4.14e-10 ***
## X                -0.002028   0.001844  -1.100 0.271377
## Cl.thickness      0.698433   0.187110   3.733 0.000189 ***
## Cell.size         0.031699   0.234893   0.135 0.892651
## Cell.shape        0.317733   0.249407   1.274 0.202680
## Marg.adhesion     0.295125   0.145972   2.022 0.043198 *
## Epith.c.size      0.272023   0.197751   1.376 0.168951
## Bare.nuclei       0.385287   0.108562   3.549 0.000387 ***
## Bl.cromatin       0.442263   0.205410   2.153 0.031313 *
## Normal.nucleoli   0.107847   0.118764   0.908 0.363837
## Mitoses           0.245077   0.328227   0.747 0.455262
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```

```
##
##     Null deviance: 703.096  on 545  degrees of freedom
## Residual deviance:  71.212  on 535  degrees of freedom
## AIC: 93.212
##
## Number of Fisher Scoring iterations: 8
```

```r
logistic_model_ex <- glm(Class ~ Cl.thickness + Cell.size + Cell.shape,
                         data = trainData, family = "binomial")
summary(logistic_model_ex)
```

```
##
## Call:
## glm(formula = Class ~ Cl.thickness + Cell.size + Cell.shape,
##     family = "binomial", data = trainData)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.8293  -0.1680  -0.0881   0.0128   2.2697
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -8.3155     0.8634  -9.631  < 2e-16 ***
## Cl.thickness   0.6481     0.1267   5.117 3.11e-07 ***
## Cell.size      0.7616     0.2147   3.547 0.000389 ***
## Cell.shape     0.7081     0.1968   3.597 0.000321 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 703.10  on 545  degrees of freedom
## Residual deviance: 130.05  on 542  degrees of freedom
## AIC: 138.05
##
## Number of Fisher Scoring iterations: 8
```

### Step 4: Predictions

- Use the predict( ) function in R to make predictions based on the Logistic Regression model in Step 3.
- The general syntax for the predict( ) function is,
  - predict(model, my_dataset, type = 'response')
- The predict( ) function returns probabilities.
- Therefore, we need to set a threshold value to classify each predicted outcome to a specific category.
- To classify the probabilities into two groups, use the ifelse( ) function in R.

For this example, - Predict the response variables for the test set.

```r
# Predict the response variables for the test set
pred <- predict(logistic_mod, newdata = testData, type = "response")
head(pred)
```

```
##           9          13          14          15          24          26 
## 0.002785780 0.256632907 0.002693492 0.999851238 0.001220712 0.002480340
```

```
# Recode factors
y_pred_num <- ifelse(pred > 0.5, 1, 0)
y_pred <- factor(y_pred_num) # convert the numeric predictors response to factors
y_act <- testData$Class

head(y_pred)
```

```
##  9 13 14 15 24 26 
##  0  0  0  1  0  0 
## Levels: 0 1
```

```
head(y_act)
```

```
## [1] 0 1 0 1 0 0
## Levels: 0 1
```

### Step 5

- Use the confusionMatrix( ) function from the **caret** package to obtain the confusion matrix of the observed and predicted classes.
- The general syntax for the confusionMatrix( ) function is,
- confusionMatrix(data, reference, positive = “positive_factor_level”)

For this example, - Calculate the performance metrics.

```
# Performance metrics
confusionMatrix(data = y_pred, reference = y_act, positive = "1")
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction  0  1
##          0 85  5
##          1  1 46
##                                           
##                Accuracy : 0.9562          
##                  95% CI : (0.9071, 0.9838)
##     No Information Rate : 0.6277          
##     P-Value [Acc > NIR] : <2e-16          
##                                           
##                   Kappa : 0.9048          
##                                           
##  Mcnemar's Test P-Value : 0.2207          
##                                           
##             Sensitivity : 0.9020          
##             Specificity : 0.9884          
##          Pos Pred Value : 0.9787          
##          Neg Pred Value : 0.9444
##              Prevalence : 0.3723
##          Detection Rate : 0.3358
##    Detection Prevalence : 0.3431
##       Balanced Accuracy : 0.9452
##
##        'Positive' Class : 1
##
```