Implementing Logistic Regression

Main Objectives

- Implement the *Logistic Regression* algorithm using *Gradient Descent*
- Apply the model to the **Auto** data set and new variable to predict **high** given **horsepower**, **weight**, **year**, and **origin**

Script 1 : Implementing the Algorithm

In Script 1, we implement the Logistic Regression model using gradient descent. We predict the probabilities of the labels being either 1 or 0. The probabilities are given by:

$\sigma(z)$ where $z = w^T X$, w = coefficients, and X = attributes

To figure out how to set our ideal coefficients we use maximum likelihood estimation which is equivalent of the negative log likelihood estimation given by:

J(w) $= -\sum_{i=1}^{n}[y_n log(f(X_n, w)) + (1 - y_n) log(1 - f(X_n, w))]$ where $f(X_n, w) = \sigma(w^T X_n)$

We minimise the above loss function by using gradient descent and calculate the new coefficients as follows:

$w = w - \alpha * \frac{\partial J(w)}{\partial w}$

where α is the learning rate and,

$\frac{\partial J(w)}{\partial w} = -\sum_{i=1}^{n}[y_n - f(X_n, w)]x_n$

The implementation for this can be found in GD1.R

Script 2 : Loading and Processing Data

In Script 2, we load the **Auto** dataset into R and then remove the rows containing missing values. Then a new variable **high** is created that takes values based on the **mpg** value (as seen in code). We are asked to predict **high** for given **horsepower**, **weight**, **year** and **origin**.

Since **origin** is a qualitative variable taking values *1*, *2* or *3*, we need to create dummy variables for it before feeding it to our model. So, we create two dummy variables **origin2** and **origin3** with value *1* as reference.

Then we normalize the variables **horsepower**, **weight** and **year** using z-score normalization and assign their values to **X** which will now serve as our new processed dataset containing only the attributes.

The values of high are stored in **y** (the labels).

Script 3: Making Predictions

In Script 3, we split our processed dataset **X** randomly into two sets **X_train** and **X_test** which will serve as our training and test sets.

Then we fit the model that we've implement to the training data as well as fitting the model to the built-in model in R for Logistic Regression and compare the coefficients. We see that the coefficients generated by our model are very similar to the ones generated by the built-in model

Comparing the coefficients of our model and the inbuilt model

```
t(my_model$best_params)
```

```
##      Intercept horsepower    weight     year  origin2   origin3
## [1,] -1.545452 -0.9536323 -3.962205 1.950374 1.627676 0.7074928
```

```
built_in_model$coefficients
```

```
## (Intercept)  horsepower      weight        year     origin2     origin3
##  -1.5454393  -0.9535524  -3.9623029   1.9503971   1.6276751   0.7074371
```

Thereafter, we make predictions on the training and test sets using the ideal values generated by our model for the coefficients/parameters.

```
## Setting the values of the best params after training the model
w <- my_model$best_params

## Making predictions on the training set

# Gives us the predicted probablities for the label being 1
# Rounding them will give us the predicted label where >= 0.5 is 1 and < 0.5 is 0
train_preds <- predict(X_train, w)
train_conf_matrix <- table(Prediction = round(train_preds), Actual = y_train)
train_error_rate <- mean(round(train_preds) != y_train)


## Making predictions on the test set

test_preds <- predict(X_test, w)
test_conf_matrix <- table(Prediction = round(test_preds), Actual = y_test)

test_error_rate <- mean(round(test_preds) != y_test)
```

Confusion Matrix for Training Set:

```
##           Actual
## Prediction  0  1
##          0 92  6
##          1 11 87
```

Training Error Rate:

```
## [1] 0.08673469
```

Confusion Matrix for Test Set:

```
##            Actual
## Prediction  0  1
##          0 84  8
##          1  9 95
```

Test Error Rate:

```
## [1] 0.08673469
```