Assignment 1 - Logistic Regression

Name: Vikas Vijaykumar Bastewad

Roll Number: 20CS10073

In [ ]:
```
import pandas as pd
import numpy as np
```

In [ ]:
```
df = pd.read_excel('../../dataset/logistic-regression/Pumpkin_Seeds_Dataset.xlsx')
df.head()
```

Out[ ]:

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Convex_Area	Equiv_Diameter	Eccentricity	Solidity	Extent	Roundness	Aspect_Ration	Compactness	Class
0	56276	888.242	326.1485	220.2388	56831	267.6805	0.7376	0.9902	0.7453	0.8963	1.4809	0.8207	Çerçevelik
1	76631	1068.146	417.1932	234.2289	77280	312.3614	0.8275	0.9916	0.7151	0.8440	1.7811	0.7487	Çerçevelik
2	71623	1082.987	435.8328	211.0457	72663	301.9822	0.8749	0.9857	0.7400	0.7674	2.0651	0.6929	Çerçevelik
3	66458	992.051	381.5638	222.5322	67118	290.8899	0.8123	0.9902	0.7396	0.8486	1.7146	0.7624	Çerçevelik
4	66107	998.146	383.8883	220.4545	67117	290.1207	0.8187	0.9850	0.6752	0.8338	1.7413	0.7557	Çerçevelik

In [ ]:
```
# encoding class column as integers 0,1 for binary classification
for i in range(len(df)):
    if df['Class'][i] == 'Çerçevelik':
        df['Class'][i] = 0
    else:
        df['Class'][i] = 1
```

```
/tmp/ipykernel_5008/3205772911.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Class'][i] = 0
/tmp/ipykernel_5008/3205772911.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Class'][i] = 1
```

In [ ]:
```
# Shuffle the dataset
data = df.sample(frac=1).reset_index(drop=True)
```

```
data.head()
```

Out[ ]:

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Convex_Area	Equiv_Diameter	Eccentricity	Solidity	Extent	Roundness	Aspect_Ration	Compactness	Class
0	90267	1263.079	540.4486	214.8378	90961	339.0155	0.9176	0.9924	0.6996	0.7110	2.5156	0.6273	1
1	60291	1020.062	402.4420	193.8990	61478	277.0648	0.8763	0.9807	0.7556	0.7281	2.0755	0.6885	1
2	62922	974.831	374.2026	216.1344	63824	283.0455	0.8163	0.9859	0.7566	0.8321	1.7313	0.7564	0
3	70087	1004.981	389.6974	229.8159	70788	298.7265	0.8076	0.9901	0.6869	0.8720	1.6957	0.7666	0
4	81601	1095.816	418.4185	249.6238	82693	322.3315	0.8025	0.9868	0.7512	0.8539	1.6762	0.7704	0

In [ ]:
```
# Spliting the dataset into features and labels
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
```

In [ ]:
```
# Normalizing the features using (Min-Max Scaling)
X_normalized = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
```

In [ ]:
```
# Split dataset into training, validation, and test sets
train_size = int(0.5 * len(data))
val_size = int(0.3 * len(data))
X_train, y_train = X_normalized[:train_size], y[:train_size]
X_val, y_val = X_normalized[train_size:train_size+val_size], y[train_size:train_size+val_size]
X_test, y_test = X_normalized[train_size+val_size:], y[train_size+val_size:]
```

In [ ]:
```
# Implementing Logistic Regression functions
def sigmoid(z):                                            # Sigmoid function to map values between 0 and 1
    z = np.array(z, dtype=np.float128)
    return 1 / (1 + np.exp(-z))

def cost_function(theta, X, y):                            # Cost function for logistic regression
    m = len(y)
    h = sigmoid(X.dot(theta))
    J = (-1/m) * np.sum(y * np.log(h) + (1 - y) * np.log(1 - h))
    return J

def gradient_descent(X, y, theta, alpha, num_iterations):   # Gradient descent algorithm to minimize the cost function and find the optimal theta
    m = len(y)                                             # number of training examples
    losses = np.zeros(num_iterations)                      # losses for each iteration
    for i in range(num_iterations):                        # loop for each iteration
        h = sigmoid(X.dot(theta))                          # hypothesis function
        theta = theta - (alpha/m) * X.T.dot(h - y)         # gradient descent update
        losses[i] = cost_function(theta, X, y)             # cost function for each iteration
    return theta, losses
```

In [ ]:

```python
# Augmenting the data with a column of ones
X_train_1 = np.concatenate((np.ones((X_train.shape[0], 1)), X_train), axis=1) # adding a column of ones to X_train
X_val_1 = np.concatenate((np.ones((X_val.shape[0], 1)), X_val), axis=1)       # adding a column of ones to X_val
X_test_1 = np.concatenate((np.ones((X_test.shape[0], 1)), X_test), axis=1)    # adding a column of ones to X_test
```

In [ ]:

```python
# Initializing model parameters
num_features = X_train_1.shape[1]  # number of features in X
theta = np.zeros(num_features)     # model parameters initialized to 0

# Hyperparameters for training the model
learning_rate = 0.01
iterations = 1000

# Training the model
theta, _ = gradient_descent(X_train_1, y_train, theta, learning_rate, iterations)
```

In [ ]:

```python
# This function predicts the output of the model
def predict(theta, X):
    return (sigmoid(X.dot(theta)) >= 0.5).astype(int)
```

In [ ]:

```python
# Predicting labels for validation set
y_val_pred = predict(theta, X_val_1)

# Calculating confusion matrix components (TP, FP, FN) for validation set
TP = np.sum((y_val == 1) & (y_val_pred == 1))
FP = np.sum((y_val == 0) & (y_val_pred == 1))
FN = np.sum((y_val == 1) & (y_val_pred == 0))

# Calculating metrics (accuracy, precision, recall) for validation set
accuracy = np.mean(y_val == y_val_pred)
precision = TP / (TP + FP) if (TP + FP) > 0 else 0.0
recall = TP / (TP + FN) if (TP + FN) > 0 else 0.0

print("Validation Set Metrics:")
print("===========================================")
print("Mean Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("===========================================")
```

```
Validation Set Metrics:
===========================================
Mean Accuracy: 0.8493333333333334
Precision: 0.9057239057239057
Recall: 0.7598870056497176
===========================================
```

In [ ]:

```python
y_test_pred = predict(theta, X_test_1)

# Calculating confusion matrix components (TP, FP, FN) for test set predictions
TP = np.sum((y_test == 1) & (y_test_pred == 1))
FP = np.sum((y_test == 0) & (y_test_pred == 1))
FN = np.sum((y_test == 1) & (y_test_pred == 0))

# Calculating metrics for test set predictions (accuracy, precision, recall)
accuracy = np.mean(y_test == y_test_pred)
precision = TP / (TP + FP) if (TP + FP) > 0 else 0.0
recall = TP / (TP + FN) if (TP + FN) > 0 else 0.0

# Printing metrics for test set predictions
print("Test Set Metrics:")
print("=========================================")
print("Mean Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("=========================================")
```

```
Test Set Metrics:
=========================================
Mean Accuracy: 0.868
Precision: 0.9116279069767442
Recall: 0.8065843621399177
=========================================
```