code

August 17, 2023

0.1 Assignment 1

0.1.1 Name: Bannuru Rohit Kumar Reddy

0.1.2 Roll Number: 21CS30011

Libraries :

[150]:
```
# import all the necessary libraries here :

import pandas as pd

import numpy as np
from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, precision_score, recall_score
```

Loading the dataset and analysing the type of data

[151]:
```
df = pd.read_excel('../../dataset/logistic-regression/Pumpkin_Seeds_Dataset.
↪xlsx')
print(df.shape)
print(df.head(1))
```

```
(2500, 13)
    Area  Perimeter  Major_Axis_Length  Minor_Axis_Length  Convex_Area  \
0  56276    888.242           326.1485           220.2388        56831

   Equiv_Diameter  Eccentricity  Solidity  Extent  Roundness  Aspect_Ration  \
0        267.6805        0.7376    0.9902  0.7453     0.8963         1.4809

   Compactness       Class
0       0.8207  Çerçevelik
```

PreProcessing the Data:

[152]:
```
# PreProcessing the data :

# Encode the 'Class' column to numerical values
class_mapping = {'Çerçevelik': 0, 'Ürgüp Sivrisi': 1}
df['Class'] = df['Class'].map(class_mapping)
```

```
# Split the dataset into train, validation, and test sets
train_data, temp_data = train_test_split(df, test_size=0.5, random_state=42)
valid_data, test_data = train_test_split(temp_data, test_size=0.4,␣
 ↪random_state=42)

# Separate features and target variables
X_train = train_data.drop('Class', axis=1).values
y_train = train_data['Class'].values

X_valid = valid_data.drop('Class', axis=1).values
y_valid = valid_data['Class'].values

X_test = test_data.drop('Class', axis=1).values
y_test = test_data['Class'].values

# Normalize/Standardize the features
mean = np.mean(X_train, axis=0)
std = np.std(X_train, axis=0)
X_train = (X_train - mean) / std
X_valid = (X_valid - mean) / std
X_test = (X_test - mean) / std

# Add a column of 1's (bias term) to X_train, X_valid, and X_test using NumPy
bias_column_train = np.ones((X_train.shape[0], 1))


# Concatenate the bias column with the original X_train, X_valid, and X_test
X_train = np.hstack((bias_column_train, X_train))
X_valid = np.hstack((np.ones((X_valid.shape[0], 1)), X_valid))
X_test = np.hstack((np.ones((X_test.shape[0], 1)), X_test))

print(X_train)
print(y_train)
```

```
[[ 1.          0.60882926  0.6459801  … -0.30180065  0.01488933
  -0.14217799]
 [ 1.          0.32818862 -0.24722132 …  1.58513179 -1.47631136
   1.68881593]
 [ 1.          1.2310118   1.65276095 … -1.43895861  1.61791432
  -1.50887079]
 …
 [ 1.          1.01272767  0.49924928 …  0.97281596 -0.95309963
   0.98300334]
 [ 1.          0.07163201 -0.07652727 …  0.42476652 -0.42705291
   0.34458931]
 [ 1.         -0.105833   -0.10490407 …  0.06773106 -0.05661523
```

```
  -0.03920798]]
[1 0 1 … 0 0 0]
```

Logistic Regression :

[153]:
```python
# Defining the model :

def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def predict(X, weights):
    return sigmoid(np.dot(X,weights.T))

def logistic_regression(X, y, num_epochs, learning_rate):
    num_samples, num_features = X.shape
    weights = np.zeros(num_features)

    for epoch in range(num_epochs):
        y_pred = predict(X, weights)
        gradient = np.dot(X.T, (y_pred - y)) / num_samples
        weights -= learning_rate * gradient

    return weights
```

Predicting values for test data based :

[154]:
```python
# Training the logistic regression model
num_epochs = 1000
learning_rate = 0.001
weights = logistic_regression(X_train, y_train, num_epochs, learning_rate)

# Make predictions on the validation set
y_pred_valid = predict(X_valid, weights)
y_pred_valid_class = np.round(y_pred_valid)

# Make predictions on the test set
y_pred_test = predict(X_test, weights)
y_pred_test_class = np.round(y_pred_test)
```

Evaluating how well the model is performing on training and test data :

[155]:
```python
def evaluate_metrics(y_true, y_pred):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    return accuracy, precision, recall

# Evaluate on validation set
```

```
accuracy_valid, precision_valid, recall_valid = evaluate_metrics(y_valid,␣
 ↪y_pred_valid_class)

# Evaluate on test set
accuracy_test, precision_test, recall_test = evaluate_metrics(y_test,␣
 ↪y_pred_test_class)

print("Validation Set Metrics:")
print(f"Accuracy: {accuracy_valid:.5f}")
print(f"Precision: {precision_valid:.5f}")
print(f"Recall: {recall_valid:.5f}")

print("\nTest Set Metrics:")
print(f"Accuracy: {accuracy_test:.5f}")
print(f"Precision: {precision_test:.5f}")
print(f"Recall: {recall_test:.5f}")
```

```
Validation Set Metrics:
Accuracy: 0.84533
Precision: 0.85797
Recall: 0.81543

Test Set Metrics:
Accuracy: 0.85400
Precision: 0.83750
Recall: 0.85532
```

[ ]: