code

August 17, 2023

0.1 Assignment 1

0.1.1 Name: Anuguru Parthiv Reddy

0.1.2 Roll Number: 21CS10006

[2]:
```
# import all the necessary libraries here
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

[3]:
```
data = pd.read_csv('../../dataset/Pumpkin_Seeds_Dataset.csv')
data.isnull().sum()              #to check for null values
data['Class'].unique()           #removes unique values to see number of classes
data['Class'].value_counts()     #to see count of each class
le = LabelEncoder()
data["Class"] = le.fit_transform(data["Class"]) #used to assign numerical␣
 ↪values(0-bad,1-good) to the classes
```

[4]:
```
# Splitting the dataset into features (x_data) and target labels (y_data)
x_data = data.iloc[:, :-1]
y_data = data.iloc[:, -1]

# Standardizing the features using StandardScaler
scaler = StandardScaler()
scaled_x_data = scaler.fit_transform(x_data)
scaled_x_data = pd.DataFrame(scaled_x_data)
scaled_x_data.columns = x_data.columns

# Splitting the data into train, validation, and test sets
x_train, x_rem, y_train, y_rem = train_test_split(scaled_x_data, y_data,␣
 ↪train_size=0.5)  # 50% for training
x_val, x_test, y_val, y_test = train_test_split(x_rem, y_rem, train_size=0.6)  ␣
 ↪         # 60% of remaining data for validation

# Resetting and dropping the index column for consistency in data handling
```

```python
x_train = x_train.reset_index()
y_train = y_train.reset_index()
x_val = x_val.reset_index()
y_val = y_val.reset_index()
x_train = x_train.drop(['index'], axis=1)
y_train = y_train.drop(['index'], axis=1)
x_val = x_val.drop(['index'], axis=1)
y_val = y_val.drop(['index'], axis=1)
y_val = y_val.iloc[:, 0]  # Extracting the Series from DataFrame

# Definition of the sigmoid function for logistic regression
def sigmoid(w, x, b):
    ans = 0
    for i in range(len(x)):
        ans = ans + (w[i] * x[i])
    return 1 / (1 + np.exp(-ans))

# Definition of the loss function (binary cross-entropy) for logistic regression
def loss(x, y, w, b):
    f = sigmoid(w, x, b)
    ans1 = y * (np.log(f))
    ans2 = (1 - y) * (np.log(1 - f))
    return -(ans1 + ans2)

# Gradient calculation for weights (w) with respect to a specific feature (j)
def grad_w(x_data, y_data, j, w, b):
    ans = 0
    for i in range(len(x_data)):
        arr = np.array(x_data.loc[i])
        ans = ans + (sigmoid(w, arr, b) - y_data.loc[i]) * arr[j]
    return ans / len(x_data)

# Gradient calculation for bias (b)
def grad_b(x_data, y_data, w, b):
    ans = 0
    for i in range(len(x_data)):
        arr = np.array(x_data.loc[i])
        ans = ans + (sigmoid(w, arr, b) - y_data.loc[i])
    return ans / len(x_data)

# Gradient Descent optimization to update weights (w) and bias (b)
def grad_descent(x_data, y_data, w, b, eps, alp):
    k = 100
    while k:
        for j in range(len(x_data.loc[0])):
            w[j] = w[j] - alp * grad_w(x_data, y_data, j, w, b)
        b = b - alp * grad_b(x_data, y_data, w, b)
```

```
        k = k - 1
    return b

# Initializing weights (w), bias (b), learning rate (alp), and convergence␣
 ↪threshold (eps)
w = np.zeros(12)
b = 0
alp = 1
eps = 0.5

# Performing gradient descent to update bias (b)
b = grad_descent(x_train, y_train, w, b, eps, alp)
```

[5]:

```
#accuracy precision and recall for Test set
tp=0
tn=0
fp=0
fn=0
for i in range(len(x_test)):
    arr=np.array(x_test.iloc[i])
    ans=sigmoid(w,arr,b)
    if(ans>=0.5):
        if(y_test.iloc[i]==1):
            tp=tp+1
        else:
            fp=fp+1
    else:
        if(y_test.iloc[i]==1):
            fn=fn+1
        else:
            tn=tn+1

print("Test data")
#for class 1
accuracy=tp/(tp+fn)
precision=tp/(tp+fp)
recall=tp/(tp+fn)
print(f"class 1 ==> accuracy: {accuracy}, precision: {precision}, recall:␣
 ↪{recall}")

#for class 0
accuracy=tn/(tn+fp)
precision=tn/(tn+fp)
recall=tn/(tn+fn)
print(f"class 0 ==> accuracy: {accuracy} precision: {precision} recall:␣
 ↪{recall}")
```

```
Test data
class 1 ==> accuracy: 0.84375, precision: 0.9264705882352942, recall: 0.84375
class 0 ==> accuracy: 0.9456521739130435 precision: 0.9456521739130435 recall:
0.8817567567567568
```

[6]:

```python
print("Validation data")
tp=0
tn=0
fp=0
fn=0
for i in range(len(x_val)):
    arr=np.array(x_val.iloc[i])
    ans=sigmoid(w,arr,b)
    if(ans>=0.5):
        if(y_val.iloc[i]==1):
            tp=tp+1
        else:
            fp=fp+1
    else:
        if(y_val.iloc[i]==1):
            fn=fn+1
        else:
            tn=tn+1

#for class 1
accuracy=tp/(tp+fn)
precision=tp/(tp+fp)
recall=tp/(tp+fn)
print(f"class 1 ==> accuracy: {accuracy}, precision: {precision}, recall:␣
 ↪{recall}")

#for class 0
accuracy=tn/(tn+fp)
precision=tn/(tn+fp)
recall=tn/(tn+fn)
print(f"class 0 ==> accuracy: {accuracy} precision: {precision} recall:␣
 ↪{recall}")
```

```
Validation data
class 1 ==> accuracy: 0.8651685393258427, precision: 0.8627450980392157, recall:
0.8651685393258427
class 0 ==> accuracy: 0.8756345177664975 precision: 0.8756345177664975 recall:
0.8778625954198473
```