logistic_regression

July 2, 2024

1 Logistic Regression

1.0.1 Imports

[11]:
```
import numpy as np
import matplotlib.pyplot as plt
```

1.0.2 Implementation

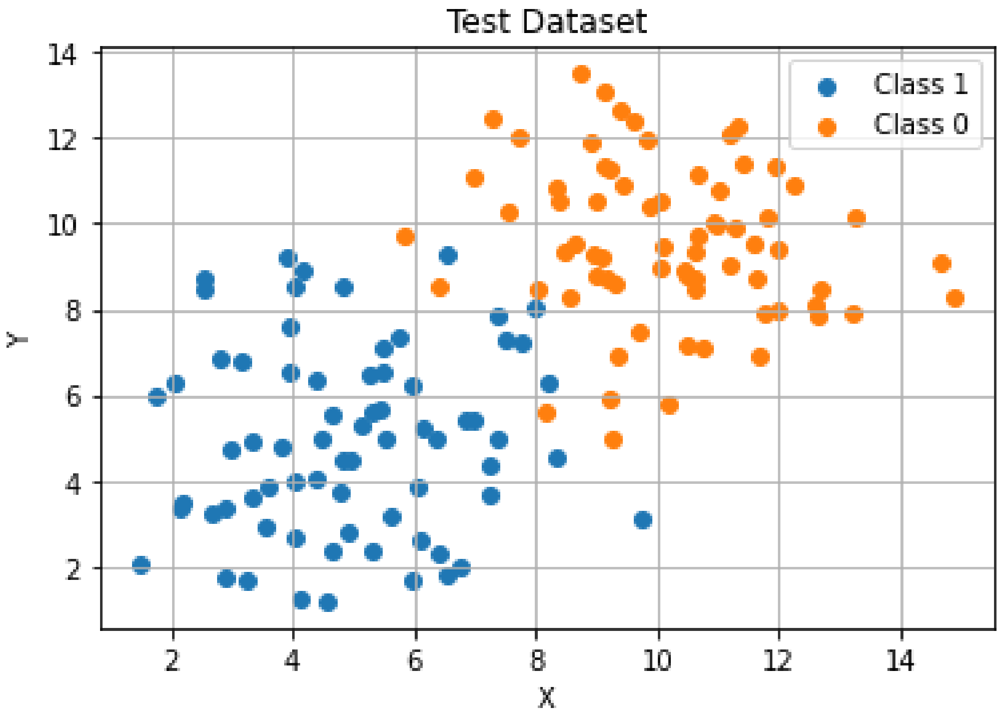
[6]:
```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

[148]:
```
def train(x, y, theta, learning_rate, n_iterations):
    # Train with gradient descent while keeping track of the cost
    for i in range(n_iterations):
        preds = sigmoid(np.dot(x, theta))
        cost = np.mean(-y * np.log(preds) - (1 - y) * np.log(1 - preds))
        gradient = np.dot(x.T, (preds - y)) / len(y)
        theta -= learning_rate * gradient
        if i % 500 == 0:
            print(f"Iteration {i}'s cost: {cost}")
    return theta
```

1.0.3 Generate Test Dataset

[149]:
```
std_dev = 2

x_train = np.array([np.random.normal(5, std_dev, 2) for x in range(70)] + [np.
 ↪random.normal(10, std_dev, 2) for x in range(70)])
x_train = np.column_stack((np.ones(x_train.shape[0]), x_train))
y_train = np.append(np.ones(70), np.zeros(70))

x_test = np.array([np.random.normal(5, std_dev, 2) for x in range(30)] + [np.
 ↪random.normal(10, std_dev, 2) for x in range(30)])
x_test = np.column_stack((np.ones(x_test.shape[0]), x_test))
y_test = np.append(np.ones(30), np.zeros(30))
```

[150]:
```
fig = plt.figure()
ax = fig.add_subplot(111)

ax.scatter(x_train[:int(x_train.shape[0]/2), 1], x_train[:int(x_train.shape[0]/
↪2), 2], label = "Class 1")
ax.scatter(x_train[int(x_train.shape[0]/2):, 1], x_train[int(x_train.shape[0]/
↪2):, 2], label = "Class 0")
ax.legend()
ax.grid()
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_title("Test Dataset")

plt.show()
```



1.0.4 Test

[151]:
```
theta = np.zeros(x_train.shape[1])
learning_rate = 0.2
n_iterations = 10000

theta = train(x_train, y_train, theta, learning_rate, n_iterations)
```

```
Iteration 0's cost: 0.6931471805599454
Iteration 500's cost: 0.1630459127801968
Iteration 1000's cost: 0.1265021403713176
Iteration 1500's cost: 0.11450031701039938
Iteration 2000's cost: 0.10706036875232687
Iteration 2500's cost: 0.10196036353302154
Iteration 3000's cost: 0.098231931931572
Iteration 3500's cost: 0.09538129911394899
Iteration 4000's cost: 0.0931286482952253
Iteration 4500's cost: 0.09130284638489632
Iteration 5000's cost: 0.08979300301470361
Iteration 5500's cost: 0.08852392785551133
Iteration 6000's cost: 0.0874427362464049
Iteration 6500's cost: 0.08651109257222145
Iteration 7000's cost: 0.08570049539117937
Iteration 7500's cost: 0.08498929335558414
Iteration 8000's cost: 0.0843607308327495
Iteration 8500's cost: 0.08380162977086272
Iteration 9000's cost: 0.08330147773130342
Iteration 9500's cost: 0.08285178267945133
```

[152]:

```python
probs = sigmoid(np.dot(x_test, theta))
preds = (probs >= 0.5).astype(int)
```

[ ]:

```python
accuracy = sum(1 for i in range(len(preds)) if preds[i] == y_test[i]) /␣
 ↪len(preds)
print(f"Accuracy: {accuracy * 100}%")
```

```
Accuracy: 96.66666666666667%
```