Logistic1

February 7, 2022

[1]:
```python
### logistic regression in python

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[2]:
```python
def sigmoid(x):
    return 1/(1+np.e**-x)
```

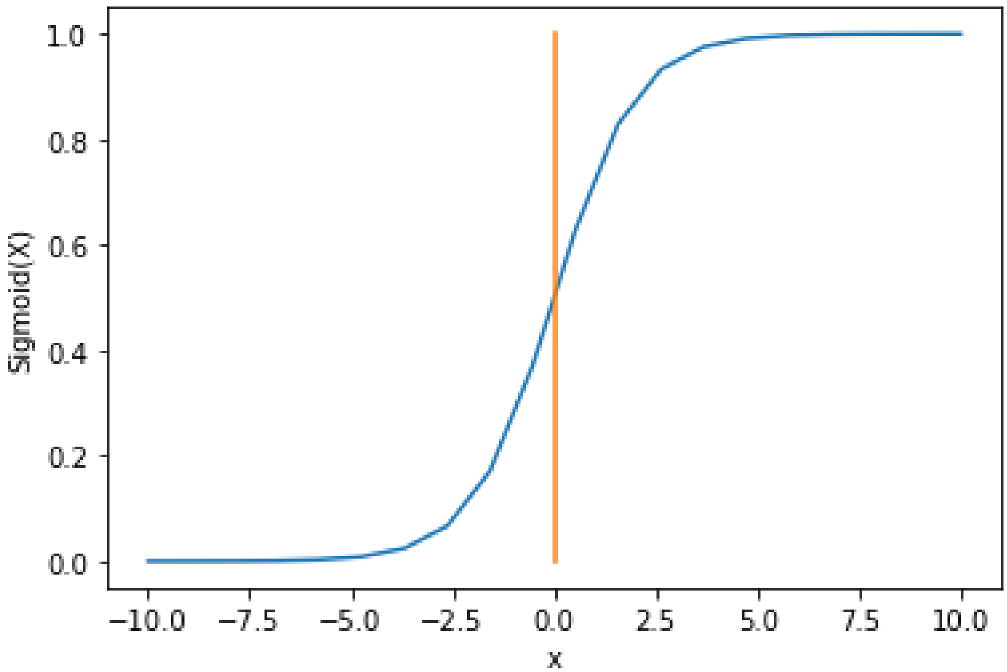
[5]:
```python
## Sigmoid function
# Import matplotlib, numpy and math
import matplotlib.pyplot as plt
import numpy as np
import math

x = np.linspace(-10, 10, 20)

z = sigmoid(x)

plt.plot(x, z)
plt.plot(np.zeros(6), [0,0.2,0.4,0.6,0.8,1]) ## Add vertical line at zero
plt.xlabel("x")
plt.ylabel("Sigmoid(X)")

plt.show()
```



[6]:
```
a = np.array([[2,3],[4,1],[5,4],[8,8],[9,1],[2,6]])

b = np.array([0,0,1,1,0,1])
```

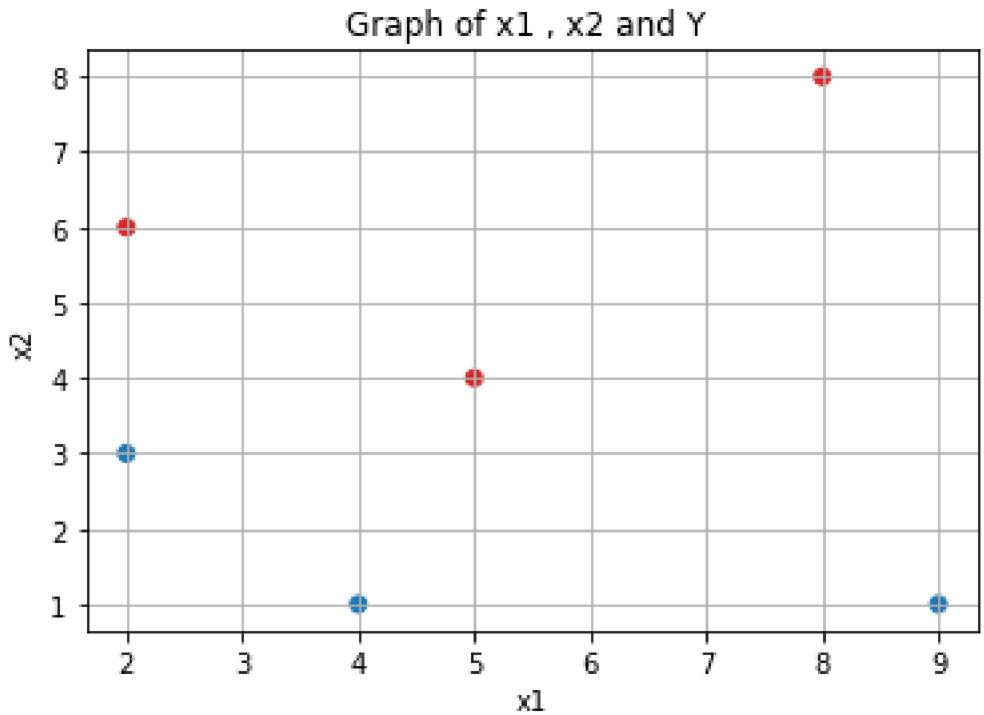
[7]:
```
df = pd.DataFrame(a, columns = ["x1","x2"])
df["y"] = b

df
```

[7]:

	x1	x2	y
0	2	3	0
1	4	1	0
2	5	4	1
3	8	8	1
4	9	1	0
5	2	6	1

[8]:
```
## All line together on the same plot for easy comparison
colors = {0:'tab:blue', 1:'tab:red'}
plt.scatter(a[:,0],a[:,1], c=df["y"].map(colors))
plt.title('Graph of x1 , x2 and Y ')
plt.xlabel('x1')
plt.ylabel('x2')
plt.grid()
plt.show()
```

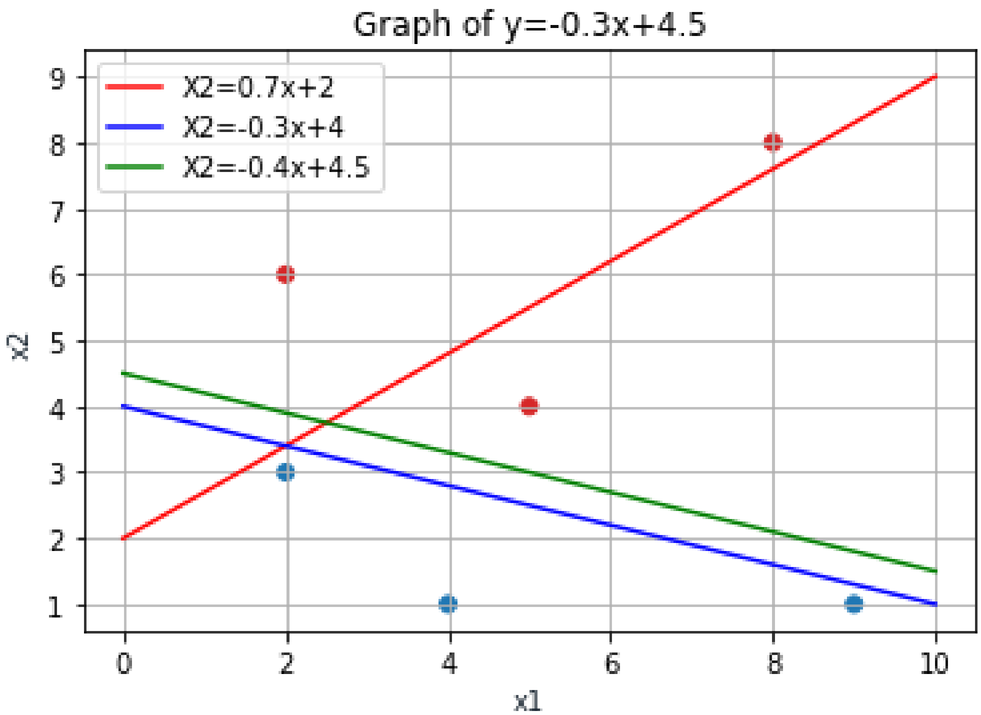


[10]:
```
x = np.linspace(0,10,100)
plt.scatter(a[:,0],a[:,1], c=df["y"].map(colors))

y = 0.7*x+2
plt.plot(x, y, '-r', label='X2=0.7x+2')

y = -0.3*x+4
plt.plot(x, y, '-b', label='X2=-0.3x+4')

y = -0.3*x+4.5
plt.plot(x, y, '-g', label='X2=-0.4x+4.5')
plt.title('Graph of y=-0.3x+4.5')
plt.xlabel('x1', color='#1C2833')
plt.ylabel('x2', color='#1C2833')
plt.legend(loc='upper left')
plt.grid()
plt.show()
```



[11]:
```
def sigmoid(x):
    return 1/(1+np.e**-x)

def Negloglikelihood(y, yhat):
    return -(np.log(yhat) * y + np.log(1 - yhat) * (1 - y))
```

[12]:
```
# Compute the point on X2
df['x2_green'] = -0.3*df["x1"]+4.5
df["x2_blue"] = -0.3*df["x1"]+4
df["x2_red"] = 0.7*df["x1"]+2

#Find the distance between each line to the correspoinding point
df["dist_green"] = df["x2"] - df['x2_green']
df["dist_blue"] = df["x2"] - df['x2_blue']
df["dist_red"] =  df["x2"] - df['x2_red']

# Convert the distance to probability using Sigmoid
df["prob_green"] = sigmoid(df["dist_green"])
df["prob_blue"] = sigmoid(df["dist_blue"])
df["prob_red"] = sigmoid(df["dist_red"])

# compute the negative log likelihood for each line
df["loglike_green"] = Negloglikelihood(df['y'],df['prob_green'])
```

```python
df["loglike_blue"] = Negloglikelihood(df['y'],df['prob_blue'])
df["loglike_red"] = Negloglikelihood(df['y'],df['prob_red'])

# Sum of negative log likelihood for each line.
print("Negative log Likelihood of Green:",sum(df['loglike_green']))
print("Negative log Likelihood of blue:",sum(df['loglike_blue']))
print("Negative log Likelihood of red:",sum(df['loglike_red']))
```

```
Likelihood of Green: 1.2393169153543424
Likelihood of blue: 1.4950662557590215
Likelihood of red: 2.821888001906791
```

[13]:
```python
def sigmoid(z):
    return 1.0/(1.0 + np.exp(-1.0*z))



def hypothesis(X, theta):
    """
    X - np array (m,n)
    theta - np arrary (n, 1)
    """
    return sigmoid(np.dot(X, theta))
```

[14]:
```python
def error(X, y, theta):
    """
    params:
        X - np array (m,n)
        y - np array (m,1)
        theta - np arrary (n,1)

    return :
        scalar value = loss value
    """
    hypo = hypothesis(X, theta)
    err = np.mean((y*np.log(hypo) + (1-y)*np.log(1- hypo)))

    return -err
```

[15]:
```python
def gradient(X, y, theta):
    """
    X - (m,n)
    y - (m,1)
    theta - (n,1)

    return - (n, 1)
    """
    hypo = hypothesis(X, theta)
    grad = (np.dot(X.T, (hypo - y)))
```

```
    return grad/X.shape[0]
```

[49]:
```
X.shape
```

[49]: (500, 2)

[46]:
```
def gradient_descent(X, y, lr = 0.5, max_iter = 30):
    theta = np.zeros((X.shape[1], 1))
    error_list = []

    for _ in range(max_iter):
        e = error(X, y, theta)
        error_list.append(e)

        grad = gradient(X, y, theta)

        #Update Rule
        theta = theta - lr*grad


    return (theta, error_list)
```

[27]:
```
theta = np.zeros((X.shape[1], 1))
theta
```

[27]:
```
array([[0.],
       [0.]])
```

[ ]:
```

```

[30]:
```
##dataset creation
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=500,
                           n_features=2,
                           n_redundant=0,
                           n_clusters_per_class=1,
                           random_state=5)

# X = dataset[:, :2]
# y = dataset[:, -1]
```

[31]:
```
X.shape
```

[31]: (500, 2)

[32]:
```
y.shape
```

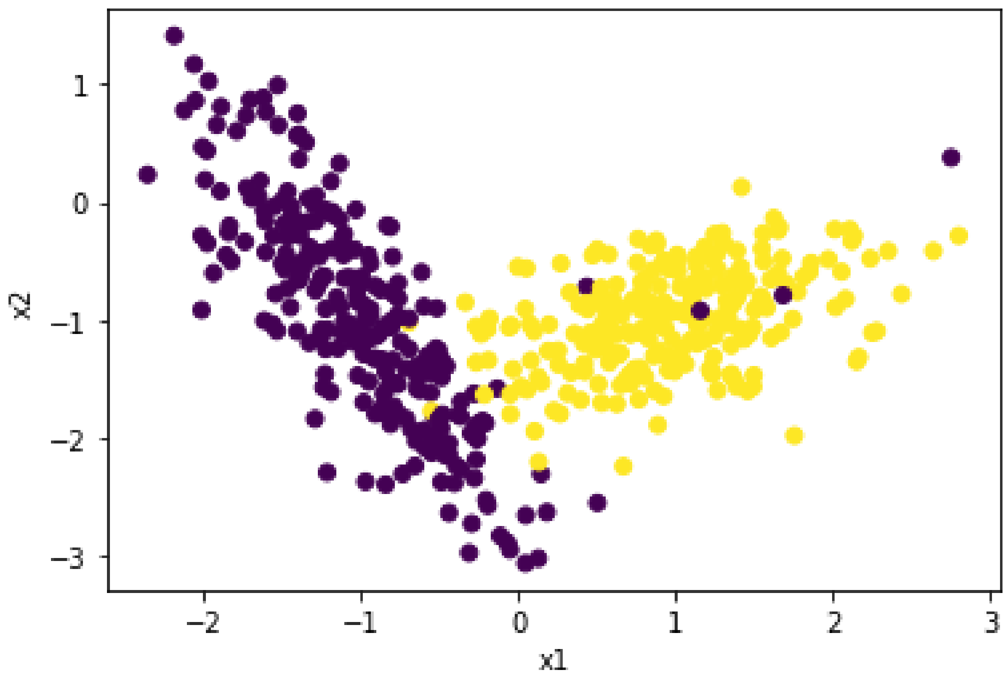
[32]: (500,)

[61]:
```
##reshaping y
y = y.reshape(-1, 1)
y.shape
```

[61]: (500, 1)

[63]:
```
plt.scatter(X[:, 0], X[:, 1], c= y)

plt.xlabel("x1")
plt.ylabel("x2")
plt.show()
```



[64]:
```
ones= np.ones((500,1))
X_ = np.hstack((ones, X))
X_[:5]
```

[64]:
```
array([[ 1.        ,  1.22167239, -0.4757541 ],
       [ 1.        , -0.2292072 , -1.85663378],
       [ 1.        , -1.34913896,  0.50458721],
       [ 1.        ,  0.31402206, -1.62029248],
       [ 1.        ,  1.13807877, -0.99148158]])
```

[65]:
```
from sklearn.model_selection import train_test_split

# reserving 20% of the data for testing purposes
X_train, X_test, y_train, y_test = train_test_split(X_, y, test_size=0.2,␣
 ↪random_state=42)
```

[66]:
```
X_train.shape
```

[66]: (400, 3)

[67]:
```
opt_theta, error_list = gradient_descent(X_train, y_train)
```

[69]:
```
opt_theta ## This one was giving mutiple values for each coefficient,
#fixed that peice of code. its the shape of y that is cuasing the problem
```

[69]:
```
array([[0.03959218],
       [0.76836401],
       [0.03801677]])
```

[40]:
```
from sklearn.linear_model import LogisticRegression
```

[41]:
```
logistic = LogisticRegression()
```

[43]:
```
logistic.fit(X_train, y_train)
```

[43]: LogisticRegression()

[44]:
```
logistic.intercept_
```

[44]: array([1.46952407])

[45]:
```
logistic.coef_
```

[45]: array([[-3.17062293e-06, 3.89591876e+00, 8.50972735e-01]])

[ ]:
```
## MOdel performance - test dataset
```

[50]:
```
y_pred = logistic.predict(X_test)
```

[51]:
```
## Accuracy ?
from sklearn.metrics import accuracy_score, confusion_matrix
```

[52]:
```
confusion_matrix(y_test, y_pred)
```

[52]:
```
array([[47,  3],
       [ 2, 48]], dtype=int64)
```

[53]:
```
accuracy_score(y_test,y_pred)
```

[53]: 0.95

[54]:
```
from sklearn.metrics import precision_score, recall_score
```

[55]:
```
precision_score(y_test, y_pred)
```

[55]: 0.9411764705882353

[56]:
```
recall_score(y_test, y_pred)
```

[56]: 0.96

[57]:
```
y_pred
```

[57]:
```
array([1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1,
       1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0])
```

[58]:
```
prob = logistic.predict_proba(X_test)
```

[60]:
```
prob[1]
```

[60]:
```
array([0.01821833, 0.98178167])
```

[ ]:
```

```