logistic-regression-from-scratch

December 13, 2023

[1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import plotly as py
import plotly.graph_objs as go
import time

init_notebook_mode(connected=True)
```

[2]:

```
def sigmoid(X, weight):
    z = np.dot(X, weight)
    return 1 / (1 + np.exp(-z))
```

[3]:

```
def loss(h, y):
    return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()
```

[4]:

```
def gradient_descent(X, h, y):
    return np.dot(X.T, (h - y)) / y.shape[0]
def update_weight_loss(weight, learning_rate, gradient):
    return weight - learning_rate * gradient
```

[5]:

```
def log_likelihood(x, y, weights):
    z = np.dot(x, weights)
    ll = np.sum( y*z - np.log(1 + np.exp(z)) )
    return ll
```

[6]:

```
def gradient_ascent(X, h, y):
    return np.dot(X.T, y - h)
def update_weight_mle(weight, learning_rate, gradient):
    return weight + learning_rate * gradient
```

[7]:

```
data = pd.read_csv(r'C:\Users\user\Desktop\ANJALI RAJ\PERCEPTRON␣
 ↪XOR\WA_Fn-UseC_-Telco-Customer-Churn.csv')
print("Dataset size")
print("Rows {} Columns {}".format(data.shape[0], data.shape[1]))
```

```
Dataset size
Rows 7043 Columns 21
```

[8]:

```
print("Columns and data types")
pd.DataFrame(data.dtypes).rename(columns = {0:'dtype'})
```

```
Columns and data types
```

[8]:

```
                    dtype
customerID         object
gender             object
SeniorCitizen       int64
Partner            object
Dependents         object
tenure              int64
PhoneService       object
MultipleLines      object
InternetService    object
OnlineSecurity     object
OnlineBackup       object
DeviceProtection   object
TechSupport        object
StreamingTV        object
StreamingMovies    object
Contract           object
PaperlessBilling   object
PaymentMethod      object
MonthlyCharges    float64
TotalCharges       object
Churn              object
```

[9]:

```
df = data.copy()
```

[10]:

```
df['class'] = df['Churn'].apply(lambda x : 1 if x == "Yes" else 0)
# features will be saved as X and our target will be saved as y
X = df[['tenure','MonthlyCharges']].copy()
X2 = df[['tenure','MonthlyCharges']].copy()
y = df['class'].copy()
```

[ ]:

```
start_time = time.time()

num_iter = 100000

intercept = np.ones((X.shape[0], 1))
X = np.concatenate((intercept, X), axis=1)
theta = np.zeros(X.shape[1])
```

```
for i in range(num_iter):
    h = sigmoid(X, theta)
    gradient = gradient_descent(X, h, y)
    theta = update_weight_loss(theta, 0.1, gradient)

print("Training time (Log Reg using Gradient descent):" + str(time.time() -␣
 ↪start_time) + " seconds")
print("Learning rate: {}\nIteration: {}".format(0.1, num_iter))
```

```
[ ]: result = sigmoid(X, theta)
```

```
[13]: f = pd.DataFrame(np.around(result, decimals=6)).join(y)
      f['pred'] = f[0].apply(lambda x : 0 if x < 0.5 else 1)
      print("Accuracy (Loss minimization):")
      f.loc[f['pred']==f['class']].shape[0] / f.shape[0] * 100
```

```
      Accuracy (Loss minimization):

[13]: 78.36149368166974
```

```
[ ]: start_time = time.time()
     num_iter = 100000

     intercept2 = np.ones((X2.shape[0], 1))
     X2 = np.concatenate((intercept2, X2), axis=1)
     theta2 = np.zeros(X2.shape[1])

     for i in range(num_iter):
         h2 = sigmoid(X2, theta2)
         gradient2 = gradient_ascent(X2, h2, y) #np.dot(X.T, (h - y)) / y.size
         theta2 = update_weight_mle(theta2, 0.1, gradient2)

     print("Training time (Log Reg using MLE):" + str(time.time() - start_time) +␣
      ↪"seconds")
     print("Learning rate: {}\nIteration: {}".format(0.1, num_iter))
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_5840\2745927695.py:3: RuntimeWarning:

overflow encountered in exp
```

```
[ ]: result2 = sigmoid(X2, theta2)
```

```
[ ]: print("Accuracy (Maximum Likelihood Estimation):")
     f2 = pd.DataFrame(result2).join(y)
     f2.loc[f2[0]==f2['class']].shape[0] / f2.shape[0] * 100
```

[ ]:

```
```