# Logistic Regression

May 14, 2019

Logistic Regression

Suppose that you are the administrator of a university department and you want to determine each applicant's chance of admission based on their results on two exams. You have historical data from previous applicants that you can use as a training set for logistic regression. For each training example, you have the applicant's scores on two exams and the admissions decision. Your task is to build a classification model that estimates an applicant's probability of admission based the scores from those two exams.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from scipy.optimize import minimize
from sklearn.preprocessing import PolynomialFeatures
```

[2]:
```
def sigmoid(Theta,X):
    return (1/(1+np.exp(-X@Theta)))
```

[3]:
```
def costFunction(Theta,X,y):
    m,n = X.shape
    Theta = np.matrix(Theta).reshape(n,1)
    H = sigmoid(Theta,X)
    return ((-1/m)*(y.T@np.log(H) + (1-y).T@np.log(1-H))).item()
```

[4]:
```
def grad(Theta,X,y):
    m,n = X.shape
    Theta = np.matrix(Theta).reshape(n,1)
    grad =(1/m)*(X.T @ (sigmoid(Theta,X)-y))
    grad = np.squeeze(np.asarray(grad))
    return grad
```

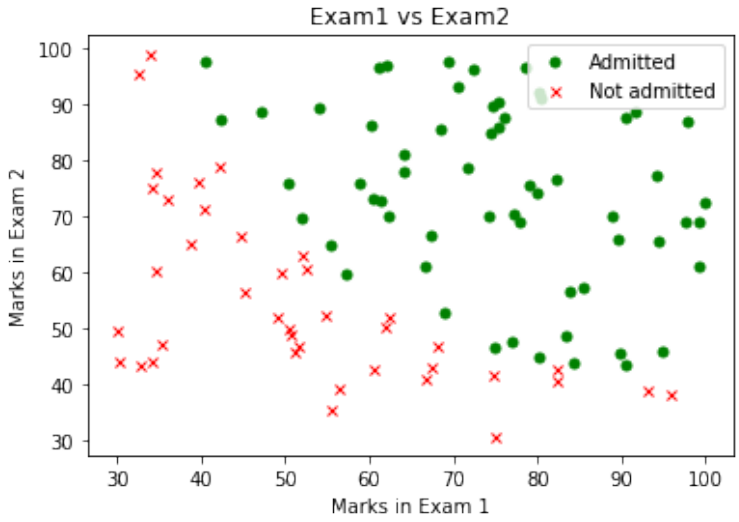
[5]:
```
def predict(Theta,X):
    sigmoidVals = sigmoid(Theta,X)
    return (sigmoidVals>=0.5).astype(int)
```

[6]:
```
data = np.loadtxt("Data/ex2data1.txt",delimiter=",")
X = data[:,:-1]
y = data[:,-1:]
```

[7]:
```
positive = (y==1)
negative = (y==0)
```

[8]:
```
fig, ax = plt.subplots()
ax.set(xlabel="Marks in Exam 1",ylabel="Marks in Exam 2",title="Exam1 vs Exam2")
ax.plot(X[positive[:,0],0],X[positive[:,0],1],c='g', marker = 'o', ls = '',␣
 ↪markersize = 5)
ax.plot(X[negative[:,0],0],X[negative[:,0],1],c='r', marker = 'x', ls = '',␣
 ↪markersize = 5)
ax.legend(["Admitted","Not admitted"],loc=1)
```

[8]:
```
<matplotlib.legend.Legend at 0x244a5956c18>
```




Regression_files/LogisticRegression_91.bb

[9]:
```
X = np.matrix(X)
y = np.matrix(y)
m,n = X.shape
X_new = np.hstack((np.ones((m,1)),X))
n = n+1
```

[10]:
```
initialTheta = np.zeros((n,1))
print("Cost when theta =␣
 ↪\n",initialTheta,"\nis",costFunction(initialTheta,X_new,y))
print("Gradient when theta =␣
 ↪\n",initialTheta,"\nis\n",grad(initialTheta,X_new,y))
```

```
Cost when theta =
 [[0.]
 [0.]
 [0.]]
is 0.6931471805599453
Gradient when theta =
 [[0.]
 [0.]
```

```
 [0.]]
is
 [ -0.1        -12.00921659 -11.26284221]
```

[11]:
```
result = minimize(fun = costFunction, x0 = initialTheta, args=(X_new,y), jac =␣
 ↪grad)
finalTheta = np.matrix(result.x).T
print("Optimized parameters after running gradient Descent are: \n",finalTheta)
```

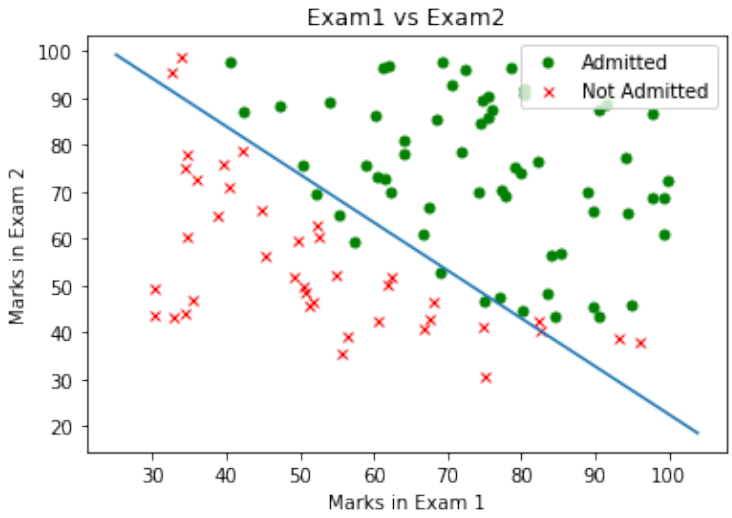
```
Optimized parameters after running gradient Descent are:
 [[-25.16133284]
 [  0.2062317 ]
 [  0.2014716 ]]

c:\users\devak\appdata\local\programs\python\python37\lib\site-
packages\ipykernel_launcher.py:5: RuntimeWarning: divide by zero encountered in
log
  """
c:\users\devak\appdata\local\programs\python\python37\lib\site-
packages\ipykernel_launcher.py:5: RuntimeWarning: invalid value encountered in
matmul
  """
```

[12]:
```
print("Cost after running gradient Descent is:␣
 ↪",costFunction(finalTheta,X_new,y))
```

```
Cost after running gradient Descent is:  0.20349770158944375
```

[13]:
```
fig, ax = plt.subplots()
ax.set(xlabel="Marks in Exam 1",ylabel="Marks in Exam 2",title="Exam1 vs Exam2")
ax.plot(X[positive[:,0],0],X[positive[:,0],1],c='g', marker = 'o', ls = '',␣
 ↪markersize = 5)
ax.plot(X[negative[:,0],0],X[negative[:,0],1],c='r', marker = 'x', ls = '',␣
 ↪markersize = 5)
x_values = [np.min(X_new[:, 1] - 5), np.max(X_new[:, 2] + 5)]
y_values = - (finalTheta[0] + finalTheta[1].item()*np.array(x_values)) /␣
 ↪finalTheta[2]
y_values = np.squeeze(np.asarray(y_values))
ax.plot(x_values,y_values)
ax.legend(["Admitted", "Not Admitted"],loc=1)
```

[13]:
```
<matplotlib.legend.Legend at 0x244a5a11780>
```




Regression_files/LogisticRegression_141.bb

[14]:
```
predictedValues = predict(finalTheta,X_new)
print("Accuracy: ",np.mean((predictedValues == y).astype(int))*100)
```

```
Accuracy:  89.0
```

Logistic Regression using Regularization

Suppose you are the product manager of the factory and you have the test results for some microchips on two different tests. From these two tests, you would like to determine whether the microchips should be accepted or rejected. To help you make the decision, you have a dataset of test results on past microchips, from which you can build a logistic regression model.

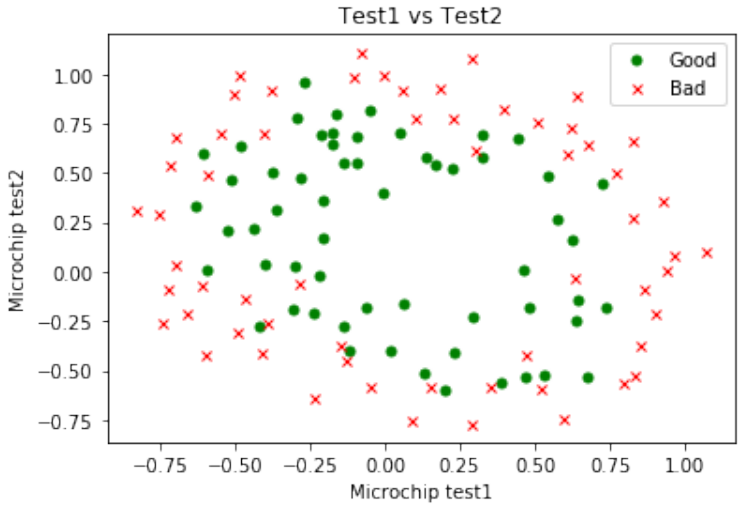
[15]:
```
def costFunctionReg(Theta,X,y,lamda):
    m,n = X.shape
    reg = 0
    Theta = np.matrix(Theta).reshape(n,1)
    H = sigmoid(Theta,X)
    J = ((-1/m)*(y.T@np.log(H) + (1-y).T@np.log(1-H)))
    reg = (0.5*lamda/m)*(Theta[1:,:].T @ Theta[1:,:])
    J = J + reg
    return J.item()
```

[16]:
```
def gradReg(Theta,X,y,lamda):
    m,n = X.shape
    Theta = np.matrix(Theta).reshape(n,1)
    grad =(1/m)*(X.T @ (sigmoid(Theta,X)-y))
    grad[1:,:] = grad[1:,:] + (lamda/m)*Theta[1:,:]
    grad = np.squeeze(np.asarray(grad))
    return grad
```

[17]:
```
data = np.loadtxt("Data/ex2data2.txt", delimiter=',')
X = data[:,:-1]
y = data[:,-1:]
```

[18]:
```
positive = (y==1)
negative = (y==0)
fig, ax = plt.subplots()
ax.set(xlabel="Microchip test1",ylabel="Microchip test2",title="Test1 vs Test2")
ax.plot(X[positive[:,0],0],X[positive[:,0],1],c='g', marker = 'o', ls = '',␣
 ↪markersize = 5)
ax.plot(X[negative[:,0],0],X[negative[:,0],1],c='r', marker = 'x', ls = '',␣
 ↪markersize = 5)
ax.legend(["Good","Bad"],loc=1)
```

[18]:
```
<matplotlib.legend.Legend at 0x244a5aae518>
```




Regression$_f$*iles*/*LogisticRegression*$_2$1$_1$.*bb*

[19]:
```
X = np.matrix(X)
y = np.matrix(y)
poly = PolynomialFeatures(6)
X_new = poly.fit_transform(X)
m,n = X_new.shape
```

[20]:
```
test_theta = np.zeros((n,1))
print("Cost when theta = Zeros\nis",costFunctionReg(test_theta,X_new,y,10))
print("Grad when theta = Zeros\nis",gradReg(test_theta,X_new,y,10))
```

```
Cost when theta = Zeros
is 0.6931471805599454
Grad when theta = Zeros
is [8.47457627e-03 1.87880932e-02 7.77711864e-05 5.03446395e-02
 1.15013308e-02 3.76648474e-02 1.83559872e-02 7.32393391e-03
 8.19244468e-03 2.34764889e-02 3.93486234e-02 2.23923907e-03
 1.28600503e-02 3.09593720e-03 3.93028171e-02 1.99707467e-02
 4.32983232e-03 3.38643902e-03 5.83822078e-03 4.47629067e-03
```

```
 3.10079849e-02 3.10312442e-02 1.09740238e-03 6.31570797e-03
 4.08503006e-04 7.26504316e-03 1.37646175e-03 3.87936363e-02]
```

[21]:
```
test_theta = np.ones((n,1))
print("Cost when theta = \nOnes\nis",costFunctionReg(test_theta,X_new,y,10))
print("Grad when theta = \nOnes\nis",gradReg(test_theta,X_new,y,10))
```

```
Cost when theta =
Ones
is 3.1645093316150095
Grad when theta =
Ones
is [0.34604507 0.16135192 0.19479576 0.22686278 0.09218568 0.24438558
 0.14339212 0.10844171 0.10231439 0.18346846 0.17353003 0.08725552
 0.11822776 0.0858433  0.19994895 0.13522653 0.09497527 0.09356441
 0.09979784 0.09140157 0.17485242 0.14955442 0.08678566 0.09897686
 0.08531951 0.10190666 0.08450198 0.18228323]
```

[22]:
```
initialTheta = np.zeros((n,1))
lamda = 1
result = minimize(fun = costFunctionReg, x0 = initialTheta,␣
 ↪args=(X_new,y,lamda), jac = gradReg)
finalTheta = np.matrix(result.x).T
print("Optimized parameters after running gradient Descent are: \n",finalTheta)
```

```
Optimized parameters after running gradient Descent are:
 [[ 1.27268739]
 [ 0.62557016]
 [ 1.1809665 ]
 [-2.01919822]
 [-0.91761468]
 [-1.43194199]
 [ 0.12375921]
 [-0.36513086]
 [-0.35703388]
 [-0.17485805]
 [-1.45843772]
 [-0.05129676]
 [-0.61603963]
 [-0.2746414 ]
 [-1.19282569]
 [-0.24270336]
 [-0.20570022]
 [-0.04499768]
 [-0.27782709]
 [-0.29525851]
 [-0.45613294]
```

```
 [-1.04377851]
 [ 0.02762813]
 [-0.29265642]
 [ 0.01543393]
 [-0.32759318]
 [-0.14389199]
 [-0.92460119]]
```

[23]:
```
print("Cost when theta : Final␣
 ↪Theta\nis",costFunctionReg(finalTheta,X_new,y,10))
```

```
Cost when theta : Final Theta
is 1.1279022737692763
```

[24]:
```
predictedValues = predict(finalTheta,X_new)
print("Accuracy: ",np.mean((predictedValues == y).astype(int))*100)
```

```
Accuracy:  83.05084745762711
```