

MA 506 Probability and Statistical Inference

Lecture 23: Logistic Regression 3

```
In [2]: import numpy as np
import matplotlib.pyplot as plt
```

We had to solve the optimization problem:

$$\min_{\beta} LL = \min_{\beta} - \sum_i \left[y_i \log(h(x_i)) + (1 - y_i) \log(1 - h(x_i)) \right], \text{ where } h(x) = \frac{1}{1 + e^{-x}}$$

Solving optimization problem for Logistic Regression

We use a gradient based approach called: **Gradient Descent**

Like before, assuming $g(x) = 1/(1 + e^{-x})$, differentiating $g(x)$ with respect to x we obtain:

$$g'(x) = \frac{-1}{(1 + e^{-x})^2} e^{-x} (-1) = \frac{e^{-x}}{(1 + e^{-x})^2} = \frac{1}{1 + e^{-x}} \cdot \frac{e^{-x}}{1 + e^{-x}} = g(x)(1 - g(x))$$

The main idea here is to use the following gradient step iteratively to converge to a good solution of β :

$$\beta^{j+1} \leftarrow \beta^j - \alpha \nabla LL \quad (2)$$

As an example, assuming we are fitting the model $h(x) = 1/(1 + e^{-(\beta_0 + \beta_1 x)})$. Hence, the polynomial in the power of e is a straight line with 2 parameters: $\beta = [\beta_0, \beta_1]$. For this case, the vector form of (2) will be

$$\begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}^{j+1} \leftarrow \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}^j - \alpha \begin{bmatrix} \partial LL / \partial \beta_0 \\ \partial LL / \partial \beta_1 \end{bmatrix}^j$$

where the left hand side is the estimate of β at iteration: $j + 1$, by using the previous estimate of β at iteration: j , offset by the gradient information.

Computing: $\nabla L L$ for the gradient based update in (2):

$$\begin{aligned}\nabla L L &= -\nabla \sum_i \left[y_i \log(h(x_i)) + (1 - y_i) \log(1 - h(x_i)) \right] \\ &= -\sum_i \left[y_i \nabla \log(h(x_i)) + (1 - y_i) \nabla \log(1 - h(x_i)) \right]\end{aligned}\quad (3)$$

Computing the required terms

$$\nabla \log(h(x)) = \nabla \log(g(X\beta)) = \frac{1}{g(X\beta)} g(X\beta)(1 - g(X\beta))X = (1 - g(X\beta))X = (1 - h(x))X$$

$$\nabla \log(1 - h(x)) = \nabla \log(1 - g(X\beta)) = \frac{1}{1 - g(X\beta)} - g(X\beta)(1 - g(X\beta))X = -g(X\beta)X$$

Putting it back in (3)

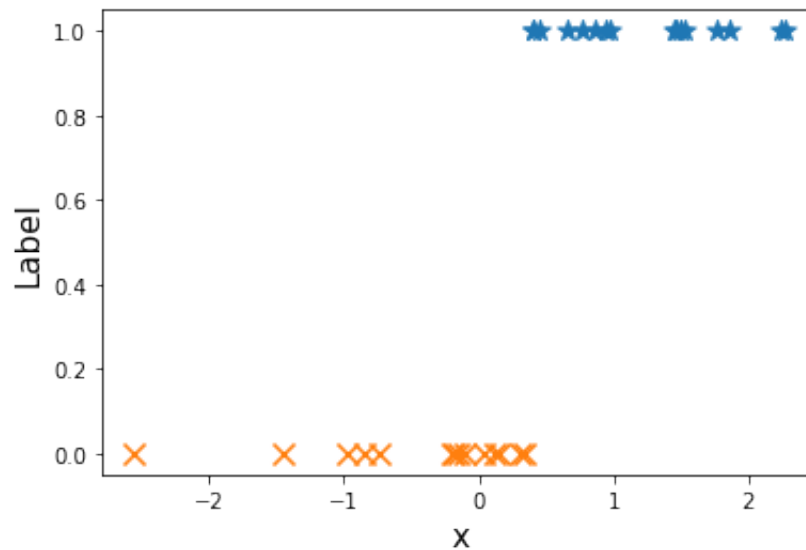
$$\begin{aligned}\nabla L L &= -\sum_i \left[y_i \nabla \log(h(x_i)) + (1 - y_i) \nabla \log(1 - h(x_i)) \right] \\ &= -\sum_i \left[y_i (1 - h(x_i)) X_i - (1 - y_i) h(x_i) X_i \right] \\ &= -\sum_i \left[y_i X_i - y_i h(x_i) X_i - h(x_i) X_i + y_i h(x_i) X_i \right] \\ &= -\sum_i \left[y_i X_i - h(x_i) X_i \right] \\ &= -\sum_i \left[(y_i - \hat{y}_i) X_i \right] \\ &= -X^T (Y - \hat{Y})\end{aligned}$$

Hence the final gradient step is:

$$\boxed{\beta^{j+1} \leftarrow \beta^j + \alpha X^T (Y - \hat{Y})}$$

Data

```
In [10]: np.random.seed(0)
l = 30
xs = np.random.randn(l).reshape(l,1)
ys = np.ones(l)
ys[xs[:,0] < 0.4] = 0
### Plotting
plt.scatter(xs[ys == 1],ys[ys == 1],marker = '*',s = 100)
plt.scatter(xs[ys == 0],ys[ys == 0],marker = 'x',s = 100)
plt.xlabel('x',size = 15)
plt.ylabel('Label',size = 15)
plt.show()
```



```
In [ ]: Logistic Regression code
```

```

In [13]: def h(X,beta):
    ## returns the value of the sigmoid function
    ypred = 1/(1 + np.exp(-1*X.dot(beta)))
    return ypred

def model_cost(X,y,beta):
    ## computes the value of the model fitting cost
    cost = 0
    for j in range(X.shape[0]):
        term1 = y[j]*np.log(h(X[j,:],beta))[0]
        term2 = (1 - y[j])*np.log(1 - h(X[j,:],beta))[0]
        cost = cost - (term1+term2)
    return cost

def gradient_descent(X,y,beta0,alpha,iteration):
    ## Implemented iteration for the gradient descent algorithm
    ypred = h(X,beta0)
    beta = [beta0]
    cost = [model_cost(X,y,beta0)]
    for j in range(iteration):
        grad_l = -1*X.T.dot(y.reshape(-1,1) - ypred)
        beta_new = beta[-1] - alpha * grad_l
        ypred = h(X,beta_new)
        beta.append(beta_new)
        print('Cost at iteration '+str(j)+' is: ',model_cost(X,y,beta_
        cost.append(model_cost(X,y,beta_new))
    return [beta_new,cost]

```

```
In [17]: beta = np.ones([2,1])
alpha = 0.01
iteration = 1000
Xs = np.concatenate((np.ones([len(xs),1]),xs),axis = 1)
beta_new, cost = gradient_descent(Xs,ys,beta,alpha,iteration)
```

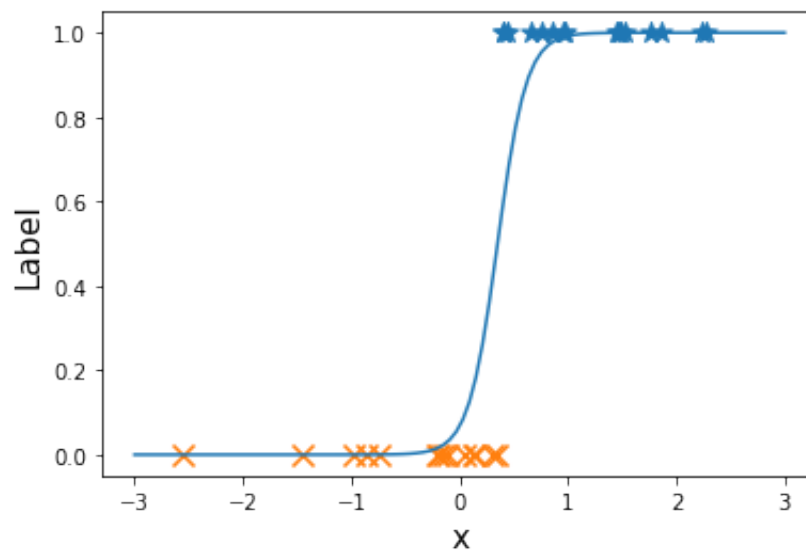
```
Cost at iteration 69 is: 7.194245216843533
Cost at iteration 70 is: 7.163521248498534
Cost at iteration 71 is: 7.133356590732667
Cost at iteration 72 is: 7.103733635129696
Cost at iteration 73 is: 7.074635562213052
Cost at iteration 74 is: 7.046046295683665
Cost at iteration 75 is: 7.017950459889214
Cost at iteration 76 is: 6.99033334025949
Cost at iteration 77 is: 6.963180846466945
Cost at iteration 78 is: 6.936479478093618
Cost at iteration 79 is: 6.910216292605354
Cost at iteration 80 is: 6.8843788754520245
Cost at iteration 81 is: 6.85895531212862
Cost at iteration 82 is: 6.833934162046552
Cost at iteration 83 is: 6.809304434077604
Cost at iteration 84 is: 6.7850555636448
Cost at iteration 85 is: 6.761177391245267
Cost at iteration 86 is: 6.737660142299762
Cost at iteration 87 is: 6.714494408232456
Cost at iteration 88 is: 6.691671128692404
```

```
In [18]: beta_new
```

```
Out[18]: array([[ -2.62491271],
               [ 7.48557568]])
```

```
In [27]: xpred = np.linspace(-3,3,100).reshape(-1,1)
Xpred = np.concatenate((np.ones([len(xpred),1]),xpred),axis = 1)
pred = h(Xpred,beta_new)
```

```
In [29]: np.random.seed(0)
l = 30
xs = np.random.randn(l).reshape(l,1)
ys = np.ones(l)
ys[xs[:,0] < 0.4] = 0
### Plotting
plt.scatter(xs[ys == 1],ys[ys == 1],marker = '*',s = 100)
plt.plot(xpred,pred,label='Prediction')
plt.scatter(xs[ys == 0],ys[ys == 0],marker = 'x',s = 100)
plt.xlabel('x',size = 15)
plt.ylabel('Label',size = 15)
plt.show()
```



In []: