

## Lojistik Regresyon (Logistic Regression)

Lojistik regresyon normal regresyonun  $\theta^T x$  olarak kullandigi agirliklar (katsayilar) ile verinin carpimini alır ve ek bir filtre fonksiyonundan gecirerek onlari 0/1 degerleri baglaminda bir olasiliga esler. Yani elimizdeki veri pek cok boyutta veri noktaları ve o noktaların 0 ya da 1 olarak bir "etiketi" olacaktır. Mesela

```
from pandas import *
df = read_csv("testSet.txt", sep='\t', names=['x', 'y', 'labels'], header=None)
df['intercept']=1.0
data = df[['intercept', 'x', 'y']]
labels = df['labels']
print df[['x', 'y', 'labels']][:10]
```

	x	y	labels
0	-0.017612	14.053064	0
1	-1.395634	4.662541	1
2	-0.752157	6.538620	0
3	-1.322371	7.152853	0
4	0.423363	11.054677	0
5	0.406704	7.067335	1
6	0.667394	12.741452	0
7	-2.460150	6.866805	1
8	0.569411	9.548755	0
9	-0.026632	10.427743	0

Goruldugu gibi veride  $x, y$  boyutlari icin etiketler (labels) verilmiş. Lojistik regresyon bu veriyi kullanarak eğitim sonrası  $\theta$ 'ları elde eder, bunlar katsayılarımızdır, artık bu katsayıları hiç gormedigimiz yeni bir veri üzerinde 0/1 etiketlerinin tahminini yapmak için kullanabiliriz.

Filtre fonksiyonu için kullanılan bir fonksiyon sigmoid fonksiyonudur,  $g(x)$  ismini verelim,

$$g(x) = \frac{e^x}{1 + e^x}$$

Bu nasıl bir fonksiyondur, kabaca davranisini nasıl tarif ederiz? Cebirsel olarak bakarsak, fonksiyon öyle bir durumda ki ne zaman bir  $x$  degeri gecerse, bu deger ne kadar büyük olursa olsun, bölendeki deger her zaman bölünenden 1 daha fazla olacaktır bu da fonksiyonun sonucunun 1'den her zaman küçük olmasını garantiler. Çok küçük  $x$  degerleri için bolum sonucu biraz daha büyük olacaktır tabii, vs.

Daha temiz bir ifade için bölün ve bölüneni  $e^{-x}$  ile carpalım,

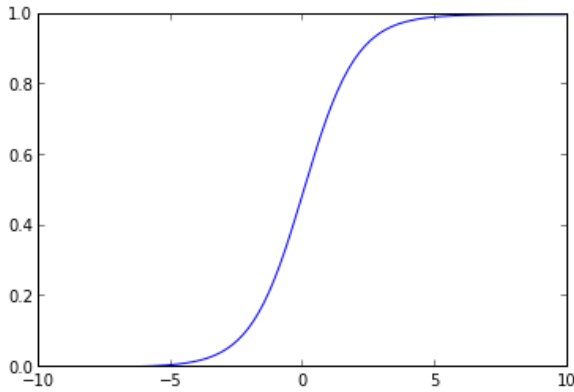
$$g(x) = \frac{e^x e^{-x}}{e^{-x} + e^x e^{-x}}$$

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

Sigmoid fonksiyonun ”-sonsuzluk ile +sonsuzluk arasindaki degerleri 0 ve 1 arasina esledigi / indirgedigi (map)” ifadesi de litaraturde mevcuttur.

```
def sigmoid(arr):
    return 1.0/(1+exp(-arr))

x = np.array(arange(-10.0, 10.0, 0.1))
plt.plot(x,sigmoid(x))
plt.savefig('logreg1.png')
```



Ustteki grafige bakinca katsayilarla carpim, toplam ardindan sonucun niye bu fonksiyona verildigini anlamak mumkun. Sigmoid'in 0 seviyesinden 1 seviyesine ziplayisi oldukca hizli ve  $x$  kordinati baglaminda (ve 0.5'ten kucuk  $y$ 'ye eslenen) sifir oncesi bolgesi, ayni sekilde sifir sonrasi (ve 0.5'ten buyuk  $y$ 'ye eslenen) bolgesi oldukca buyuk. Yani bu fonksiyonu secmekle veriye katsayilarla carpilip 0 ya da 1 bolgesi altina dusmesi icin oldukca genis bir sans veriyoruz. Boylece veriyi iki parcaya ayirmak icin sansimizi arttirmis oluyoruz.

Peki sigmoid fonksiyonu bir olasilik fonksiyonu (dagilimi) olarak kullanilabilir mi? Entegralini alalim, ve  $-/+$  sonsuzluklar uzerinden alan hesabi yapalim, sonucun 1 cikmasi gerekli,

```
import sympy
x = sympy.Symbol('x')
print sympy.integrate('1/(1+exp(-x))')

x + log(1 + exp(-x))
```

Daha temizlemek icin

$$x + \ln(1 + e^{-x})$$

$x$  ifadesi ayni zamanda suna esittir  $x = \ln(e^x)$ . Bu ifade bize kolaylik saglayacak boylece,

$$\ln e^x + \ln(1 + e^{-x})$$

diyebiliriz. Dogal log'un ( $\ln$ ) carpimlari toplamlara donusturdugunu biliyoruz, bunu tersinden uygulayalim,

$$\ln(e^x \cdot 1 + e^x e^{-x})$$

$$\ln(e^x + 1) = \ln(1 + e^x)$$

```
print log (1+exp(-inf))
print log(1+exp(inf))
```

0.0

inf

Demek ki fonksiyon bir olasilik dagilimi olamaz, cunku egri altindaki alan sonsuz buyuklugunde. Aslinda bu fonksiyonun kumulatif dagilim fonksiyonu (cumulative distribution function -CDF-) ozellikleri vardir, yani kendisi degil ama turevi bir olasilik fonksiyonu olarak kullanilabilir (bu konumuz disinda). Her neyse, sigmoid'in bir CDF gibi hareket ettigini  $g$ 'nin 0 ile 1 arasinda olmasindan da anliyoruz, sonucta CDF alan demektir (yogunlugun integrali) ve en ust degeri 1 demektir, ki bu CDF tanimina uygundur.

Simdi elimizde olabilecek  $k$  tane degisken ve bu degiskenlerin bilinmeyen katsayilari icin 0 ve 1'e eslenecek bir regresyon olusturalim. Diyelim ki katsayilar  $\theta_0, \dots, \theta_k$ . Bu katsayilari degiskenler ile carpip toplayarak  $h(x)$ 'e verelim, (0/1) cikip cikmayacagi katsayilara bagli olacak, verideki etiketler ile  $h(x)$  sonucu arasinda bir baglanti kurabilirsek, bu bize katsayilari verebilir. Bu modele gore eger  $\theta$ 'yi ne kadar iyi secersek, eldeki veriye etiketlerine o kadar yaklasmis olacagiz. Simdi sigmoid'i katsayilarla beraber yazalim,

$$h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

"Veriye olabildigince yaklasmak icin en iyi  $\alpha$ 'yi bulmak" sozu bize maksimum olurluk (maximum likelihood) hesabini hatirlatmalidir. Bu hesaba gore icinde bilinmeyen  $\alpha$ 'yi barindiran formulun uzerinden tum verinin sonuclarinin teker teker birbiri ile carpimi olabildigince buyuk olmalidir. Bu ifadeyi maksimize edecek  $\alpha$  veriye en uygun  $\alpha$  olacaktir.

Simdi her iki etiket icin ve sigmoid'i kullanarak olasilik hesaplarini yapalim,

$$P(y = 1|x; \theta) = h_{\theta}(x)$$

$$P(y = 0|x; \theta) = 1 - h_{\theta}(x)$$

Not: Olasilik degerleri (buyuk  $P(\cdot)$  ile) ve CDF fonksiyonlari olurluk hesabinda kullanilabilir.  $P(\cdot)$  ile CDF baglantisi var,  $P(X < x)$  gibi kumulatif alansal hesaplarin CDF uzerinden gerceklestirilebildigini hatirlayalim.

Devam edelim, hepsi bir arada olacak sekilde yanyana koyarsak ve sonuca,  $y$ 'yi dogru tahmin edip etmedigimizin olcumunu de eklersek,

$$p(y|x; \theta) = (h_{\theta}(x))^y (1 - h_{\theta}(x))^{1-y}$$

Olurluk icin tum veri noktalarini teker teker bu fonksiyona gecip sonuclarini carpacagiz (ve verilerin birinden bagimsiz olarak uretildigini farzediyoruz), eger  $m$  tane veri noktası var ise

$$L(\theta) = \prod_{i=1}^m (h_{\theta}(x^i))^{y^i} (1 - h_{\theta}(x^i))^{1-y^i}$$

Eger log'unu alirsak carpimlar toplama donusur, isimiz daha rahatlasir,

$$l(\theta) = \log L(\theta)$$

$$= \sum_{i=1}^m y^i \log((h_{\theta}(x^i))) + (1 - y^i) \log((1 - h_{\theta}(x^i)))$$

Iste bu ifadenin maksimize edilmesi gerekiyor.

Ama daha fazla ilerlemeden once bir esitlik ve bir turev gostermemiz gerekiyor. Once esitlik

$$1 - g(z) = g(-z)$$

Ispat

$$1 - \frac{1}{1 + e^{-z}} = \frac{1 + e^{-z} - 1}{1 + e^{-z}}$$

$$\frac{e^{-z}}{1 + e^{-z}} = \frac{1}{1 + e^z}$$

Hakikaten son esitligin sag tarafina bakarsak,  $g(-z)$ 'yi elde ettigimizi goruyoruz.

□

Simdi tureve gelelim,

$$g'(z) = \frac{d}{dz} \frac{1}{1 + e^{-z}}$$

Ispat

$$= \frac{1}{(1 + e^{-z})^2} (e^{-z})$$

$e^{-z}$  turevinden bir eksi isareti geleceğini beklemis olabilirsiniz, fakat hatırlayacağımız üzere

$$\frac{d}{dx} \frac{1}{1 + x} = \frac{-1}{(1 + x)^2}$$

Yani eksiler birbirini yoketti. Simdi iki ustteki denklemin sag tarafini acalim

$$= \frac{1}{1 + e^{-z}} \frac{e^{-z}}{1 + e^{-z}}$$

$$= \frac{1}{1 + e^{-z}} \frac{1}{1 + e^z}$$

Carpimda iki bolum var, bolumler  $g(z)$  ve  $g(-z)$  olarak temsil edilebilir, ya da  $g(z)$  ve  $1 - g(z)$ ,

$$= g(z)(1 - g(z))$$

□

Bu baglamda ilginç bir diger denklem log sansi (log odds) denklemdir. Eger ilk bastaki denklemi dusunursek,

$$p = P(y = 1|x; \theta) = g(z) = \frac{e^z}{1 + e^z}$$

Bu denklem 1 olma olasiligini hesapliyor. Temiz bir denklem log sansi olabilir ki bu denklem olma olasiligini olmama olasiligina boler ve log alır.

$$\log\left(\frac{p}{1-p}\right)$$

olarak gosterilir. Simdi biraz daha cambazlik,  $1 - g(z) = g(-z)$  demistik, ve  $g(-z)$ 'nin de ne oldugunu biliyoruz  $\frac{1}{1+e^z}$ , log sansini bu sekilde yazalim,  $\frac{1}{1+e^z}$  ile bolelim daha dogrusu  $1 + e^z$  ile carpalim ve log alalim,

$$\log\left(\frac{e^z}{1+e^z}1 + e^z\right) = \log(e^z) = z = \theta^T x$$

Artik olurluk denkleminde donebiliriz. Olurlugu nasıl maksimize ederiz? Gradyan cikisi (gradient ascent) kullanılabilir. Eger olurluk  $l(\theta)$ 'nin en maksimal oldugu noktadaki  $\theta$ 'yi bulmak istiyorsak (dikkat sadece olurlugun en maksimal noktasini aramiyoruz, o noktadaki  $\theta$ 'yi ariyoruz), o zaman bir  $\theta$  ile baslariz, ve adim adim  $\theta$ 'yi maksimal olana dogru yaklastiririz. Formül

$$\theta_{yeni} = \theta_{eski} + \alpha \nabla_{\theta} l(\theta)$$

Ustteki formül niye isler? Cunku gradyan  $\nabla_{\theta} l(\theta)$ , yani  $l(\theta)$ 'nin gradyeni her zaman fonksiyon artisinin en fazla oldugu yonu gosterir. Demek ki o yone adim atmak, yani  $l(\theta)$ 'a verilen  $\theta$ 'yi o yonde degistirmek (degisim tabii ki  $\theta$  bazinda,  $\theta$ 'nin degisimi), bizi fonksiyonun bir sonraki noktasina yaklastiracaktir. Sabit  $\alpha$  bir tek sayi sadece, atilan adimin (hangi yonde olursa olsun) olcegini azaltip / arttirabilmek icin disari-dan eklenir. Adim yonu vektor, bu sabit bir tek sayi. Carpimlari vektoru azaltir ya da cogaltir [3].

Simdi  $\nabla_{\theta} l(\theta)$  turetmemiz gerekiyor.

Eger tek bir  $\frac{\partial l(\theta)}{\partial \theta_j}$ 'yi hesaplarsak ve bunu her  $j$  icin yaparsak, bu sonuclari bir vektörde ustuste koyunca  $\nabla_{\theta} l(\theta)$ 'yi elde ederiz.

$$\begin{aligned} \frac{\partial l(\theta)}{\partial \theta_j} &= y \frac{\frac{\partial}{\partial \theta_j} g(\theta^T x)}{g(\theta^T x)} - (1 - y) \frac{\frac{\partial}{\partial \theta_j} g(\theta^T x)}{1 - g(\theta^T x)} \\ &= \left( y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \end{aligned}$$

Simdi en sagdaki kismi acalim,

$$\frac{\partial}{\partial \theta_j} g(\theta^T x) = g'(\theta^T x) \frac{\partial}{\partial \theta_j} \theta^T x = g'(\theta^T x) x_j$$

$\frac{\partial}{\partial \theta_j} \theta^T x$  nasıl  $x_j$  haline geldi? Cunku tum  $\theta$  vektorunun kismi turevini aliyoruz fakat o kismi turev sadece tek bir  $\theta_j$  icin, o zaman vektördeki diger tum ogeler sifir olacaktır,

sadece  $\theta_j$  1 olacak, ona tekabül eden  $x$  ogesi, yani  $x_j$  ayakta kalabilecek, diğer  $x$  ogelerinin hepsi sıfırla carpılmış olacak.

Türevin kendisinden de kurtulabiliriz şimdi, daha önce gösterdiğimiz eşitliği devreye sokalım,

$$= g(\theta^T x)(1 - g(\theta^T x))x_j$$

Bu son formülü 3 üstteki formülün sağ tarafına geri koyarsak, ve basitleştirirsek,

$$(y(1 - g(\theta^T x)) - (1 - y)g(\theta^T x))x_j$$

Carpımı daha temiz görmek için sadece  $y, g$  harflerini kullanırsak,

$$(y(1 - g) - (1 - y)g)x_j = (y - yg - g + yg)x_j = (y - g)x_j$$

yani

$$= (y - g(\theta^T x))x_j$$

$$= (y - h_\theta(x))x_j$$

İşte  $\nabla_\theta l(\theta)$  için ne kullanacağımızı bulduk. O zaman

$$\theta_{yeni} = \theta_{eski} + \alpha(y - h_\theta(x))x_j$$

Her  $i$  veri noktası için

$$\theta_{yeni} = \theta_{eski} + \alpha(y^i - h_\theta(x^i))x_j^i$$

Kodu isletelim,

```
def grad_ascent(data_mat, label_mat):
    m,n = data_mat.shape
    label_mat=label_mat.reshape((m,1))
    alpha = 0.001
    iter = 500
    theta = ones((n,1))
    for k in range(iter):
        h = sigmoid(dot(data_mat,theta))
        error = label_mat - h
        theta = theta + alpha * dot(data_mat.T,error)
```

```

    return theta

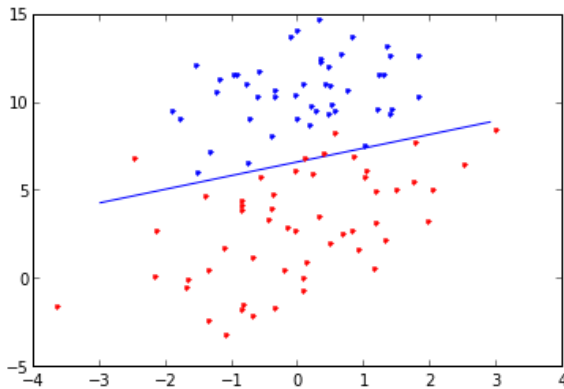
theta = np.array(grad_ascent(array(data),array(labels).T ))
print theta.T

[[ 4.12414349  0.48007329 -0.6168482 ]]

def plot_theta(theta):
    x = np.array(arange(-3.0, 3.0, 0.1))
    y = np.array((-theta[0]-theta[1]*x)/theta[2])
    plt.plot(x, y)
    plt.hold(True)
    class0 = data[labels==0]
    class1 = data[labels==1]
    plt.plot(class0['x'],class0['y'],'b.')
    plt.hold(True)
    plt.plot(class1['x'],class1['y'],'r.')
    plt.hold(True)

plot_theta(theta)
plt.savefig('logreg2.png')

```



Ustteki kod bir dongu icinde belli bir  $x$  noktasından baslayarak gradyan inisi yapti ve optimal  $\theta$  degerlerini, yani regresyon agirliklarini (weights) hesapladi. Sonra bu agirliklari bir ayrac olarak ustte grafikledi. Ayracin oldukca iyi degerler buldugu belli oluyor.

Rasgele Gradyan Cikisi (Stochastic Gradient Ascent)

Acaba  $\theta$ 'yi guncellerken daha az veri kullanmak mumkun mu? Yani yon hesabi icin surekli tum veriyi kullanmasak olmaz mi?

Olabilir. Guncellemeyi sadece tek bir veri noktasini kullanarak yapabiliriz. Yine gradyani degistirmis oluruz, sadece azar azar degisim olur, fakat belki de bu sekilde sonuca daha cabuk ulasmak mumkun olacaktir.



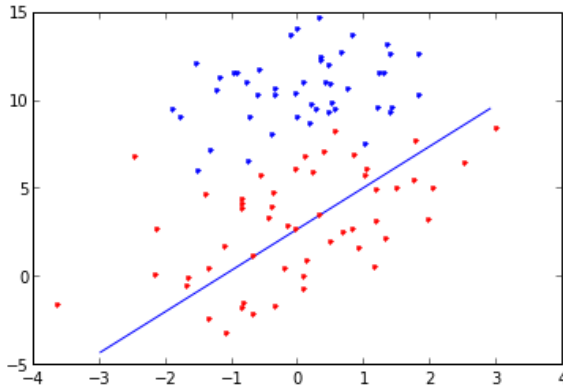
Kodlama acisindan,  $\theta$  guncellemesi icin buldugumuz formulu tek nokta bazinda da vermistik. O zaman o tek noktayi sirayla alip guncellersek, otomatik olarak yeni bir sekilde gradyan cikisi yapmis oluruz.

```
def stoc_grad_ascent0(data_mat, label_mat):
    m,n = data_mat.shape
    label_mat=label_mat.reshape((m,1))
    alpha = 0.01
    theta = ones((n,1))
    for i in range(m):
        h = sigmoid(sum(dot(data_mat[i],theta)))
        error = label_mat[i] - h
        theta = theta + alpha * data_mat[i].reshape((n,1)) * error
        theta = theta.reshape((n,1))
    return theta

theta = np.array(stoc_grad_ascent0(array(data),array(labels).T ))
print theta.T

[[ 1.01702007  0.85914348 -0.36579921]]

plot_theta(theta)
plt.savefig('logreg3.png')
```



Neredeyse isimiz tamamlandi. Ustteki grafik pek iyi bir ayrac gostermedi. Niye? Problem cok fazla salinim (oscillation) var, yani degerler cok fazla uc noktalar arasinda gidip geliyor. Ayrica veri noktalarini sirayla isliyoruz, veri tabii ki rasgele bir sekilde siralanmis olabilir, ama siralanmamissa, o zaman algoritmaya raslantisal noktalar vermek icin kod icinde zar atmamiz lazim. Metotun ismi "rasgele (stochastic)" gradyan cikisi, bu rasgelelik onemli. 2. problemi duzeltmek icin yapilacak belli, 1. problem icin  $\alpha$  degeri her dongude belli oranda kucultulerek (yani  $\alpha$  artik sabit degil) sonuca yaklasirken oradan buraya savrulmasini engellemis olacagiz. Yeni kod altta,

```

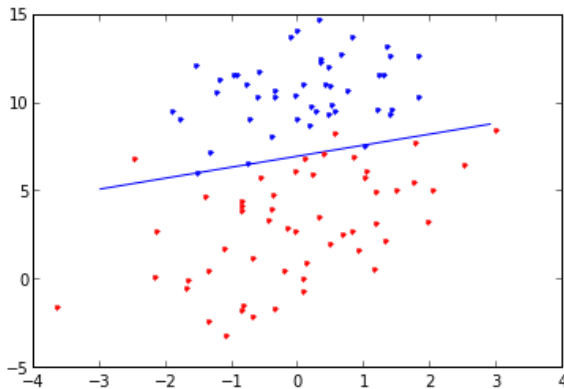
def stoc_grad_ascent1(data_mat, label_mat):
    m,n = data_mat.shape
    iter = 150
    label_mat=label_mat.reshape((m,1))
    alpha = 0.01
    theta = ones((n,1))
    for j in range(iter):
        data_index = range(m)
        for i in range(m):
            alpha = 4/(1.0+j+i)+0.0001
            rand_index = int(random.uniform(0,len(data_index)))
            h = sigmoid(sum(dot(data_mat[rand_index],theta)))
            error = label_mat[rand_index] - h
            theta = theta + alpha * data_mat[rand_index].reshape((n,1)) * error
            theta = theta.reshape((n,1))
    return theta

theta = np.array(stoc_grad_ascent1(array(data),array(labels).T ))
print theta.T

[[ 14.67440542   1.30317067  -2.08702677]]

plot_theta(theta)
plt.savefig('logreg4.png')

```



Sonuc cok iyi, ayrica daha az islemlerle bu noktaya eristik, yani daha az islem ve daha hizli bir sekilde sonuca ulasmis olduk.

Tahmin (Prediction)

Elde edilen agirliklari tahmin icin nasil kullaniriz? Bu agirliklari alip, yeni veri noktası ile carpip sonuclari sigmoid'den gecirdigimiz zaman bu noktanin "1 etiketi olma olasiligini" hesaplamis olacagiz. Ornek (diyelim ki mevcut veri noktası icinden bir veriyi, -mesela 15. nokta- sanki yeniymis gibi sectik)

```
pt = df.ix[15,['intercept','x','y']]
print sigmoid(dot(array(pt), theta)),
print 'label =',labels[15]
```

```
[ 0.99999653] label = 1
```

Oldukca yuksek bir olasilik cikti, ve hakikaten de o noktanin gercek degeri 1 imis.

Kaynaklar

[1] <http://cs229.stanford.edu/notes/cs229-notes1.pdf>

[2] Harrington, P. Machine Learning in Action

[3] Bu sekilde azar azar sonuca yaklasmaya ugrasmak tabii ki her fonksiyon icin gecerli degildir, cunku eger fonksiyonda "yerel maksimumlar" var ise, gradyan cikisi bu noktalarda takilip kalabilir (o yerel tepelerde de birinci turev sifirlanir, gradyanin kafasi karisir). Gradyan metotunun kullanmadan once fonksiyonumuzun tek (global) bir maksimumu olup olmadigini dusunmemiz gerekir. Fakat sanliyiz ki olurluk fonksiyonu tam da boyle bir fonksiyondur (sans degil tabii, bu ozelligi sebebiyle secildi). Fonksiyon icbukeydir (concave), yani tek bir tepe noktası vardır. Bir soru daha: olurlugun icbukey oldugunu nasil anladik? Fonksiyona bakarak pat diye bunu soylemek mumkun, degiskenlerde polinom baglaminda kupsel ve daha ustü seviyesinde ustellik yok, ayrıca log, exp icbukeyligi bozmuyor.