

Lojistik Regresyon

Lojistik regresyon normal regresyonun $\theta^T x$ olarak kullandigi agirliklar / katsayilar ve veri carpimini 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. Lojistik regresyon egitimi sonrasi elimize gecen θ 'lar, katsayilarimizdir, artik bu katsayilari tahmin yapmak icin kullanabiliriz. Filtre fonksiyonu icin kullanılan bir fonksiyon sigmoid fonksiyonudur.

Sigmoid fonksiyonu

$$\frac{e^x}{1 + e^x}$$

Sezgisel olarak bakarsak, fonksiyon oyle bir durumda ki, ne zaman bir x degeri gecerse, bu deger ne kadar buyuk olursa olsun, bolendeki deger her zaman bolunenden 1 daha fazla olacaktır bu da fonksiyonun sonucunun 1'den her zaman kucuk olmasini garantiler. Cok kucuk x degerleri icin bolum sonucu biraz daha buyuk olacaktır tabii, vs. Daha temiz bir ifade icin bolen ve boluneni e^{-x} ile carpalim,

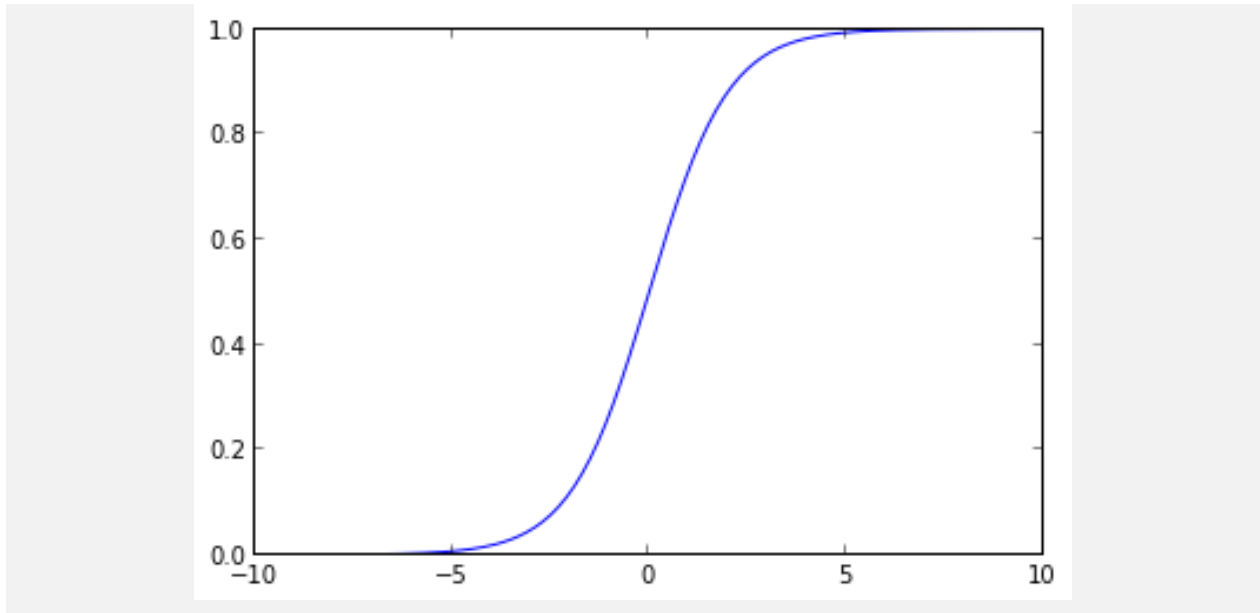
$$\frac{e^x e^{-x}}{e^{-x} + e^x e^{-x}}$$

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

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))  
plot(x,sigmoid(x))
```

```
[<matplotlib.lines.Line2D at 0xafd074c>]
```



Peki üstteki fonksiyon bir olasılık fonksiyonu olabilir mi? Entegralini alalım,

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

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

Daha temizlemek için

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

x ifadesi aynı zamanda suna esittir $x = \ln(e^x)$. Bu ifade bize kolaylık sağlayacak böylece,

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

diyebiliriz. Doğal log'un (\ln) carpımları toplamlara donusturdugunu biliyoruz, bunu tersinden uygulayalım,

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

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

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

0.0
inf

Demek ki fonksiyon bir olasılık dağılımı olamaz, çünkü eğri altındaki alan sonsuz büyüklüğünde. Aslında bu fonksiyonun kümülatif dağılım fonksiyonu (cumulative distribution function -CDF-) özellikleri vardır, yani kendisi değil ama türevi bir olasılık fonksiyonu olarak kullanılabilir (bu konumuz dışında).

CDF durumunda g 'nin 0 ile 1 arasında olması da dağılım altındaki alanın en fazla 1 olabilmesi durumunu ortaya çıkarır ki bu CDF tanımına uygundur.

Şimdi elimizde olabilecek k tane değişken ve bu değişkenlerin bilinmeyen katsayıları için 0 ve 1'e eslenecek bir regresyon oluşturalım. Diyelim ki katsayılar $\theta_0, \dots, \theta_k$. Bu katsayıları değişkenler ile çarpıp toplayarak $h(x)$ 'e verelim, (0/1) çıkıp çıkmayacağı katsayılarla bağlı olacak, verideki etiketler ile $h(x)$ sonucu arasında bir bağlantı bize katsayıları verebilir. Bu modele göre eğer θ 'yi ne kadar iyi seçersek, eldeki veriye o kadar yaklaşımla olacağız.

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

“Veriye olabildigince yaklaştırmak için en iyi α 'yi bulmak” sözü bize maksimum olurluk (maximum likelihood) hesabını hatırlatmalı. Bu hesaba göre içinde bilinmeyen α 'yi barındıran formülün üzerinden tüm verinin sonuçlarının teker teker birbiri ile çarpımı olabildigince büyük olmalıdır. Bu ifadeyi maksimize edecek α veriye en uygun α olacaktır.

Şimdi olasılıkları düşünelim

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

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

Not: Olasılık değerleri (büyük $P(\cdot)$ ile), CDF fonksiyonları (dağılım olmasa da) olurluk hesabında kullanılabilir. Bu arada $P(X < x)$ gibi alansal hesaplar CDF üzerinden gerçekleştirilebildiğini hatırlatalım.

Hepsi bir arada olacak şekilde yanyana koyarsak ve sonuçta, y 'ye, ne kadar yaklaşıp yaklaşılmadığımızın ölçümünü de eklersek,

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

Olurluk için tüm veri noktalarını teker teker bu fonksiyona geçip sonuçlarını çarpacağız (ve verilerin birinden bağımsız olarak üretildiğini farzediyoruz), eğer m tane veri noktası var ise

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

Eğer log'unu alırsak çarpımlar toplama dönüşür, isimiz daha rahatlasır,

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

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

Iste bu ifadenin maksimize edilmesi gerekiyor.

Ama daha fazla ilerlemeden önce bir esitlik ve bir turev gostermemiz gerekiyor. Önce 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,

$$g'(z) = \frac{d}{dz} \frac{1}{1 + e^{-z}} = \frac{1}{(1 + e^{-z})^2} (e^{-z})$$

e^{-z} turevinden bir eksi isareti gelecegini beklemis olabilirsiniz, fakat hatirlayacagimiz uzere

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

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 $1 - g(z)$,

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

□

Artik olurluk denklemine donebiliriz. Olurlugu nasıl maksimize ederiz? Gradyan cikisi (gradient ascent) kullanılabilir. Eger olurluk $l(\theta)$ 'nin en maksimal oldugu noktadaki θ 'yi bulmak istiyorsak (dikkat sadece olurlugun en maksimal noktasini aramiyoruz, o noktadaki θ 'yi ariyoruz), o zaman bir θ ile baslariz, ve adim adim θ '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 gradyani her zaman fonksiyon artisinin en fazla oldugu yonu gosterir. Demek ki o yone adim atmak, yani $l(\theta)$ 'a verilen θ 'yi o

yonde degistirmek (degisim tabii ki θ bazinda, θ 'nin degisimi), bizi fonksiyonun bir sonraki noktasina yaklastiracaktir. Sabit α bir tek sayi sadece, atilan adimin (hangi yonde olursa olsun) olcegini azaltip / arttirabilmek icin disaridan 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 vektorde 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,

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

$\frac{\partial}{\partial \theta_j} \theta^T x$ nasil x_j haline geldi? Cunku tum θ vektorunun kismi turevini aliyoruz fakat o kismi turev sadece tek bir θ_j icin, o zaman vektordeki diger tum ogeler sifir olacaktir, sadece θ_j 1 olacak, ona tekabul eden x ogesi, yani x_j ayakta kalabilecek, diger x ogelerinin hepsi sifir la carpilmis olacak.

Turevin kendisinden de kurtulabiliriz simdi, daha once gosterdigimiz esitligi devreye sokalim,

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

Bu son formulu 3 ustteki formulun sag tarafina geri koyarsak, ve basitlestirirsek,

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

Carpimi daha temiz gormek icin sadece y, g harflerini kullanirsak,

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

yani

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

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

Iste $\nabla_{\theta}l(\theta)$ icin ne kullanacagimizi bulduk. O zaman

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

Her i veri noktası icin

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

Veriye bakalim, ve kodu isletelim,

```

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']
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

```

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 ))
theta.T

```

```

array([[ 4.12414349,  0.48007329, -0.6168482 ]])

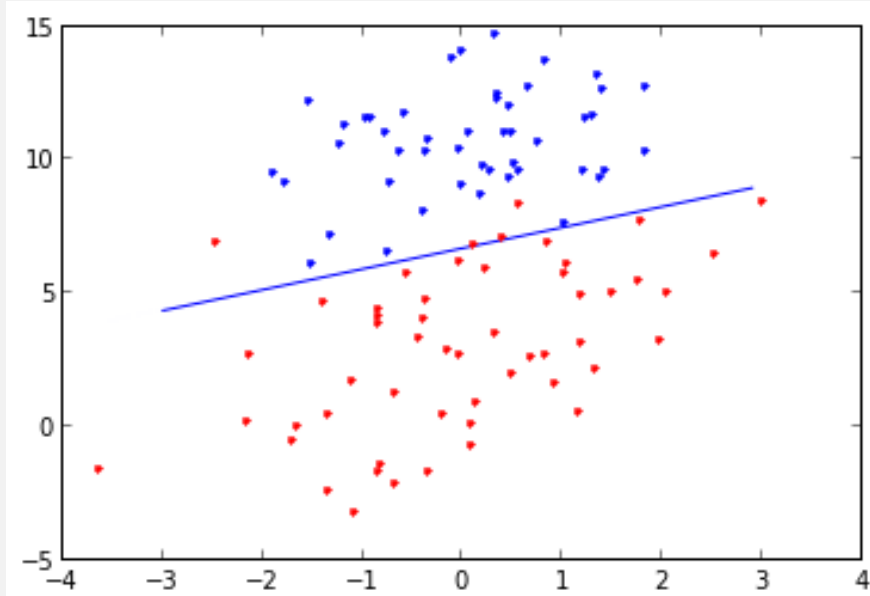
```

```

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)

```



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

Rasgele Gradyan Cikisi (Stochastic Gradient Ascent)

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

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

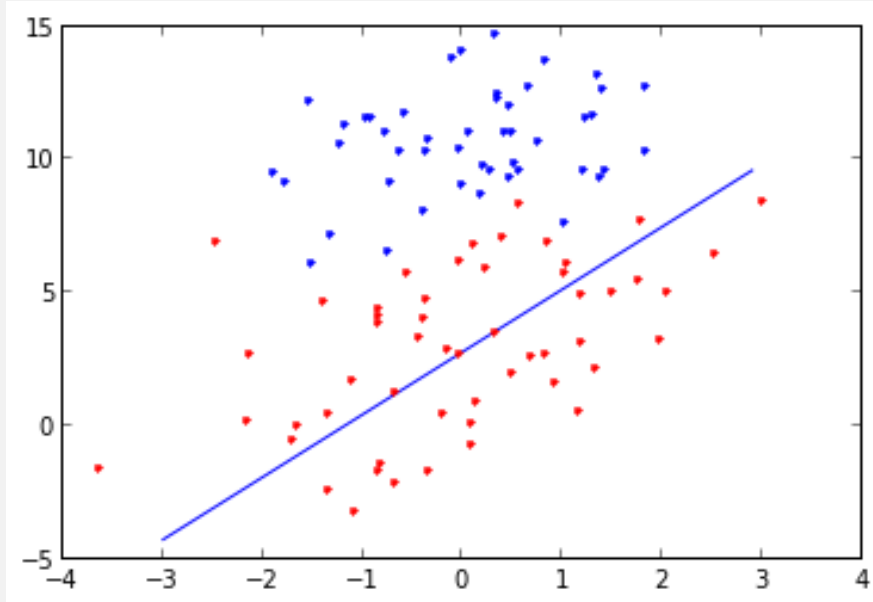
Kodlama acisindan, θ guncellemesi icin buldugumuz formulu tek nokta bazinda da vermistik. O zaman o tek noktayi sirayla alip guncellersek, otomatik olarak yeni bir sekilde gradyan cikisi yapmis oluruz.

```
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 ))
theta.T
```

```
array([[ 1.01702007,  0.85914348, -0.36579921]])
```

```
plot_theta(theta)
```



Neredeyse isimiz tamamlandı. Ustteki grafik pek iyi bir ayrac göstermedi. Niye? Problem çok fazla salınım (oscillation) var, yani değerler çok fazla uç noktalar arasında gidip geliyor. Ayrıca veri noktalarını sırayla işliyoruz, veri tabii ki rasgele bir şekilde sıralanmış olabilir, ama sıralanmamışsa, o zaman algoritmaya raslantısal noktaları vermek için kod içinde zar atmamız lazım. Metotun ismi “rasgele (stochastic)” gradyan çıkışı, bu rasgelelik önemli. 2. problemi düzeltmek için yapılacak belli, 1. problem için α değeri her döngüde belli oranda küçültülerek (yani α artık sabit değil) sonuca yaklaşıırken oradan buraya savrulmasını engellemiş olacağız. 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

```

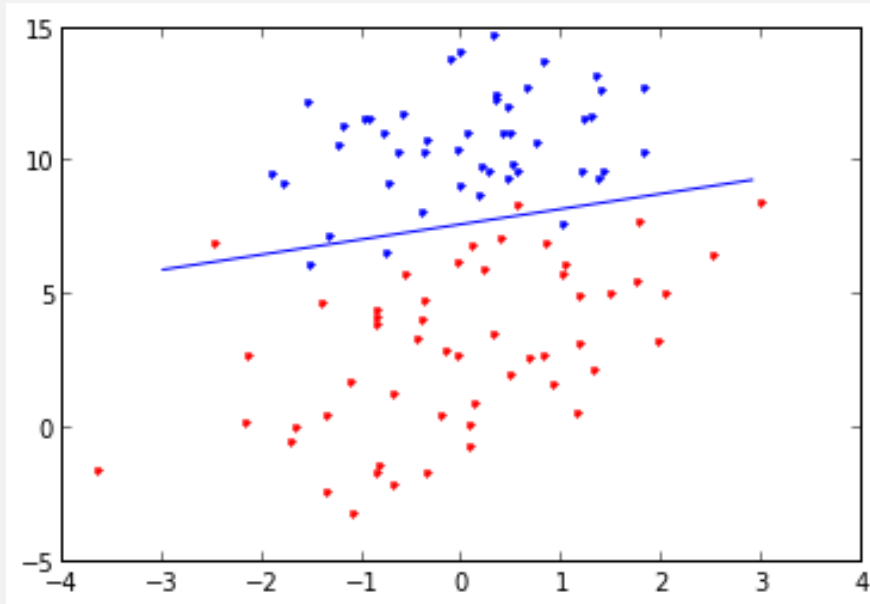
```

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

```

```
array([[ 12.99254269,   0.96527827,  -1.6941379 ]])
```

```
plot_theta(theta)
```



Sonuc cok iyi, ayrıca daha az işlemle bu noktaya eristik, yani daha az işlem ve daha hızlı bir şekilde sonuca ulaşmış olduk.

Tahmin (Prediction)

Elde edilen ağırlıkları tahmin için nasıl kullanırız? Bu ağırlıkları alıp, yeni veri noktası ile çarpıp sonuçları sigmoid'den geçirdiğimiz zaman bu noktanın “1 etiketi olma olasılığını” hesaplamış olacağız. Örnek (diyelim ki mevcut veri noktası içinden bir veriyi, -mesela 15. nokta- sanki yeniymiş gibi seçtik)

```

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

```

```
[ 0.99998631] 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 (ne de olsa bu noktalarda da birinci turev sifirlanir). 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 noktasi vardir. Bir soru daha: olurlugun icbukey oldugunu nasil anladik? Fonksiyona bakarak pat diye bunu soylemek mumkun, degiskenlerde polinom baglaminda kupsellik yok, ayrica log, exp icbukeyligi bozmuyor.