MAKİNE ÖĞRENMESİ

Ne öğreneceğiz? : Teorik temeller, genel kullanılan algoritmalar, bunların nasıl uygulandığı.

Nerelerde kullanılır? : Veri madenciliği, elle programlama yapamayacağımız uygulamalarda …

Makine öğrenmesi nedir?

Arthur Samuel (1959): Bilgisayarlara açık bir programlamaya gerek kalmaksızın kendi kendine öğrenme imkanı tanıyan alan.

* Dama oyun programı yazdı.
* 10000 oyunu bilgisayar kendi kendine oynuyor.
* Taşların tahta üzerindeki dizilimlerine göre kazanma ya da kaybetme durumunu öğrenmek.

Çok sayıda öğrenme algoritması var:

* Supervised learning: Bilgisayar birşeyin nasıl yapıldığı öğretilir, sonra bu bilgiyi kullanarak benzer işleri yapması beklenir.
* Unsupervised learning: Bilgisayarın birşeyi nasıl yapacağını kendi kendine öğrenmesi ve bu bilgiyi benzer işlerde kullanması.

SUPERVISED LEARNING

Örnekle başlayalım: İnşaat sektöründe ev fiyatlarının tahmini.

* Evin metrekaresi ile fiyatına dair veriler. Bunlar birbiriyle nasıl ilişkili.
  + Elimizde olan metrekareye karşılık gelen ev fiyatları veri setine Training set adı verilir.
    - Algoritmaya doğru cevapları içeren bir set veriyoruz.
    - Sonra algoritmanın bu training seti kullanarak verisi eilimizde olmayan durumlar için tahminde bulunmasını bekliyoruz.
    - Bu örnek için ev fiyatını tahmin etmesini bekliyoruz.

Bu tür problemlere regresyon problemi denir: bu problemlerde süreklilik vardır.

Diğer bir örnek: Bir tümörün boyutuna göre iyi ya da kötü huylu olmasını tahmin etmek.

* Bu bir sınıflandırma (clasification) problemi.
  + Tümör ya iyi huylu ya da kötü huylu olabilir.
* Tümörün boyutuna göre iyi huylu (0) ya da kötü huylu(1) olduğunu gösteren bir grafik çizilebilir.
* Aynı veriler yalnızca tek bir özellik kullanılarak, mesela tümör büyüklüğü, tek boyutta da ifade edilebilir.
* Birden fazla özellik kullanılırsa, mesela tümör büyüklüğü ve yaş, bu iki özelliğin birbiriyle ilişkisi bir grafikte gösterilebilir.
  + Veriler kümelere (gruplara) ayrılabilir. Ö

\*\*İdealde bir algoritmanın sonsuz sayıda özelliği işleyebilmesi istenir.

* Bu kadar çok özelliği bir bilgisayar nasıl işler? Bellek problem olur.
  + Çözüm support vector machine.

UNSUPERVISED LEARNING

Burada elimizdeki veriler etiketlenmemiştir. Yani özelliklerin birbirileriyle ilişkileri tanımlanmamıştır.

* Veri seti verilir ve bilgisayardan bu seti yapılandırması beklenir.
  + Bunu yapmanın yollarından birisi veriyi kümelendirmek (clustering).

LINEAR REGRESSION

* Daha önce incelediğimiz ev fiyatları örneğine geri dönelim:
  + Bu örnek bir supervised learning regresyon problemidir.
* Problemin çözümünde ilk gerekli şey nedir?
  + Bir veri seti
    - m: eğitim örneklerinin sayısı
    - x: input değişkenleri/özellikler
    - y: output değişkenleri, (hedef değişkenler)
      * (x,y) tek bir eğitim örneği
      * (xi,yi) belirli bir örnek (i. Eğitim örneği)
        + İ eğitim setinde index
* Diyelim ki eğitim setimiz aşağıdaki gibi olsun:

m2 cinsinde büyüklük Fiyat(1000 TL)

2014 460

1416 232

1534 315 m=47 (eğitim set örn)

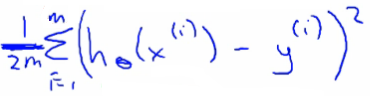
852 178

… …

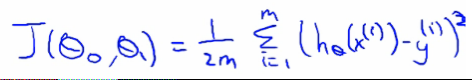
* Eğitim seti tanımlandı. Şimdi bu seti nasıl kullanacağız?
  + Eğitim seti alınır
  + Bir öğrenme algoritmasına gönderilir.
  + Bu algoritma bir output fonksiyonu üretir (h, hipotez)
    - Bu fonksiyon bir input değeri alır. Örneğin evin büyüklüğü.
    - Y değerini hesaplamaya çalışır.
* h (hipotes) nasıl temsil ederiz? Nasıl gösteririz.
  + Bunun anlamı Y, x’in lineer fonksiyonudur.
  + değerleri parametre olarak adlandırılır
    - başlangıç koşulu
    - eğim.
* Bu tür bir fonksiyon tek değişkenli lineer regresyondur. (Univariate linear regression)
* Özet:
  + Bir hipotez bir takım değişkenleri alır
  + Learning sistem tarafından belirlenen parametreleri kullanır
  + Bu inputa dayanarak bir tahminde bulunur.

LİNEER REGRESYON UYGULAMA (COST FUNCTION)

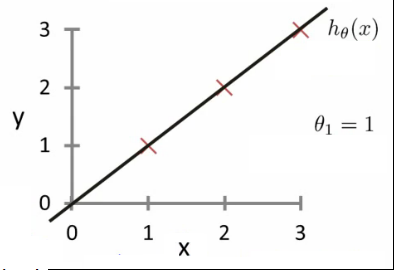
* Bir cost fonksiyonu verilerimize en iyi doğruyu nasıl fit edeceğimizi gösterir.
* için değerler seçmek gerekir (parametreler)
  + Farklı değerleri farklı fonksiyonlar verecektir.
  + ve --- y eksenini 1.5 te kesen yatay bir doğru elde ederiz.
  + pozitif eğime sahip bir doğru elde ederiz.
* Eğitim setimize bağlı olarak düz bir doğruyu oluşturacak parametreleri belirlemek istiyoruz.
  + Parametreleri öğle seç ki eğitim örneklerindeki y değerine mümkün olduğu kadar yakın olsun.
    - Eğitim setindeki x’leri ile birlikte kullanarak gerçek y değerine mümkün olduğu kadar yakın değer elde etmeye çalışır.
* Bu durumu nasıl formülize ederiz?
  + En yakın değere ulaşmaya çalışmak aslında bir minimizasyon problemidir.
  + (hθ(x) - y)2ifadesinin minimize edilmesidir.
    - Yani her bir örnek için h(x) ile y arasındaki farkı minimize et.
  + Bu ifadeyi tüm eğitim seti için yazarsak



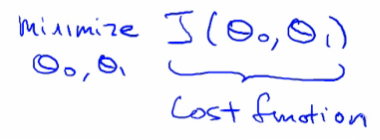
* Burada 1/2m:
  + 1/m ortalamayı aldığımızı söyler.
  + 1/2m buradaki 2 matematiksel işlemleri biraz daha kolaylaştırır ve bulduğumuz sabitleri değiştirmez.
* Bizim bu ifadeyi θ0/θ1 üzerinden minimize etmemiz gerekiyor.
  + Yani hipotez fonksiyonunda yerine konulduklarında ortalamada y değerine en yakın değerleri elde etmeliyiz.
* Bu bir cost fonksiyonudur.



* Bu cost fonksiyonunu minimize etmek istiyoruz.
* **Hipotez:** Tahmin makinesi, bir x değeri ver ve bir y değeri al.



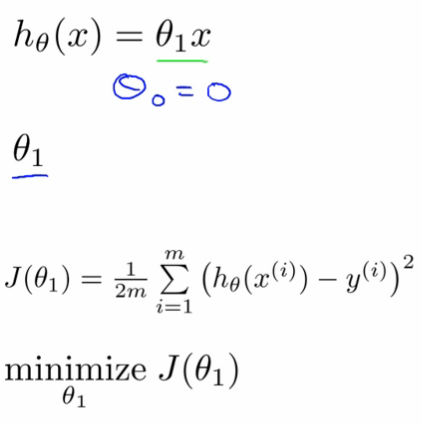
* **Cost:** Eğitim seti verilerini kullanarak, hipotezi mümkün olduğu kadar doğru yapacak teta değerlerini belirlemek için bir yol.



* + Bu cost fonksiyonu birçok regresyon fonksiyonu için uygun bir seçimdir.
  + Muhtemelen en yaygın kullanılandır.

**Cost Fonksiyonunu biraz daha yakından inceleyelim:**

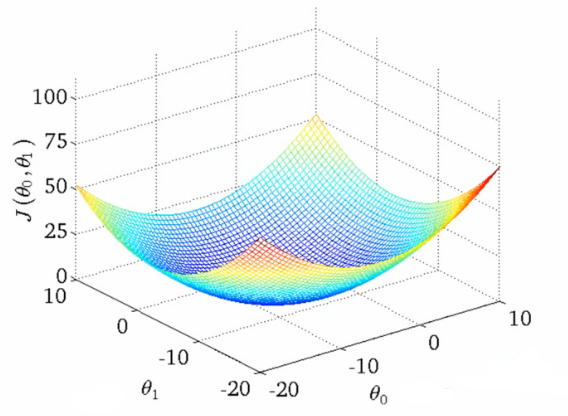
* Cost fonksiyonunu neden kullanıyoruz?
  + Parametreleri belirler.
  + Parametrelerle elde edilen değer hipotezin nasıl davranacağını belirler. Farklı parametreler farklı eğriler oluşturur.
* Basitleştirilmiş hipotez: Bu yolla J() cost fonksiyonunu biraz daha iyi anlamak mümkün



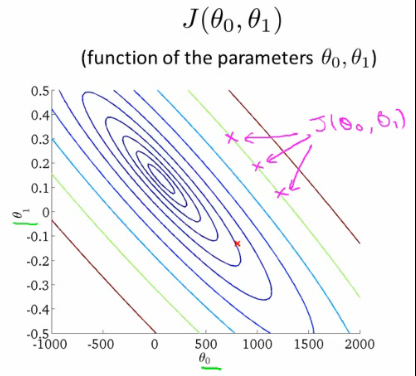
* Buna göre cost fonksiyonu başlangıç noktasından (0,0) geçer.
* Burada anlamamız gereken iki temel fonksiyon var:
  + hθ(x)
  + J(θ1)
* Örneğin:
  + θ1 = 1 olsun.
  + J(θ1)=0 olur.
* Bir öğrenme algoritmasının optimizasyon hedefi J(θ1) fonksiyonunu minimize eden θ1 değerini bulmaktır.
  + Bu örnekte θ1=1 en iyi değerdir.

Cost Fonksiyonuna Yakın Bakış:

* Yukarıdaki örnekte kullandığımız cost fonksiyonunu, hipotezi ve hedefi yine kullanalım.
* Bu kez hipotezimiz iki değişkenli olsun
  + Bu durumda cost fonksiyonu:
    - J(θ0, θ1) olur.
  + Örneğin:
    - θ0 = 50
    - θ1 = 0.06
  + Önceki örnekte θ1 vs J(θ1) grafiğini çizmiştik. Tek parametre vardı
  + Şimdiyse iki parametremiz var.
  + Bu durumda 3D yüzey grafiği elde edilir. Eksenler:
    - X = θ1
    - Z = θ0
    - Y = J(θ0,θ1)



* Bu grafikte yükseklik (y) cost fonksiyonunun değerini verir.
* Yüzey grafiği yerine contour grafiği kullanabiliriz.



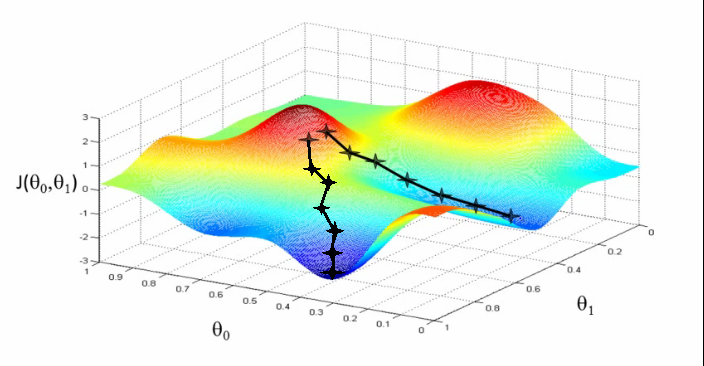
* Farklı renklerde elipslerden oluşur.
* Her renk aynı J(θ0,θ1) değerine sahiptir ancak θ0, θ1 faklı değerler alabileceğinden farklı konumlar olacaktır.
* Ekrandan dışarı doğru çıkan bir tas gibi düşünebilirsiniz.
* Her nokta (kırmızı) θ0 ve θ1 için bir çift parametre değerini temsil eder.
  + Bu örnekte:
    - θ0 = ~800
    - θ1 = ~-0.15
    - bu iyi bir fit değildir.
  + Eğer:
    - θ0 = ~360
    - θ1 = 0
    - daha iyi bir hipotez verir ancak yine de tam bir fit değildir.
  + Nihayetinde contour grafiğinin tam orta noktası minimum değeri dolayısıyla en iyi hipotezi verir.
* Bu işlemleri otomatik olarak yapacak verimli bir algoritma arıyoruz.

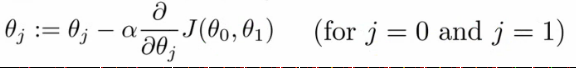
**GRADIENT DESCENT ALGORİTMASI**

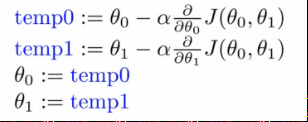
* Hedefi cost fonksiyonu J() yi minimize etmektir.
* Makine öğrenmesi optimizasyon problemlerinde yaygın olarak kullanılmıştır.
* İşe genel bir J() fonksiyonu arayarak başlar.
* Problemimiz:
  + Elimizde J(θ0, θ1) var.
  + min J(θ0, θ1) bulmayı istiyoruz.
* Gradient descent daha genel fonksiyonlara uygulanır. Örneğin.
  + J(θ0, θ1, θ2 .... θn)
  + min J(θ0, θ1, θ2 .... θn)

**Nasıl Çalışır:**

* Başlangıçta ilk tahminleri yaparak işe başlar.
  + (0,0) ya da başka bir noktadan başlar
  + θ0, θ1 değerlerini azar azar değiştirerek J(θ0, θ1) değerini küçültmeye çalışır.
* Her parametre değiştirildiğinde, J(θ0,θ1) değerini mümkün olan en fazla düşüren gradient değeri seçilmiş olur.
* Bu adımı tekrarla.
* Local bir minimuma ulaşana kadar devam et.
* Bu algoritmanın ilginç bir özelliği vardır.
  + Başlangıç noktanız elde edeceğiniz minimumu belirleyebilir.



* Converge olana kadar aşağıdakini yap
  + 
* Anlamı?
  + θj değerini güncelle
* α (alpha)
  + learning rate(öğrenme hızı)
  + adım büyüklüğünü kontrol eder.
    - α büyükse gradient descent agressive
    - α küçükse adım büyüklüğü küçüktür.
* Algoritmanın uygulanmasında önemli bir nokta:
  + j=0 ve j=1 demek her iki (bu örnek için) parametrenin de aynı anda güncellenmesi demektir.
  + Yani



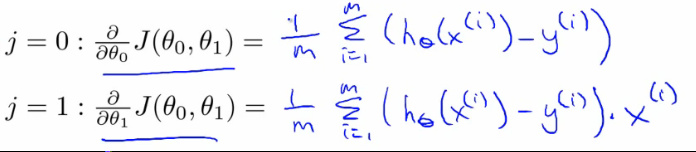
* İki anahtar terim var.
  + Alfa
  + Kısmi türev.
* Kısmi türev:
  + ilgili noktadan bir teğet alıp bu teğetin eğimini bulmak.
  + Aşağıya doğru inildiği için negatif eğim oluşacak.
* Lokal minimuma ulaşıldığında
  + Teğetin türevi sıfırdır. (J() türevi sıfırdır)

**LINEAR REGRESSION WITH GRADIENT DESCENT**

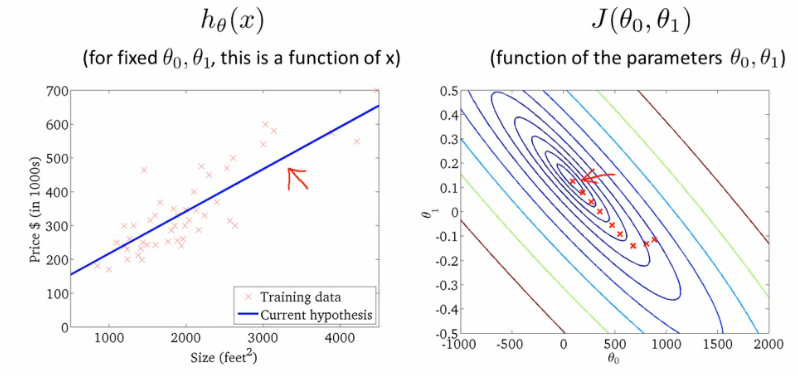
* J(θ0, θ1) cost fonksiyonunu minimize etmek için gradient descent.
* Bu durumda kısmı türev oluşacaktır.



* Her parametre için türevi belirlemeliyiz.
  + j=0
  + j=1
* θ0 ve θ1 için bu türevin ne olduğunu bulalım.
  + Bu ifade j=0 ve j=1 için elde edilirse:



* Şimdi bu değerler gradient descent algoritmasında kullanılabilir.
* Nasıl çalışır?
  + Gradient descent ile farklı lokal optimumlarla karşılaşma riski vardır.
  + Ancak linear regression fonksiyonu daima konveks bir fonksiyondur. Yani daima tek bir minimumu vardır.
    - Çukur tabak şeklindedir.
    - Tek bir global optimum
      * Bu durumda gradient descent daima global optimuma ulaşacaktır.
  + Çalışırken:
    - Başlangıç değerleri:
    - θ0 = 900
    - θ1 = -0.1



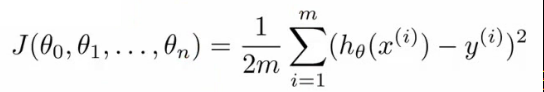
* Global mimimuma ulaşır.
* Aslında bu Batch Gradient Descent olarak adlandırılır.
  + Yani her adımda tüm eğitim verilerine bakılır.
    - Her adım m eğitim verisi için hesaplanır.
  + Bazı durumlarda non-batch versiyonu da mevcuttur. Eğitim verisinin küçük bir kısmı kullanılır (m çok büyükse)

**ÇOK DEĞİŞKENLİ LINEAR REGRESSION**

* Çok değişken demek birden fazla özellik demektir.
* Daha önce:
  + X= ev büyüklüğünü kullanarak
  + y= ev fiyatlarını tahmin ettik.
* Şimdi daha fazla özelliği dikkate alırsak (oda sayısı, kat sayısı, evin yaşı…)
  + Dört özellik x1, x2, x3,x4
    - x1- evin büyüklüğü
    - x2- oda sayısı
    - x3- kat sayısı
    - x4- evin yaşı
  + y output (çıktı) değişkeni (fiyat)
* notasyon:
  + n- özelliklerin sayısı (4)
  + m- örnek sayısı(tablodaki satır sayısı)
  + xi- bir örneğe ait vektör
    - x, n boyutlu bir özellik vektörü
    - örneğin x3 üçüncü evdir ve bu eve ait dört özelliği de içerir.
  + xji- i. eğitim örneğinin j özelliğinin değeri.
    - x23 -üçüncü evdeki oda sayısı.
* Şimdi birden fazla özelliğimiz olduğuna göre:
  + Hipotezimiz hangi formda olmalı?
  + Daha önce hipotezimiz: hθ(x) = θ0 + θ1x
    - İki parametre
    - Bir değişken
  + Şimdi çok değişkenimiz var:
    - hθ(x) = θ0 + θ1x1 + θ2x2 + θ3x3 + θ4x4
    - örneğin: hθ(x) = 80 + 0.1x1 + 0.01x2 + 3x3 - 2x4
      * ev fiyatlarını tahmin etmeye çalışan bir hipotez.
      * Parametreler yine cost fonksiyonu üzerinden hesaplanır.
  + Notasyonda kolaylık olması için: x0 =1 alınır.
    - Yani her i. örnek için bir 0. özellik vardır.
    - Bu durumda özellik vektörü n+1 boyutlu bir vektör olur.
      * Bu sütun vektörüdür ve x olarak adlandırılır.
      * Her örneğe ait bir x vektörü vardır.
    - Parametreler de yine 0 indeksten başlan ve n+1 boyutlu vektördür.
      * Bu da bir sütun vektörüdür ve θ olarak adlandırılır.
      * Bu vektör her örnek için aynıdır.
  + Bunları göz önüne alarak hipotez :
    - hθ(x) = θ0x0 + θ1x1 + θ2x2 + θ3x3 + θ4x4 şeklinde yazılır
  + Bu ifade:
    - hθ(x) =θ*T* X şekilde yazılabilir.
    - θ*T* [1xn+1] bir matristir.
    - Θ bir sütun vektörü olduğuna göre transpozisyon operasyonu onu bir satır vektörüne dönüştürür.
    - Önce:
      * θ*T*, [n+1x1] matristi
    - Şimdi,
      * θ*T*, [1xn+1] matris.
    - Yani θ*T* ve X matrislerinin boyutları uyuşuyor. Bu durumda
      * hθ(x)=[1xn+1]\*[n+1x1]
      * başka bir değişle:
        + parametre vektörünün transpozunun bir girdi örneğinin X vektörüyle çarpımı tahmin edilen hipotezi verir. Bu [1x1] boyutludur. Yani tek bir değerdir.

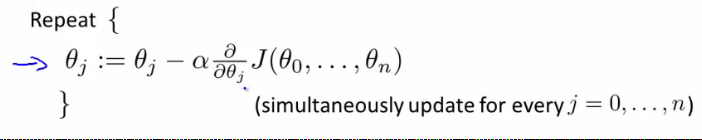
**Çok Değişkenli Gradient Descent**

* Parametreleri θ0 dan θn kadar almak yerine tek bir vektör θ olarak alınır.
* Cost fonksiyonu J(θ)

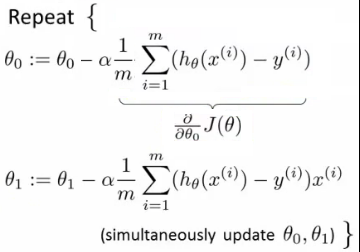


J(θ)

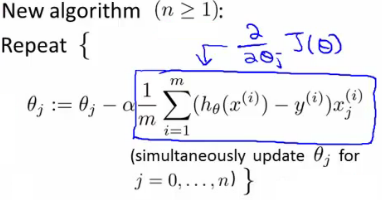
* Gradient descent: J(θ)



* Bu algoritmayı uygularsak:
  + n=1 olsun



* + n>=1:

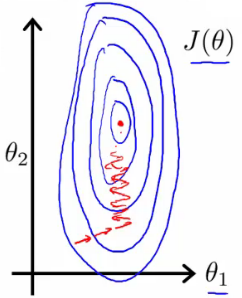


unutmayın

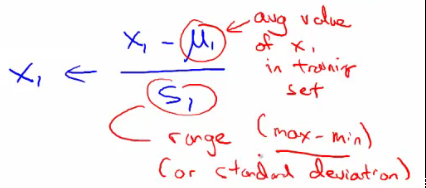
* Bu işlem her bir j için (0-n arası) aynı anda güncelleme işlemi olarak gerçekleştirilir.

**Gradient Descent-Özellik Ölçeklendirmesi**

* Birden fazla özelliği olan bir problemle uğraşıyorsanız
  + Özellik değerlerinin benzer ölçekte olmasına dikkat edin.
    - Daha hızlı converge olur.
* Örneğin:
  + x1=büyüklük (0-2000)
  + x2= oda sayısı (1-5)
  + Her iki özellik arasındaki ölçek farkı çok büyük.

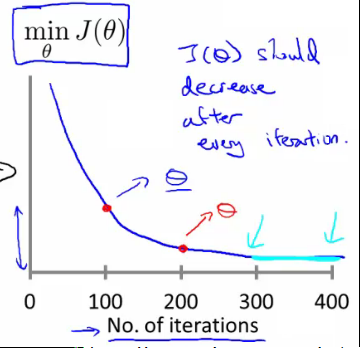


* Bu durumda gradient descent osilasyona girerek global minimuma daha geç ulaşır.
* Bunu engellemek için ölçeklendirme yapmak gerek:
  + x1=büyüklük/2000, x2=oda sayısı/5
  + bunun sonucunda kontur grafiği daha dairesel hale gelir.
  + Genel olarak özellik değeri -3<x<3 aralığına ölçeklendirilebilir.
* Farklı bir ölçeklendirme yöntemi:
  + Her bir xi değeri yerine aşağıdaki ifadeden gelen xi değeri kullanılabilir.

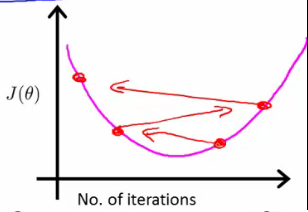


**Gradient Descent-Öğrenme Hızı(α)**

* Öğrenme hızı gradient descentin çalışmasını etkiler.
* Nasıl anlaşılır?
* J(θ) nın iterasyon sayısına göre grafiği çizilir.



* J(θ) azalıyorsa--- gradient descent çalışıyor.
* Mavi aralıkta J(θ) değişmiyor. Convergence gerçekleşmiş
* Eğer J(θ) artıyorsa---- bunun anlamı daha küçük bir alfa değerine gerek olduğudur.
* Alfa büyük olduğu için J(θ) şekildeki gibi osilasyon yapar.

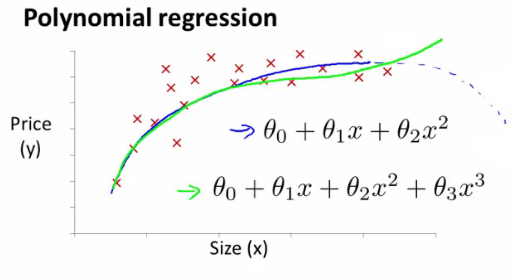




* Bir başka problem J(θ) nın dalgalı çıkması.
  + Daha küçük alfa değeri kullanılmalı.

**Özellikler ve Polinomial Regression**

* Özelliklerin seçimi?
* Örnek: Ev fiyatları.
  + İki özellik olsun: Evin alanına ait iki değer var. Arsanın genişliği (x1) ve uzunluğu (x2)
  + Bu iki özelliği mutlaka kullanmak zorunda değiliz.
  + Bunlardan yeni bir özellik oluşturabilirsiniz.
    - Örneğin arsanın alanı (x)=x1\*x2
    - h(x) = θ0 + θ1x



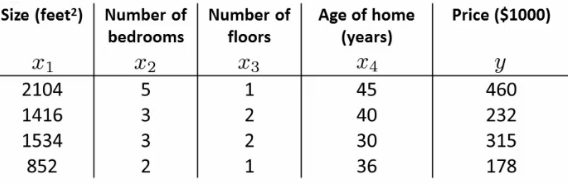
Mavi: kuadratik fonksiyon

Yeşil: kübik fonksiyon

**Normal Denklemi**

* Bazı durumlarda normal denklemi linear regresyon problemlerinde daha iyi sonuç verir.
* Normal denklemi tetayı analitik olarak çözer.
* Nasıl çalışır:
* Cost fonksiyonu J(θ) = aθ2+ bθ + c
* Teta gerçek bir sayı, vektör değil.
* Bu fonsiyon nasıl minimize edilir?
* 0
* Daha kompleks problemlerde: Teta n+1 boyutlu bir vektör.
* J(θ), vektöre bağlı bir fonksiyon.
* Bu fonksiyonu minimize etmek için: tüm teta değerleri için J(θ) nın türevi alınır.

Örnek:



X0

1 1

1 1

* X tasarım matrisi oluşturulur:

1 2104 5 1 45

1 1416 3 2 40

1 1534 3 2 30

1 852 2 1 36

* y matrisi oluşturulur.

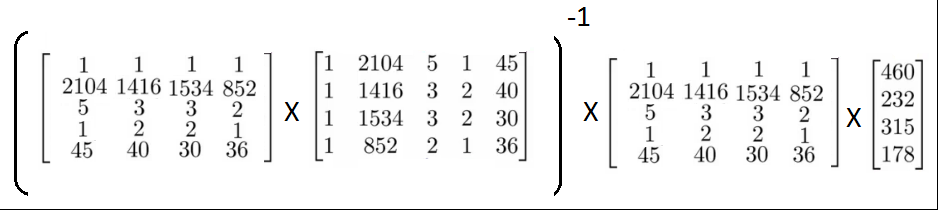
460

232

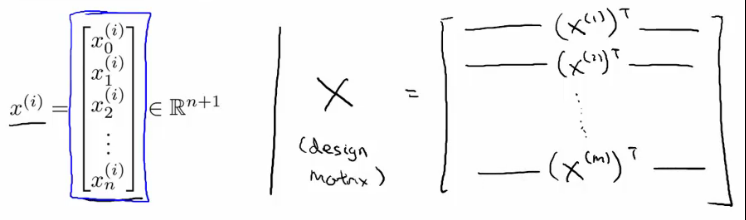
315

178

 ile teta hesaplanır



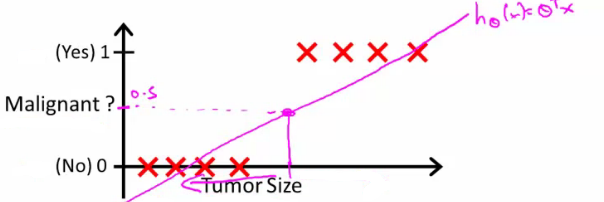
Genel hali



**Logistic Regression:**

**Sınıflandırma:**  y değeri kesikli bir değerdir.

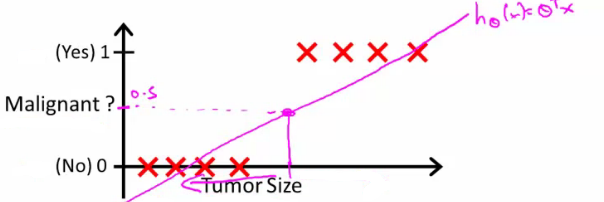
* Logistik regression algoritmasıyla y değerinin hangi grubun içinde olduğu (hangi sınıfa ait olduğu) bulunur.
* Sınıflandırma problemlerinde
  + Y = 0,1
  + 0: ilgili sınıf içinde yok
  + 1: ilgili sınıfa dahil.
* İkili sınıflandırma problemi:
* Sınıflandırma algoritması nasıl geliştirilir?
  + Tümör boyutu-kötü huylu olması.
  + Burada linear regression kullanabiliriz!!!
    - hθ(x) = (θ*T* x)



* Eşik değeri belirlenir: 0.5------- hθ(x) >=0.5 ise y=1

hθ(x) <0.5 ise y=0

* Linear regression bu örnek için doğru bir sınıflandırma yapıyor.

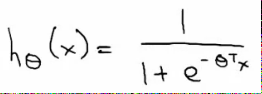


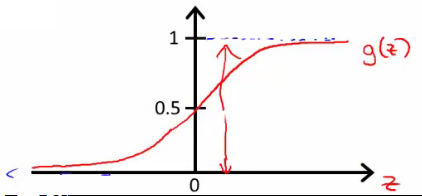
Logistic regression kullanılır:

**Hipotez Temsili**

* Sınıflandırmada hipotezi temsil için nasıl bir fonksiyon kullanacağız?
  + Daima 0-1 arası değerler istiyoruz.
* Bu amaçla hθ(x) = g((θ*T* x))
* g(z), z gerçek bir sayı.

g(z) = 1/(1 + e*-z*) logistic fonk., sigmoid fonk.



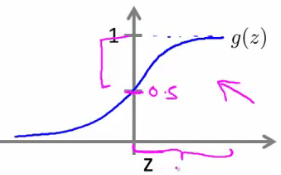


* Eğrinin değerlendirilmesi: Örnek:
* X özellik veötörü:
  + X= x0=1, x1=tümör boyutu X=
  + Diyelimki hθ(x)=0.7
    - Tümörün kanser olma ihtimali %70
    - hθ(x) = P(y=1|x ; θ) verilen x değerine karşın y’nin 1 olma ihtimali
  + ikili sınıflandırma problemi olduğuna göre:
    - P(y=1|x ; θ) + P(y=0|x ; θ) = 1
    - P(y=0|x ; θ) = 1 - P(y=1|x ; θ)

**Desicion Baundary (Karar Sınırı)**

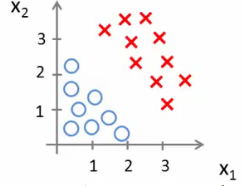
Hipotez fonksiyonun ne hesapladığının daha iyi anlaşılmasını sağlar.

* Sigmoid fonksiyonun kullanım şekillerinden birisi:
  + y nin 1’den büyük olma ihtimali 0.5’ten daha büyük olduğunda y=1 alabiliriz
  + Aksi durumda y=0 dır.
* hθ(x) tam olarak 0.5 ten ne zaman büyük olur?
  + Sigmoid fonksiyonuna bakalım:
  + z>=0 olduğunda g(z)>=0.5
  + Bu durumda z>0 ise g(z)>=0,5 diyebiliriz.



* + z = (θT x)>=0 olduğunda hθ >=0.5 dir.
  + Buna göre hipotez θT x >=0 olduğunda y=1 değerini yahmin etmektedir.
  + Benzer şekilde hipotez θT x <0 olduğunda y=0 değerini tahmin etmektedir.
  + Diyelim ki

hθ(x) = g(θ0 + θ1x1+ θ2x2) verilsin.



* θ0 = -3
* θ1 = 1
* θ2 = 1

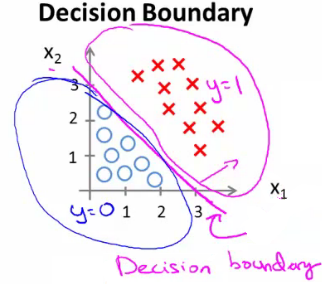
Olsun. Parametre vektörü θ=-3;1;1

θT=-3 1 1

z = (θT x)

Eğer -3x0 + 1x1 + 1x2 >= 0 veya -3 + x1 + x2 >= 0 ise y=1

x1+x2>=3 denilir ve bu ifadenin grafiği çizilirse:



Karar sınırını veren doğru hθ değerinin tam olarak 0.5 olduğu noktalar kümesidir.

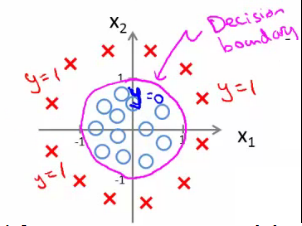
Karar sınırı hipotezin bir özelliğidir. Veri setinin özelliği değildir.

NON\_LINEAR Decision boundary:

Bu durumda hθ(x) = g(θ0 + θ1x1+ θ3x12 + θ4x22) gibi alıyoruz.

θT=-1 0 0 1 1 olsun:

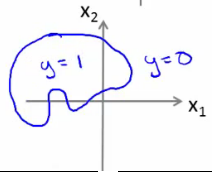
-1 + x12 + x22 >= 0 veya x12 + x22 >= 1 ise y=1 olmalı.



Daha karmaşık karar sınırı elde edilebilir mi?

Daha yüksek dereceden polinom terimleri kullanarak elde edebilirsiniz

hθ(x)=g(θ0 + θ1x1+ θ2x2 + θ3x12+θ4x12x2+θ5x12x22+θ6x13x2+...)

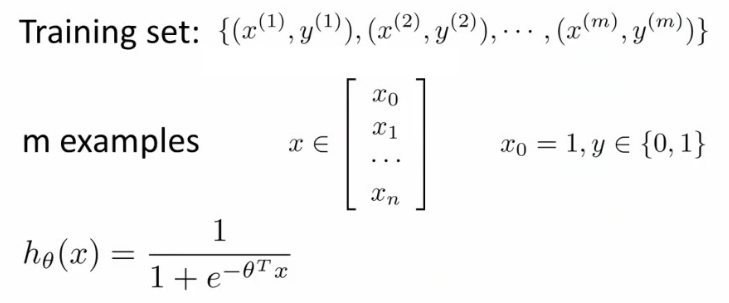


LOGISTIC REGRESYON COST FONKSİYONU

Teta parametreleri logistic regresyon da nasıl fit edilecek?

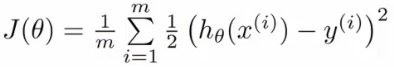
m örnekten oluşan bir eğitim seti olsun.

Her bir örneğin n+1 uzunluğunda sütun vektörü var.



Bu duruma göre tetayı nasıl seçiyoruz?

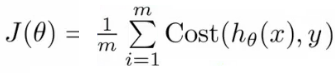
Linear regresyon tetayı bulurken şu denklemi kullanıyor



Bu ifadede hata hesabının yapıldığı üslü ifadenin yerine yazılmak üzere bir cost() fonksiyonu tanımlarız.



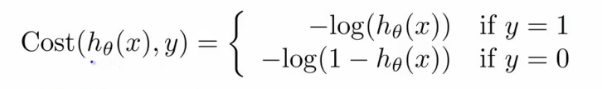
Ifadeyi basitleştirmek için indisler kaldırılabilir.

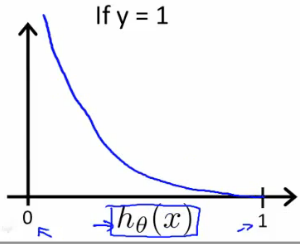


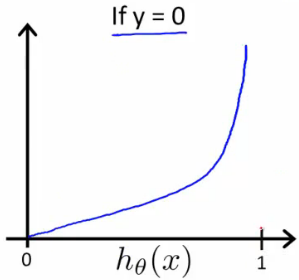
Algoritmanın hesapladığı sonuç hθ(x) ve gerçek sonuçta y olduğunda algoritmanın karşılaması gereken maliyet

Bu fonksiyon non-convex bir fonksiyon ve logistic regresyon da kullanılırsa çalışabilir.

Konvex logistic regresyon foksiyonu

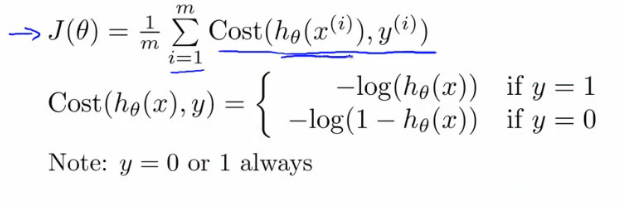






BASİTLEŞTİRİLMİŞ COST FONKSİYONU VE GRADIENT DESCENT

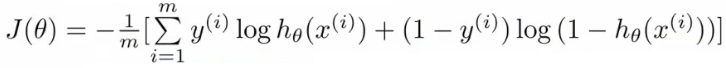
Logistic regresyon cost fonksiyonu:



y nin sadece iki değer alabilmesinden dolayı Cost() fonksiyonunu iki satır değil tek bir satırda yazabiliriz.

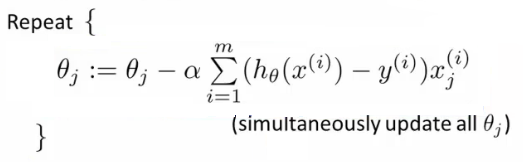
**cost(hθ,(x),y) = -ylog( hθ(x) ) - (1-y)log( 1- hθ(x) )**

Cost fonksiyonu:



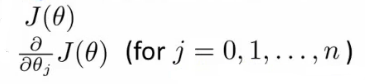
J(theta) yı minimize eden teta değerlerini bulmak için gradient descent kullanılabilir:





**İleri Optimizasyon**

* Daha önce cost fonksiyonunu minimize etmek için gradient descent kullandık
* Şimdi logistic regresyon cost fonksiyonunu minimize etmek için daha ileri kavramalara bakalım.
  + Bunlar büyük çaplı makine öğrenmesi problemleri için (çok fazla sayıda özelliğin yer aldığı data setlerinde) kullanışlıdır.
* *Garident descent aslında ne yapar?*
  + Cost fonksiyonu J(θ), bunu minimize etmek istiyoruz
  + θ değerini input olarak alan ve aşağıdaki ifadeleri hesaplayan bir koda gerek var.



* Kod bu iki ifadeyi hesaplarken, Gradient descent sürekli aşağıki güncellemeyi yapar.

C:\C9295485\61F5C22D-B693-4A5E-ACB6-3D06409C113C_dosyalar\image021.png

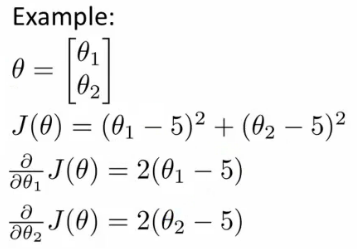
* Buna göre;
  + J(θ) ve türevini hesaplayacak kod
  + Sonra bu değerleri gadient descent te yerine koymak

gerekiyor.

* Gradient descente alternatif olarak kullanılabilecek yöntemeler.
  + **Conjugate gradient**
  + **BFGS** (Broyden-Fletcher-Goldfarb-Shanno)
  + **L-BFGS** (Limited memory - BFGS)
* Bunlar daha optimize algoritmalardır
* Daha karmaşıktırlar
* bazı özellikleri
  + **Avantajları** 
    - Alfa (öğrenme hızı) değerini elle seçmeye gerek yok
      * Bir grup alfa değerini deneyerek iyi olanı deçen bir döngü kullanırlar (line search algorithm)
    - Genellike gradient descentten daha hızlıdırlar
      * Yalnızca iyi bir öğrenme hızı seçmekten daha fazlasını yaparlar
    - Karmaşsını anlamaya gerek olmadan başarılı bir şekilde kullanılabilirler
  + **Dezavantajları**
    - Degbugging daha zor olabilir
    - Kendiniz yazmaya kalkışmayın. (Nümerik konusunda uzman olmadıkça)
    - Farklı kütüphaneler, farklı uygulama yöntemleri kullanabilir. Performans etkilenebilir

**İler cost minimizasyon algoritmalarını kullanmak**

* Bu algoritmalar nasıl kulllanılır
  + Diyelim ki elimizde aşağıdaki gibi bir örnek var.



Function [jval,gradient]=costFunction(theta)

jval=(theta(1)-5)^2+(theta(2)-5)^2;

gradient=zeros(2,1);

gradient1=2\*(theta(1)-5);

gradient2=2\*(theta(2)-5);

* Bu örnekte
  + θ1 ve θ2 (iki parametre)
  + Cost fonksiyonu J(θ) =(θ1- 5)2 + (θ2- 5)2
  + J(θ) nın θ1 ve θ2 türevleri 2(θi - 5)
* İlk önce cost fonksiyonunu tanımlamamız gerek

**function [jval, gradient] = costFunction(THETA)**

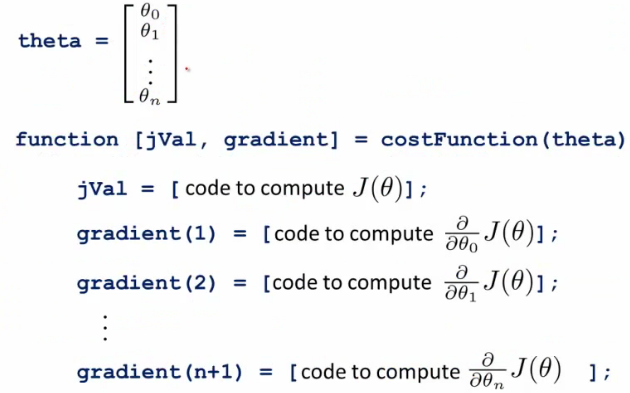
* Cost fonksiyonu için input **THETA**, θ parametrelerinin bir vektörüdür
* **costFunction** iki değer döndürür
  + **jval**
    - Cost fonksiyonunu nasıl hesaplarız
      * Bu örnek için =(θ1 – 5)2 + (θ2 - 5)2
  + **gradient**
    - 2 x 1 vektor
    - 2 eleman iki kısmi türevden gelir.
    - Yani bu n boyutlu bir vektördür
* Cost fonksiyonu belirlendikten sonra ileri algoritmayı aşağıdaki gibi çağırabiliriz

**options**= optimset('GradObj', 'on', 'MaxIter', '100'); **% define the options data structure**

**initialTheta**= zeros(2,1); # set the initial dimensions for theta **% initialize the theta values**

**[optTheta, funtionVal, exitFlag]**= fminunc(@costFunction, initialTheta, options); **% run the algorithm**

* Burada
  + **options** algoritma için seçenekler veren bir veri yapısı
  + **fminunc**
    - cost fonksiyonunu minimize eden fonksiyon (**f**ind **min**imum of **unc**onstrained multivariable function)
  + **@costFunction,** costFunction function için işaretçi
* Octave uygulaması için
  + **initialTheta,** en az iki boyutlu bir matris olmalı
* Logistic regresyona bunu nasıl uygularız ?
  + Elimizdekiler



* Burada
  + theta n+1 boyutlu sütun vektörü
  + Octave indekslemesi 0 dan değil 1 den başlar

**Çok sınıflı sınıflandırma problemleri**

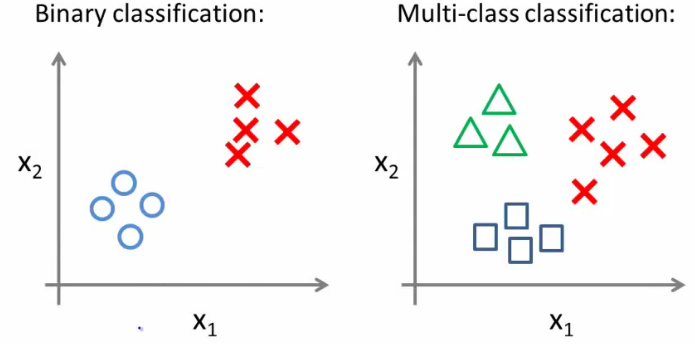
* Çok sınıflı sınıflandıma problemlerinde logistic regresyon kullanmak: **one vs. all**
* Multiclass – evet ya da hayırdan (1 ve 0) daha fazlasını içerir
* Örneğin:
  + email foldering/tagging: work, friends, family, hobby

y=1 y=2 y=3 y=4

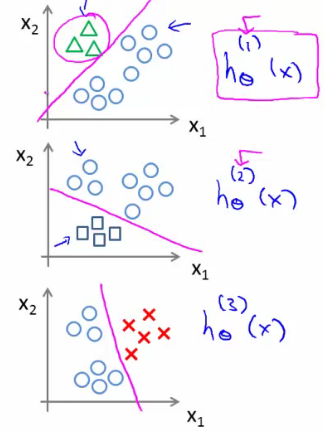
* + medical diagrams: not ill, cold, flu

y=1 y=2 y=3

* + Weather: sunny, cloudy, rainy snow
    - * y=1 2 3 4



* Üç sınıfı olan bir veri setiyle öğrenim algoritmasını nasıl çalıştırırız?
  + one vs. all sınıflandırmasını kullanmak, ikili sınıflandırmanın multiclass sınıflandırma için de çalışmasını sağlar
* **One vs. all sınıflandırması**
  + Eğitim seetini üç ayrı binary sınıflandırma problemini bölün



* **Genelde**
  + **hθ(i)(x)= P(y=i |x; θ) (i=1,2,3)**
  + Train a logistic regression classifier hθ(i)(x) for each class i to predict the probability that y=i
  + On a new input, *x* to make a prediction, pick the class *i* that maximizes the probability that hθ(i)(x) = 1
  + max hθ(i)(x)

<[*http://www.holehouse.org/mlclass/06\_Logistic\_Regression.html*](http://www.holehouse.org/mlclass/06_Logistic_Regression.html)> kaynağından eklendi

07\_Regularization

11 Şubat 2017 Cumartesi

20:28

**07: Regularization**

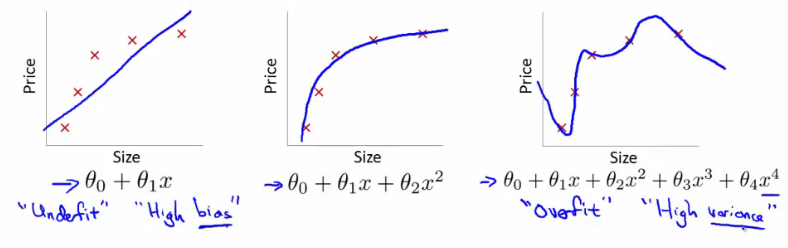
[Previous](http://www.holehouse.org/mlclass/06_Logistic_Regression.html) [Next](http://www.holehouse.org/mlclass/08_Neural_Networks_Representation.html) [Index](http://www.holehouse.org/mlclass/index.html)

**The problem of overfitting**

* So far we've seen a few algorithms - work well for many applications, but can suffer from the problem of overfitting
* What is overfitting?
* What is regularization and how does it help

**Overfitting with linear regression**

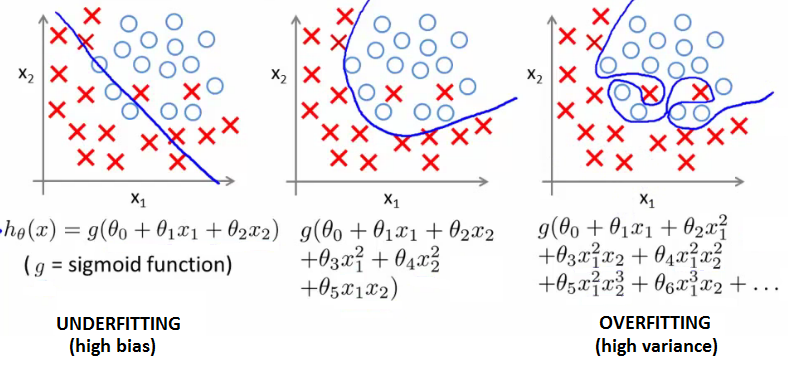
* Using our house pricing example again
  + Fit a linear function to the data - not a great model
    - This is **underfitting** - also known as **high bias**
    - Bias is a historic/technical one - if we're fitting a straight line to the data we have a strong preconception that there should be a linear fit
      * In this case, this is not correct, but a straight line can't help being straight!
  + Fit a quadratic function
    - Works well
  + Fit a 4th order polynomial
    - Now curve fit's through all five examples
      * Seems to do a good job fitting the training set
      * But, despite fitting the data we've provided very well, this is actually not such a good model
    - This is **overfitting** - also known as **high variance**
  + Algorithm has high variance
    - High variance - if fitting high order polynomial then the hypothesis can basically fit any data
    - Space of hypothesis is too large (çok değişken)



* To recap, if we have too many features then the learned hypothesis may give a cost function of exactly zero
  + But this tries too hard to fit the training set
  + Fails to provide a *general* solution - **unable to generalize** (apply to new examples)

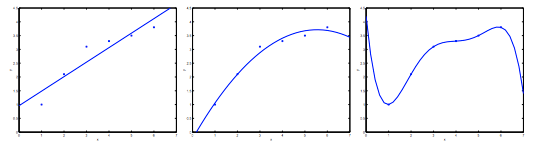
**Overfitting with logistic regression**

* Same thing can happen to logistic regression
  + Sigmoidal function is an underfit
  + But a high order polynomial gives and overfitting (high variance hypothesis)



**Addressing overfitting**

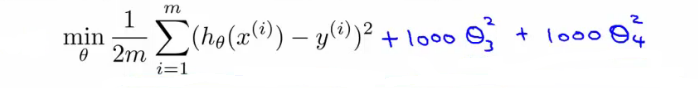
* Later we'll look at identifying when overfitting and underfitting is occurring
* Earlier we just plotted a higher order function - saw that it looks "too curvy"



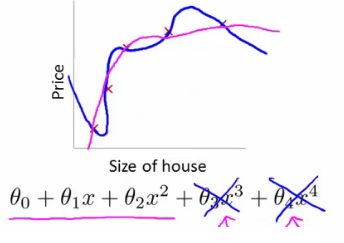
* + Plotting hypothesis is one way to decide, but doesn't always work
  + Often have lots of a features - here it's not just a case of selecting a degree polynomial, but also harder to plot the data and visualize to decide what features to keep and which to drop
  + If you have lots of features and little data - overfitting can be a problem
* How do we deal with this?
  + 1) **Reduce number of features**
    - Manually select which features to keep
    - Model selection algorithms are discussed later (good for reducing number of features)
    - But, in reducing the number of features we lose some information
      * Ideally select those features which minimize data loss, but even so, some info is lost
  + 2) **Regularization**
    - Keep all features, but reduce magnitude of parameters θ
    - Works well when we have a lot of features, each of which contributes a bit to predicting y

**Cost function optimization for regularization**

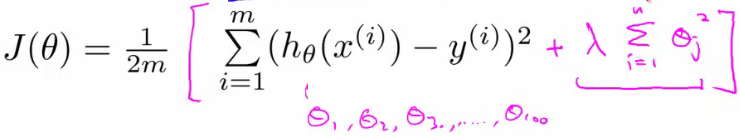
* Penalize and make some of the θ parameters really small
  + e.g. here θ3 and θ4



* The addition in blue is a modification of our cost function to help penalize θ3 and θ4
  + So here we end up with θ3 and θ4 being close to zero (because the constants are massive)
  + So we're basically left with a quadratic function



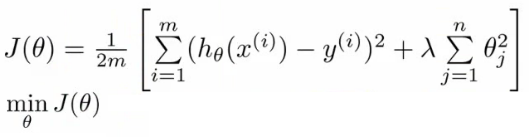
* In this example, we penalized two of the parameter values
  + More generally, regularization is as follows
* Regularization
  + Small values for parameters corresponds to a simpler hypothesis (you effectively get rid of some of the terms)
  + A simpler hypothesis is less prone to overfitting
* Another example
  + Have 100 features x1, x2, ..., x100
  + Unlike the polynomial example, we don't know what are the high order terms
    - How do we pick the ones to pick to shrink?
  + With regularization, take cost function and modify it to shrink all the parameters
    - Add a term at the end
      * This regularization term shrinks every parameter
      * By convention you don't penalize θ0 - minimization is from θ1 onwards



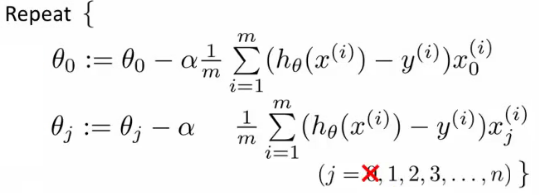
* In practice, if you include θ0 has little impact
* **λ**is the **regularization parameter**
  + Controls a trade off between our two goals
    - 1) Want to fit the training set well
    - 2) Want to keep parameters small
* With our example, using the **regularized objective** (i.e. the cost function with the regularization term) you get a much smoother curve which fits the data and gives a much better hypothesis
  + If **λ** is very large we end up penalizing ALL the parameters (θ1, θ2 etc.) so all the parameters end up being close to zero
    - If this happens, it's like we got rid of all the terms in the hypothesis
      * This results here is then underfitting
    - So this hypothesis is too biased because of the absence of any parameters (effectively)
* So, **λ**should be chosen carefully - not too big...
  + We look at some automatic ways to select **λ**later in the course

**Regularized linear regression**

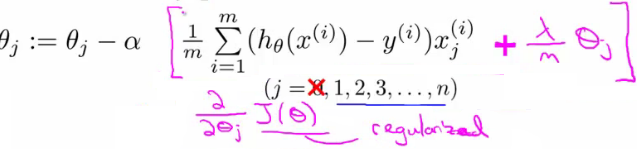
* Previously, we looked at two algorithms for linear regression
  + Gradient descent
  + Normal equation
* Our linear regression with regularization is shown below



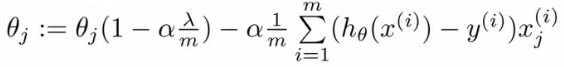
* Previously, gradient descent would repeatedly update the parameters θj, where j = 0,1,2...n simultaneously
  + Shown below



* We've got the θ0 update here shown explicitly
  + This is because for regularization we don't penalize θ0so treat it slightly differently
* How do we regularize these two rules?
  + Take the term and add λ/m \* θj
    - Sum for every θ (i.e. j = 0 to n)
  + This gives regularization for gradient descent
* We can show using calculus that the equation given below is the partial derivative of the regularized J(θ)



* The update for θj
  + θj gets updated to
    - θj- α \* [a big term which also depends on θj]
* So if you group the θjterms together



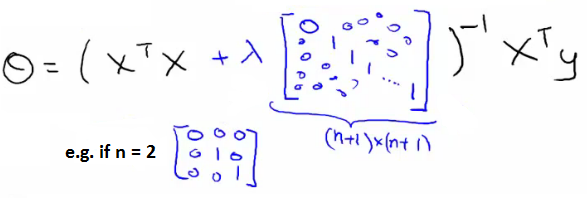
* The term

C:\63898A45\3BF83D1C-FC48-45FD-83A1-7079774071A1_dosyalar\image010.png

* Is going to be a number less than 1 usually
* Usually learning rate is small and m is large
  + So this typically evaluates to (1 - a small number)
  + So the term is often around 0.99 to 0.95
* This in effect means θjgets multiplied by 0.99
  + Means the squared norm of θja little smaller
  + The second term is exactly the same as the original gradient descent

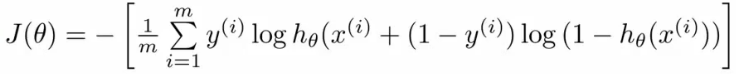
**Regularization with the normal equation**

* Normal equation is the other linear regression model
  + Minimize the J(θ) using the normal equation
  + To use regularization we add a term (+ λ [n+1 x n+1]) to the equation
    - [n+1 x n+1] is the n+1 identity matrix

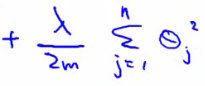


**Regularization for logistic regression**

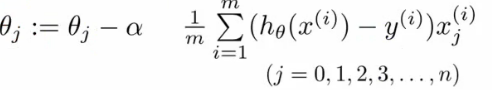
* We saw earlier that logistic regression can be prone to overfitting with lots of features
* Logistic regression cost function is as follows;



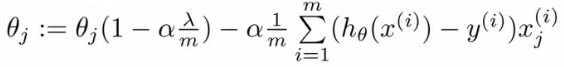
* To modify it we have to add an extra term



* This has the effect of penalizing the parameters θ1, θ2 up to θn
  + Means, like with linear regression, we can get what appears to be a better fitting lower order hypothesis
* How do we implement this?
  + Original logistic regression with gradient descent function was as follows

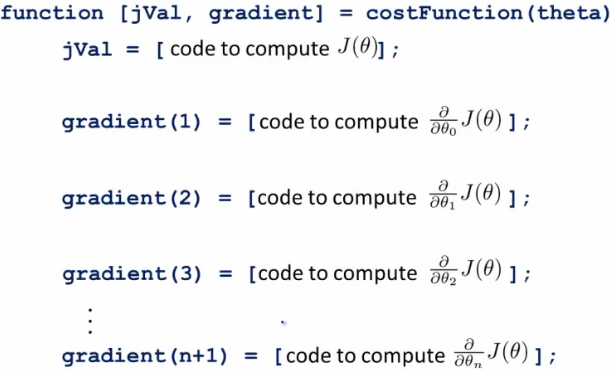


* Again, to modify the algorithm we simply need to modify the update rule for θ1, onwards
  + Looks cosmetically the same as linear regression, except obviously the hypothesis is very different

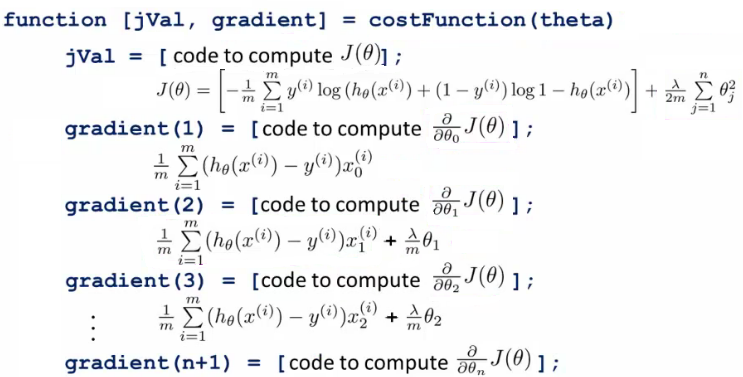


**Advanced optimization of regularized linear regression**

* As before, define a costFunction which takes a θ parameter and gives jVal and gradient back



* use **fminunc**
  + Pass it an **@costfunction** argument
  + Minimizes in an optimized manner using the cost function
* **jVal**
  + Need code to compute J(θ)
    - Need to include regularization term
* Gradient
  + Needs to be the partial derivative of J(θ) with respect to θi
  + Adding the appropriate term here is also necessary



* Ensure summation doesn't extend to to the lambda term!
  + It doesn't, but, you know, don't be daft!***http://www.holehouse.org/mlclass/07\_Regularization.html***