

Machine Learning

Supervised Learning (Denetimli Öğrenme): Bilgisayarın ne yapacağını öğretilmesi (Ör: Regresyon, Classification)

Unsupervised Learning (Denetimsiz Öğrenme): Bilgisayarın kendi kendine öğrenmesi (Ör: Cluster)

- Verdiğimiz tüm veri topluluğuna **veri seti** denir.

Regresyon (Gerileme)

- Burada amaç, bir bağımsız değişkene karşılık gelen sürekli çıktıyı doğru şekilde tahmin etmektir.
- Sürekli çıktı, belirli bir aralıkta sonsuz değer alabilen veridir.

Classification (Sınıflandırma)

- Bir ayrık değere sahip bir çıktı tahmin etmeye yönelik probleme denir. Genellikle binary olur fakat multi-class da olabilir.
- Kesikli çıktı, sınırlı ve ayrık (belirli ve sayılabilir) bir dizi değerden birini alan veridir.

Cluster (Kümeleme)

- Burada sadece dataset'ler belirlidir fakat sonuçları ve karşılıkları yoktur. Bu algortimadan bu data'ları kendi içerisinde gruplandırmasını bekliyoruz. Böylelikle yapay zekaya birşey öğretmeyip yapay zekanın kendisinin bir şeyler öğrenmesini beklemiş oluyoruz.
- Örnek olarak Google News verilebilir. Burada bir haber başlığı için çok sayıda haber makalesini bu başlık altında toplaması kümeleme algoritması kullanılarak yapılmıştır.

Lineer Regresyon

- Veri seti, öğrenim algoritmasına verilir. Bu algorithmadan bizim için en optimal fonksiyon çıkartmasını bekleriz. Böylelikle de en uygun tahminleri yapabilmesini istemiş oluruz. Yani girilen input için oluşturduğu fonksiyon ile en uygun çıktıyı vermesini bekleriz.
- Kabaca amacımızın data'yı düz bir çizgiye oturtmak olduğunu söyleyebiliriz.

$$h(x) = \theta_0 + \theta_1 \cdot x$$

h: Predict (Tahmin) function

x: input

$\theta(0)$: başlangıç değeri

$\theta(1)$: eğim

Maliyet Fonksiyonu (Cost Function)

- **Squared Error Function** da denir.
- Burada optimizasyon hedefimiz $J(\theta(0) + \theta(1))$ ifadesini minimize etmeye çalışmaktır.
- **Linear regresyondaki** maliyet fonksiyonunda **local minimum yoktur, sadece global minimum vardır.**

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m \left(h(x^{(i)}) - y^{(i)} \right)^2$$

J: Cost function ($\theta(0)$ ve $\theta(1)$ ' in minimize edilmesine bağlı)

m: total data (total training example)

h: Predict hypothesis (function)

y: gerçek değer

$\theta(0) = 0$ Olduğu Durumdaki Maliyet Fonksiyonu

- Burada $\theta(0) = 0$ olarak alınır ve formül buna göre işler.

$$h(x) = \theta_1 \cdot x$$

$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^m \left(h(x^{(i)}) - y^{(i)} \right)^2$$

- Buradan hareketle de optimizasyon hedefi, $J(\theta(1))$ ifadesini minimize etmek olur.

Gradient Descent (Kademeli Azaltma)

- Burada parametre sayısı 2'den fazla olabilir.
- " $:=$ " sembolü eşitlik değil de **assignment** (atama) anlamına gelir. (Ör: $a := b$ ifadesi "b'yi a'ya ata" olarak ifade edilir.)
- Burada parametrelerimizde, **yani $\theta(0)$ ve $\theta(1)$ 'de, eş zamanlı güncelleme yapmak** çok büyük önem arz etmektedir. Yani örneğin, yeni $\theta(0)$ değeri bulunduktan sonra $\theta(1)$ değerini bulmak için yeni $\theta(0)$ değerini değil eski $\theta(0)$ değerini kullanacağız. Her iki parametre için de bu güncelleme eş zamanlı olarak yapılmalı.

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \theta_2, \dots, \theta_n)$$

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} \left[\frac{1}{2m} \sum_{i=1}^m \left(h(x^{(i)}) - y^{(i)} \right)^2 \right]$$

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} \left[\frac{1}{2m} \sum_{i=1}^m \left(\theta_0 + \theta_1 \cdot x^{(i)} - y^{(i)} \right)^2 \right]$$

α : learning rate (öğrenme oranı)

- Bir dağın tepesinden indiğimizi düşünürsek eğer burada inerken attığımız adımın büyüklüğünü " α " değeri olarak ilişkilendirebiliriz.

$\partial / (\partial * \theta(j)) [J(\theta(0), \theta(1), \theta(2), \dots, \theta(n))]$: Parametrelerin türevi (fonksiyonda ne kadar parametre varsa)

- Bu terim, maliyet fonksiyonunun o parametre yönündeki değişim oranını veya eğimini gösterir.

j=0 için

$$\theta_0 := \theta_0 - \alpha \cdot \left(\frac{1}{m} \sum_{i=1}^m \left(h(x^{(i)}) - y^{(i)} \right) \right)$$

j=1 için

$$\theta_1 := \theta_1 - \alpha \cdot \left(\frac{1}{m} \sum_{i=1}^m \left(h(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)} \right)$$

Batch Gradient Descent (Toplu Kademeli Azaltma)

Bu yöntemde, her adımda tüm veri noktaları (m) kontrol edilir. Yani, tüm veri seti her iterasyonda değerlendirilir.

Matris ve Vektörler

- $A(ij)$ ifadesindeki i ifadesi **satır**, j ise **sütunu** temsil eder.
- Vektörler **tek sütundan oluşan matrislerdir**. Örneğin $Y(i)$ ifadesinde i vektördeki hangi satırdaki eleman olduğunu belirtir.
- Boyutları farklı olan matrisler **toplanamaz**.
- Matrislerde **çarpım işlemi** gerçekleşirken **ilk matrisin ilk satırı ile 2. matrisin ilk sütunu çarpılır ve çarpımlar toplanarak sonuç matrisinin ilk elemanı ortaya çıkmış olur**.
- Burada oluşan sonuç matrisinin satır sayısını **ilk matrisin satır sayısı, sütun sayısını da ikinci matrisin sütun sayısı belirler**.
- **Bir matrisle vektörün çarpım şartı matrisin satır sayısı ile vektörün sütun sayısının aynı olmasıdır**. Örneğin bir A matrisiyle bir x vektörü çarpılırken A matrisinin ilk satırı ile x vektörünün ilk elemanı çarpılır ve bu da y isimli bir vektörün ilk elemanı olmuş olur.
- **Matrislerde genel çarpım şartı ise ilk matrisin sütun sayısı ile ikinci matrisin satır sayısının aynı olmasıdır**. Aksi hâlde çarpma işlemi gerçekleştirilemez.
- Matrislerde cebirdeki çarpma işleminin özelliklerinden biri olan yer değiştirme özelliği **yoktur**. Matrislerde çarpma sırası önemlidir. (Ör: A ve B bir matris $A * B$

$\neq B * A$)

- Sadece **kare matrislerin**, yani satır sayısı ile sütun sayısı eşit olan matrislerin **tersi vardır**.
- **Bir matrisin transpozu** o matrisin satırlarının sütun, sütunlarının satır olarak yer değiştirmiş hâlidir diyebiliriz.

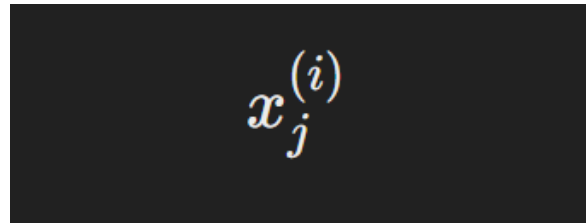
Çoklu Linner Regresyon

- n tane özelliği olan bir data'nın uygun çıktıyı vermesi amaçlanır.

n: Feature (Özellik) sayısı

$x^{(i)}$: i'inci training example.

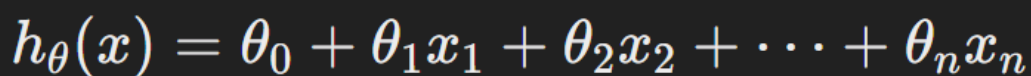
- Burada i ifadesini üs olarak değil de verimizdeki i'inci örnek olarak değerlendirmeliyiz. Mesela bir veri setinde 2 tane training example varsa 2.'yi belirtmek için $x^{(2)}$ şeklinde ifade ederiz.


$$x_j^{(i)}$$

$x^{(j)}$: i'inci trainig example'ın j'inci özelliği

- Burada j ifadesini x'in altına yazarız. j, x'in altına yazılır ve söz konusu training example'ın özelliğini temsil eder. (Ör: $x^{(2)}_3$ Bu ifade x veri setinin 3. training example'ının 2. özelliğini belirtir.)

- Bu regresyonda data sayısı arttığı için hipotez formülü değişir:


$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

- Bu ifade de aslında θ vektörünün transpozu ile x vektörünün çarpımıdır. Burada $\theta(0)$ değeri aslında $x(0)$ değerinin katsayısıdır fakat biz her zaman $x(0)$ değerini 1 olarak kabul ederiz.

Çoklu Gradient Descent

- Buradaki formül de şu şekilde güncellenecektir:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m \left(h_{\theta}(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

Burada bütün parametreler yukarıdaki formül kullanılarak güncellenecektir. Eş zamanlı olarak güncellemek önem atfeder.

Feature Scaling (Özellik Ölçeklendirme)

- Gradient Descent'in daha verimli çalışabilmesi için veri özelliklerinin aralık bandının birbirine daha yakın olması gerekir. Feature scaling ve mean normalization gibi yöntemler ile bunu sağlamak amaçlanır.

- Yan tarafta bir min-max normalizasyon türünde özellik ölçeklendirme formülü gösterilmiştir. Burada veri, maksimum ve minimum değerler arasında yeniden ölçeklendirilmiş olur.

$$x_{\text{scaled}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

$$\text{scaled value} = \frac{x}{\text{aralık farkı}}$$

- Bu tarafta ise scaling yani sadece veriyi aralığın genişliğine bölme işlemi yapılıyor. Burada veri aralığı sıkıştırılmaz, sadece sayıların daha küçük bir formata indirgenmesi sağlanır.

Mean Normalization (Ortalama Normalizasyonu)

Burada özelliği ortalama değerden çıkartıp aralık büyüklüğüne böldüğümüzde ortalama normalizasyon sağlanmış olur.

$$x_{\text{normalized}} = \frac{x - \mu}{x_{\text{max}} - x_{\text{min}}}$$

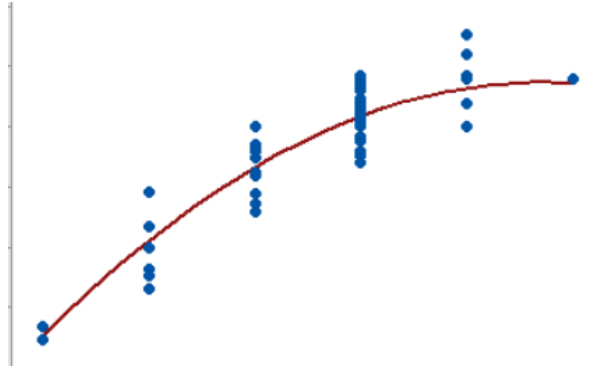
μ : Özelliğin ortalama değeri

Learning Rate Alfa (α)

- Gradient Descent algoritması maliyet fonksiyonu $J(\theta)$ minimize eden θ değerini bulabilmeyi amaçlıyordu.
- Eğer gradient descent düzgün çalışmıyorsa buradan alfa değerinin yeterince küçük olmadığını, küçültmemiz gerektiğini anlayabiliriz. Fakat buradaki küçüklük durumu da önem atfeder. Çünkü alfa değerimiz çok küçük olursa gradient descent algoritmamız çok fazla tekrar yapar, sonucunda algoritma istenen sonucu çok uzun süre sonra verebilir.
- Özetle alfa değeri çok küçük olursa yakınsama işlemi çok yavaş gerçekleşebilir ve eğer bu değer çok büyük olursa $J(\theta)$ minimize olmayabilir ve minimuma ulaşmayabilir.

Polinomal Regresyon

- Eldeki veri setini düz bir çizgi yerine polinomal bir çizgiyle karşılamamız mantıksal bağlamda daha uygun olduğu durumlarda kullanılır.
- Eğer burada 2. dereceden bir model oluşturulursa bu uygun olmayabilir. Örneğin x eksenini *size*, y eksenini *price* olsun. Buradaki



veriler de yandaki grafikteki gibi model üzerinde konumlanırsın. Bu durumda *size* artmaya devam ettikçe bir süre sonra *price* düşmeye başlayacaktır ve mantıksal olarak hatalıdır. Bu bağlamda 2. dereceden model oluşturmak yerine 3. derece bir model oluşturmayı tercih etmek mantıklı olacaktır.

$$h_{\theta}(x) = \theta_0 + \theta_1 \cdot \text{size} + \theta_2 \cdot \text{size}^2 + \theta_3 \cdot \text{size}^3$$

$$h_{\theta}(x) = \theta_0 + \theta_1 \cdot \text{size} + \theta_2 \cdot \sqrt{\text{size}}$$

- Ek olarak burada aslında sondaki *size* ifadesini karekök içerisine alsaydık da doğru bir yol izlemiş olurduk.
- Böyle kübik bir model kullanıldığı zaman *feature scaling* teriminin önemi de artacaktır. Gradient descent algoritması kullanmak istiyorsak buradaki *size* değerlerini birbirine yakınlaştırmak çok önemlidir.

Normal Equation (Normal Denklem)

- Gradient Descent algoritması ile sürekli tekrarlarla global minimum noktasına ulaşarak $J(\theta)$ 'nın minimizasyonu sağlanıyordu.
- Normal equation ile θ değeri analitik olarak çözümlene şansı yakalarız. Yani bu algoritma ile sürekli tekrarlara gerek kalmadan tek seferde optimum noktasına ulaşılabilir.
- Normalde 2. dereceden bir denklemin minimizasyonu yapılırken denklemin türevi alınıp 0'a eşitlenirdi. Burada da maliyet fonksiyonunun her bir θ değeri için türevi alınıp bu değer 0'a eşitlenir, böylelikle minimizasyon sağlanır. Bu birinci yoldur.
- İkinci yol ise şöyledir: Tüm özellikleri içeren bir matris oluşturulur. Matrisin ilk sütunu 1'lerden oluşur (çünkü bu sütun 0 indeksindeki x 'e tekabül eder.). Kısaca her bir sütun bir özellik belirtir. Bu matrisin aynısını gerçek değer (yani y) için yapalım. Böylelikle de y adlı vektörümüzü oluşturmuş oluruz.

- Burada x , $(m \times n+1)$ 'lik bir matris, y de $(m \times 1)$ 'lik bir vektör olmuş oldu (m değeri toplam training example sayısı, n değeri ise özellik sayısı. $n+1$ denilmesinin sebebi de 0 indeksindeki x değerini matrise eklememizdir.).
- X vektörü, 0'dan n 'ye kadar $n+1$ 'lik bir özellik(feature) vektörü. 1. training example'ın x vektörünün transpozu alınarak design matrix adında bir matris oluşturulur ve transpozu alınan x vektörü ilk satıra eklenir. Bu şekilde m 'e kadar bu işlem böyle tekrar ederek $(m \times n+1)$ 'lik design matrix oluşturulur. Böylelikle elimizde X ve Y matrisleri oluşmuş olur. Böylelikle aşağıdaki formüle geçebiliriz.

$$\theta = (X^T X)^{-1} X^T y$$

- Bu işlemi Octave'da `pinv (X' *X) *X' *y` ifadesi ile yapabiliriz.
- Burada *feature scaling* kullanılmasına gerek yoktur. Fakat *gradient descent* kullanılacaksa kesinlikle kullanılmalıdır.

Nerelerde Gradient Descent Nerelerde Normal Equation Kullanılmalı

- m tane training example ve n tane feature olduğunu farz edelim. Gradient descent kullanılacaksa burada en uygun α değerinin bulunması gerekir. Bu da ekstra maliyete sebep olur. Ek olarak gradient descent'te detaylara ve özelliklere göre çok fazla iterasyon gerekebiliyor, bu da bizim için bir kayıptır.
- Normal equation'da α seçilmesi işlemi, grafik çizerek yakınsama işlemlerine bakmak ve yakınsamanın doğurduğu iterasyonlar gibi işlemler yoktur.
- Gradient descent algoritması çok yüksek özellik(feature) sayılarında bile iyi şekilde çalışabilmektedir. Fakat Normal Equation işleminde özellik sayısı büyük olursa bu işlem o kadar uzar, algoritma o kadar yavaş çalışır.
- Burada n sayısı 1000'in altında ise teta parametrelerini çözmek için normal equation algoritmasını kullanmak daha verimli olacaktır. Fakat bu sayı yükseldikçe gradient descent algoritması daha verimli hâle gelecektir.

