Numpy ile Lineer Regresyon

H. Canberk KARAOĞLAN – 1188132104

Giriş

50’li yıllarda Alan Turing’in “Makineler düşünebilir mi?” sorusuyla başlamış ve temelinde insan işgücüne alternatif olarak bilgisayar sistemlerinin yer alabilmesi amaçlanarak makinelerin de tıpkı insanlar gibi çalışabilmesi ve olası durumları öngörebilir ve önlem alabilir kılan Yapay Zeka’nın kullanım alanları gün geçtikçe katlanmakta, metodolojisi daha da genişlemekte ve her alanda kullanılmaya başlanmaktadır. Bu çalışmada Yapay Zeka’ya ait Makine Öğrenmesi alanındaki en standart algoritmalardan birisi olan Lineer Regresyon’un çalışma prensiplerinin incelenmiş, Lineer Cebir’in Regresyon’daki işlemlerde karşılığı gelen kullanımı anlatılmış ve Python 3.7 kullanılarak kodlaması yapılmıştır. İlgili bu çalışmada sırasıyla, öncelikle Makine Öğrenmesi terimine yüzeysel bir giriş yapılmış, ardından matris tabanlı işlemlere değinilerek kodlamada Makine Öğrenmesi konusundaki sağladığı avantajlara değinilmiş, Python’a ait Numpy kütüphanesi ile regresyon işlemlerinin kodlaması yapılırken yine matris tabanlı bir programlama dili olan MATLAB ile kıyaslaması yapılmış ve son olarak ise Python’da regresyon için kullanılan çeşitli kütüphaneler incelenerek sonuçlar, temel metodolojik yaklaşımlarla kıyaslanmıştır.

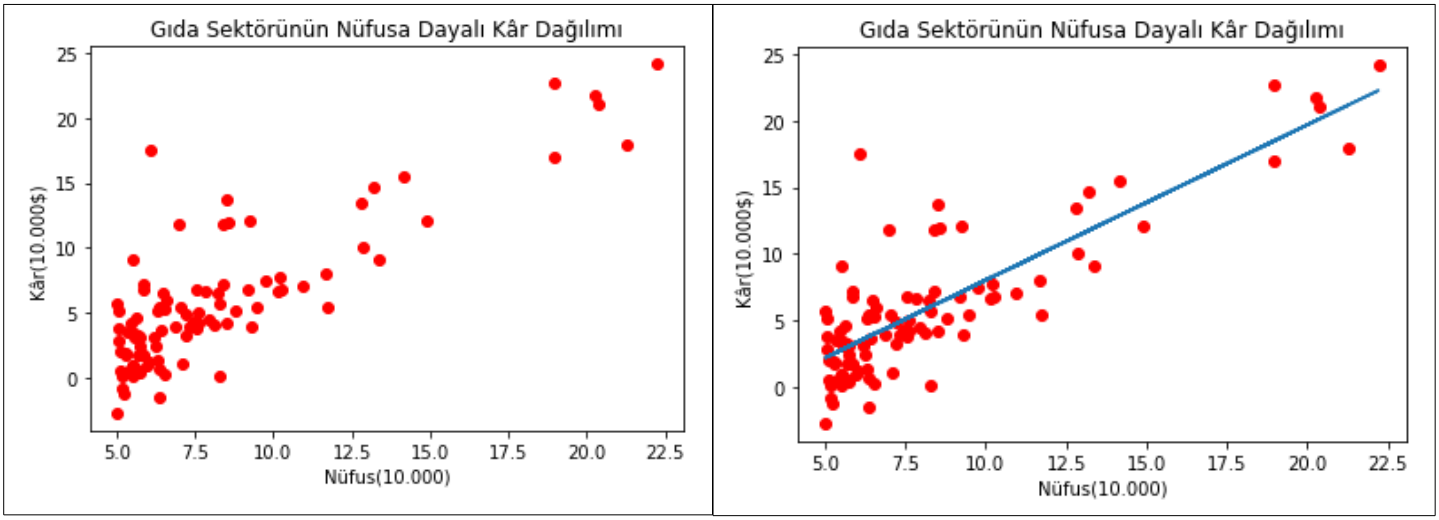
1. MAKİNE ÖĞRENMESİ

Makine öğrenmesi yapısal işlev olarak öğrenebilen ve veriler üzerinden tahmin yapabilen algoritmaların çalışma ve inşalarını araştıran bir sistemdir. Makine öğrenmesi metodolojisi, eğitim verilerinin etiketlenmesine ve bunun ortaya çıkardığı işlevselliğe dayalı olarak üç grupta incelenebilir.

- Gözetimli Öğrenme (Supervised Learning)
  - Regresyon
    - Lineer
    - Polinomsal
  - Sınıflandırma
- Gözetimsiz Öğrenme (Unsupervised Learning)
  - Kümeleme (Clustering)
  - Anomali Tespiti (Anomaly Detection)
  - Sinir Ağları (Gözetimlide de kullanılabilir)
- Diğer (Semi-Supervised Learning gibi)

Regresyonda amaç, sonucu belirtilen (etiketlenmiş) verilerden çeşitli algoritmalar ile öğrenme gerçekleştirilerek farklı parametreler için bir sonucu tahmin edebilmektir. Bu tahminin gerçekleştirilebilmesi için üstünde durulması gereken birkaç adım mevcuttur.

- Gösterim Sağlanabiliyorsa Veriler Plot ile Gösterilir (Çoklu Parametrelerde Feature Extraction gibi yöntemler kullanılarak etkin parametre gösterimi sağlanabilir)
- Çıkarılan Diyagrama Bağlı Olarak Eğitim Verilerine Uygun Bir Hipotez Belirlenir
- Hipotezde Başlangıç Olarak Bir Theta Değeri Verilir (Bu çalışmada 0 verilecektir)
- Uygun bir hata fonksiyonu (cost function, birçok çeşidi mevcuttur, bu çalışmada kareler farkını kullanacağız) kullanılarak hata değeri çıkarılır.
- Çeşitli algoritmalarla (Bu çalışmada Gradyan İnişi kullanılacaktır) hata oranını en düşüğe indiren ideal Theta değerleri bulunur.



Şekil 1: Verilerin Plot ile Gösterimi ve İlgili Hipotezin Uyarlanması

2. REGRESYON VE MATRİS TABANLI İŞLEMLER

Diğer makine öğrenmesi algoritmalarında olduğu gibi, Regresyon algoritmalarında da bir eğitim verisine ihtiyaç duyulmaktadır. Regresyon algoritması bir gözetimli öğrenme çeşididir, veriler etiketlidir. Verilerin etiketli olmasından kasıt, sistemdeki girdilere karşılık gelen sonuç eğitim verilerinin içinde mevcuttur. Regresyon ile geliştirilen model, sistemdeki girdilere karşılık çıkan sonuçları göz önüne alarak sisteme gönderilen parametrelere karşılık gelebilecek sonucu tahmin etmeye çalışmaktadır.

Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

Şekil 2: Örnek Bir Eğitim Kümesi

Şekil 2'de örnek bir eğitim veri kümesi tanımlanmıştır. Bu eğitim verisi, bir evin adım birimiyle tanımlanmış alan ve yatak odası sayısı bilgisiyle evin kirasını tahmin etmek için kullanılan bir eğitim verisidir. Bu örnekte sistemdeki girdi parametreleri (X), evin alan bilgisi ve yatak odası sayısıdır ve çıktı parametresi (Y) ise tahmin edilmeye çalışılan kira değeridir. Yaygın literatüre göre; tahmin edilmeye çalışılan çıktı parametresi, girdi parametrelere bağlı değişkenlik gösterdiği için Y parametresi bağımlı değişken ve girdi parametreleri ise birbirinden bağımsız olarak farklı değerlere sahip olabileceğinden ötürü X parametreleri ise bağımsız değişken olarak adlandırılabilmektedir.

Şekil 1'de de gösterildiği gibi, Regresyon'da amaç tüm eğitim verilerini kapsayacak bir düzeyde tahmini gerçekleştirebilmeyi sağlayabilecek bir hipotezi geliştirmek ve bu hipotezin matematiksel karşılığının bulunmasıdır. Şekil 1'deki diyagramda nüfus bir girdi (X) parametresi, kâr oranı bir çıktı (Y) parametresi ve mavi ile çizilen çizgi ise örnek bir hipotezdir. Tahmin işlemi, girilen bir X parametresinin hipotezdeki Y parametresine karşılık gelecek değerin bulunmasıyla gerçekleştirilir.

Hipotezin (h(x)) genel olarak matematiksel tanımı

$$h(x) = \theta^T X$$

Bu ifadeyi genel olarak tanımlamak gerekirsek, θ hipotezin matematiksel tanımındaki katsayıları temsil ederken X ise girdi verimiz olarak tanımlanmaktadır. Örnek olarak tek parametreli bir hipotezin matematiksel tanımı

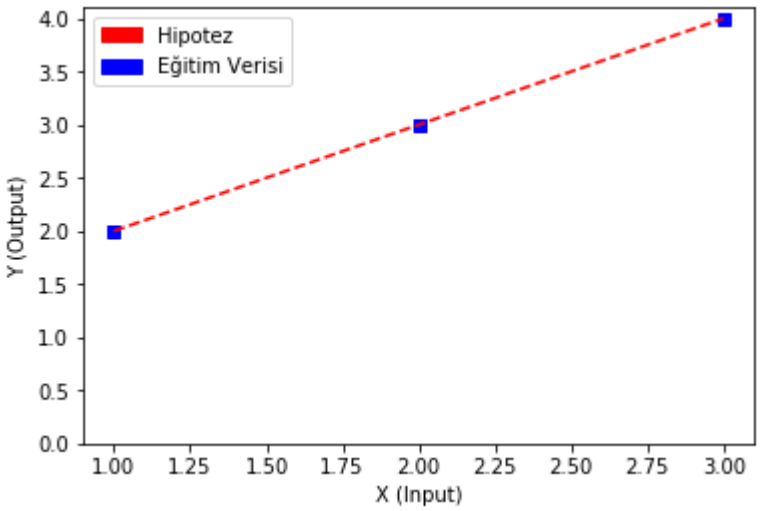
$$h(x) = \theta_0 + \theta_1 X$$

biçiminde ele alınır. Örnek olarak girdi verisi olarak X = [1,2,3] ve bu girdi verilerine karşılık gelen çıktı verilerini Y = [2,3,4] olarak ele alalım. Matematiksel olarak denklemimiz

$$Y = 1 + X$$

olacaktır. Hipotez ise, bu denkleme karşılık olarak gelebilecek denkleme karşılık gelecektir ve θ katsayıları tam olarak burada kullanılmaktadır.

$$y = h(x) = \theta_0 * 1 + \theta_1 * X$$



Şekil 3: Veri Setlerinin Dizilmesi ve İlgili Hipotezin Uyarlanması

Şekil 3'te gösterildiği gibi, hipotez belirlendikten sonra herhangi verilen bir X değerinin tahmini, hipotezin üzerinden karşılık gelen Y değerinin bulunmasıyla gerçekleştirilecektir. Bir modelin eğitilmesi ve modelin karşılık gelen sonucu tahmin etmesi temel olarak bu şekilde gerçekleşmektedir. Bunun için tüm regresyon problemlerinde θ değerlerinin belirlenebilmesi, tahminin sağlanabilmesi için temel bir gereksinimdir.

Tüm θ değerleri ve X değerleri bir vektör (tek boyutlu dizi) içinde tanımlanır.

$$\theta = [\theta_0, \theta_1, \theta_2 \ldots \theta_n] \quad X = [X_0, X_1, X_2 \ldots X_n]$$

X_0 olarak belirtilen değişken θ_0 ile çarpımı için mevcuttur ve değeri hemen hemen her zaman 1 olarak tanımlanır. Vektör içinde bir değişken olarak tanımlanmasının sebebi θ ve X'in eleman sayısının Lineer Cebir içindeki matris çarpımı kuralına göre (ilk matrisin satır sayısı ikinci matrisin sütun sayısına eşit olması gerekliliği) eşit olma zorunluluğudur, eğer bu değer tanımlanmazsa çarpım işlemi gerçekleştirilemez. θ ve X'in birer vektör olarak tanımlanması $h(x) = \theta^T X$ olarak tanımlanan hipotezin implementasyonunu oldukça kolaylaştırmaktadır. Bu şekilde matris tabanlı bir çarpım gerçekleştirilerek gereksiz döngülerle oluşacak uzun kodlar tek satırlık bir matris çarpımına indirgenir ve farklı öznitelik sayısına sahip birçok farklı problemlerde hipotez

$$h(x) = \theta_0 X_0 + \theta_1 X_1 + \ldots + \theta_n X_n$$

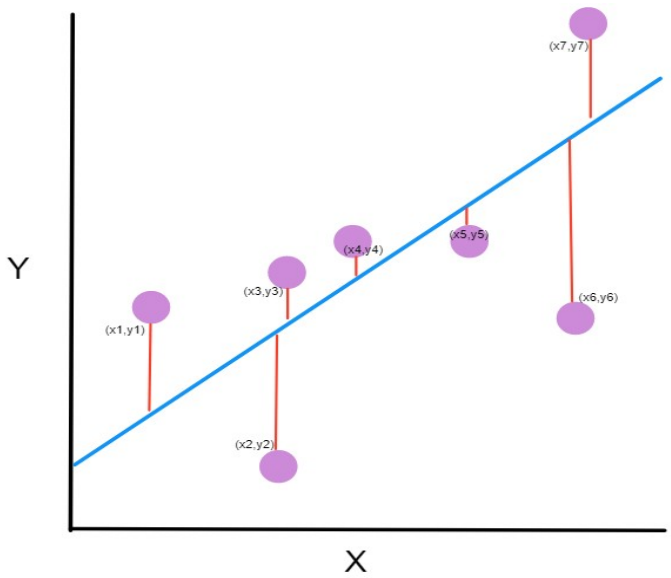
biçiminde ele alınıp teker teker yazılmak ve farklı problemlerde düzeltilmek yerine genel ifade kullanılarak kod yazma zahmetinden ve karmaşasından korunulmuş olunur. Örneğin 100 öznitelikli bir problemde hipotezin 0'dan 100'e kadar yukarıdaki denklem çerçevesinde kodsal olarak tanımlandığı düşünülürse her anlamda ortaya çıkan gereksiz maliyet ve karmaşa daha da öngörülebilir. Sadece bu açıdan bakılırsa bile, matris tabanlı kodlamanın gerek regresyon gerek diğer yapay öğrenme algoritmalarında çok ciddi bir ihtiyaç olduğu oldukça açıktır.

İdeal θ değerlerini bulmanın regresyon ve diğer yapay öğrenme algoritmalarında temel bir gereksinim olduğu daha önceden belirtilmiştir. İdeal θ değerlerinden bahsedilen değerler, hipotezin hata oranını ve sapma değerlerini olabilecek en düşük ve tahmin tutarlılığını en yüksek kılan değerlerdir. İdeal θ değerlerinin bulunabilmesi için gerekli olan birçok hata hesaplama yöntemi ve bu hata oranını indiren θ değerlerini hesaplayan birçok algoritma mevcuttur. Bu çalışmada hata hesaplama yöntemi olarak sıklıkla kullanılan Ortalama Hata Karesi (Mean Squared Error) ve hata oranını düşüren ideal θ değerlerini bulan algoritma olarak ise Gradyan İnişi (Gradient Descent) kullanılmıştır.

Hata değerine girmeden önce Şekil 1'de görüldüğü gibi tüm eğitim verilerinin hepsini tek bir hipotezde toplamak genel itibariyle imkansızdır, bu odaklı bir hipotez geliştirmek overfitting problemlerine (sonradan anlatılacaktır) sebebiyet verir ve genel olarak hipotezi başarısız kılar. Ortalama Hata Karesi değeri, noktaların hipoteze olan uzaklıkların karelerinin toplamı ile bulunmaktadır. Bu, analitik geometrideki uzaklık ile özdeleştirilebilen bir değerdir.

$$J = 1/(2*m)*\sum_{i=1}^{n}(h_i(x)-y_i)^2$$

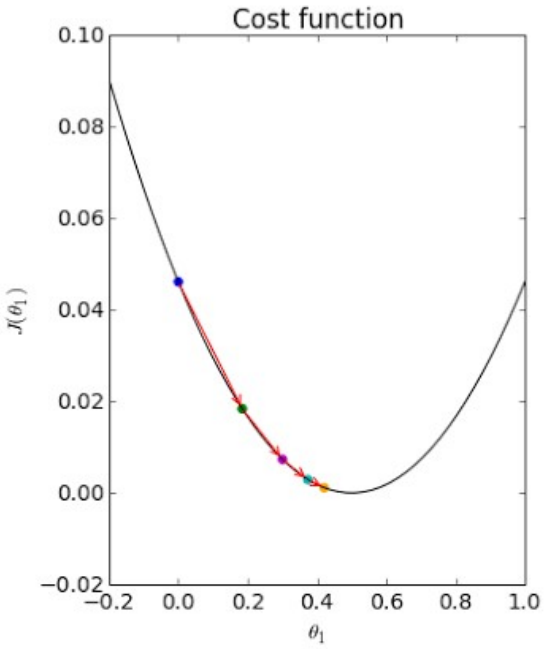
Yukarıdaki denklemde m toplam eğitim verisi adedini, n öznitelik sayısını belirtmektedir. Bahsedilen bu hata değeri aynı zamanda literatürde Cost Function olarak da geçmektedir.



Şekil 4: Konumlanan Verilerin Hipoteze Nazaran Uzaklığının Belirlenmesi

Hata oranı hesaplama yapıldıktan sonra yapılacak iş, bu hata oranının gerek gelecekte eklenecek veriler için başarım oranının / tutarlılığın yükseltilmesi gerekse sapma değerinin azaltılmasıdır. Sapmanın azaltılması için uygun θ değerlerinin belli iterasyonlarla değişimini sağlayan algoritmalara ihtiyaç duyulmaktadır. Bu çalışmada Gradyan İnişi algoritması kullanılmıştır.

Gradyan İnişi algoritması, belli adımlarla θ'nın belli formülasyona dayalı değişimini sağlayarak iterasyonlarla J'nin olabilecek en düşük değeri almasını sağlayan türevsel bir algoritmadır. J/ θ grafiğiyle de gösterimi sağlanabilecek diyagramda oluşan şekilde J'nin minimum noktasına gelebilecek θ değerlerini iterasyonlarla bulmaya çalışır.



Şekil 5: J(θ) / θ Diyagramı

Şekil 5'te de görüldüğü gibi, J değeri θ değerlerine göre değişim göstermektedir, Gradyan İnişi algoritmasının temel amacı J noktasının en düşük olduğu optimum noktasını ve bu noktadaki θ değerlerinin bulunmasını sağlamaktır.

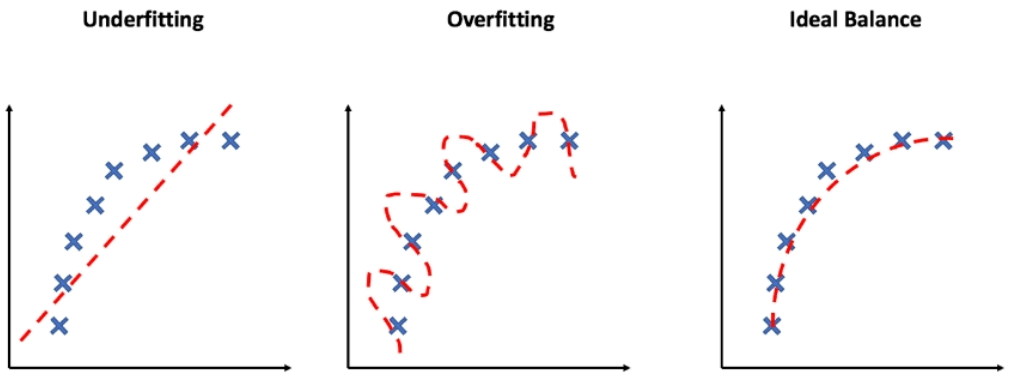
$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Yukarıda Gradyan İnişi'nin temel olarak formülü verilmiştir. Tüm θ değerlerinin değişimi bu denklem bağlamında senkron olarak belli bir iterasyon / döngü içinde sağlanır. Burada verilen α değeri Şekil 5'te gösterilen diyagramda, belli bir noktadan optimum noktasına doğru ilerleme oranını / adım birimini belirtmektedir. α değerine çok küçük değer verilmesi iterasyon sayısını arttırarak işlem maliyetini arttıracaktır. Çok büyük değer verilmesi durumunda ise overshooting adı verilen, konveks diyagramda minimum noktaya yaklaşmak yerine giderek uzaklaşıp işlem sonunda sonsuz (Inf) değerinin dönmesine sebebiyet veren hata meydana gelir. α değerinin belirlenmesinde net bir kural yoktur, genellikle deneme yanılma yoluyla uygun değer bulunur. α ile çarpılan ifade ise türevsel bir ifadedir, Hata değerini çıkartan J fonksiyonunun θ değerlerine göre türevini ifade etmektedir. Bu ifade Hata Fonksiyonu'na (Cost Function) ve uygulanan algoritmaya göre değişmektedir. Yukarıda tanımlanan J fonksiyonuna göre formülasyon:

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

Bu çerçevede yapılan belli iterasyonlardan sonra elde edilen θ değerleriyle örnek girdi veriler çarpılarak tahmin gerçekleştirilmiş olur.

Hipotezlerin hatalı belirlenmesinden ötürü ortaya çıkan Underfitting ve Overfitting sorununa da değinmek oldukça faydalı olacaktır. Hipotez eğer mevcut eğitim verileriyle bile örtüşmüyorsa bu sorun Underfitting olarak geçmektedir, herhangi bir şekilde tutarlılık beklenmemektedir. Overfitting ise hipotezin tamamen mevcut eğitim verilerine oldukça bağlı bir şekilde geliştirilmesi sonucu ortaya çıkan bir sorundur. Hipotez, eğitim verileriyle uyumluluk sağlayacak bir şekilde geliştirilmesine rağmen test olarak konulan verilerde yanıltma ihtimali çok yüksektir.



Şekil 6: Underfitting, Overfitting ve İdeal Hipotezin Temsili Diyagramı

3. NUMPY KÜTÜPHANESİ

Numpy kütüphanesine girmeden önce, matris tabanlı işlemlerin kodlamada nasıl bir kolaylık ve sadelik sağladığını gösterebilmek amacıyla basit bir matris çarpma işleminin sınıf temelli olmayan yapısal kodlama ve sınıf tabanlı Numpy kütüphanesinin kullanılarak yapılan kodlamanın kıyaslanması faydalı olacaktır.

```
#Numpy Kullanilmadan
d1 = [[1,2],[3,4],[1,2]] #3*2 matris
d2 = [[4,5,2],[6,7,1]] #2*3 matris
r1,c1,c2 = len(d1),len(d1[0]),len(d2[0]);
if (len(d1)==len(d2[0])):
    cMatrisi = [];
    for i in range(0,r1): #Carpim matrisinin boyutlariyla olusturulmasi
            cMatrisi.append([0]*c2)
    for j in range(0,r1):
        for k in range(0,c2):
            for l in range(0,c1):
                cMatrisi[j][k] += d1[j][l] * d2[l][k]
else:
    print("Matrisler carpilamaz")
print(cMatrisi);
```

```
#Numpy ile
import numpy as np
nD1,nD2 = np.array(d1),np.array(d2);
nCMatris = nD1.dot(nD2);
print(nCMatris)
```

İlk kod, bir matris çarpımının basit bir yapısal kodlamasını temsil ederken ikinci kod Numpy kütüphanesi ile kodlanmasını temsil etmektedir. İlk kodda tüm exception durumlarının (örn. boş elemanlı dizilerin işleme sokulması hata üretmelidir) ele alınmamasına rağmen, dizi tanımlanmaları ve yazdırma kodları gözardı edildiğinde sadece çarpma işlemi 11 satır tutmaktadır. Numpy kütüphanesi kullanımında ise bu işlem sadece 1 satırda yapılabilmektedir. Bir hata olduğunda 11 satır yerine 1 satırda aranması ve bu kolaylığın, sadeliğin sağlanması satır bazlı yaklaşıldığında 11 kat daha verimli bir kod yazılmasının önünü açacaktır.

Numpy'nin matris tabanlı mevcut birçok işlevi gerçekleştirebilecek fonksiyonları mevcuttur. Bu bölümde Numpy'nin tüm fonksiyonlarından daha ziyade ağırlıklı olarak Regresyon problemlerinde kullanılabilecek fonksiyonlara yer verilecektir.

- Matris Oluşturma

  Numpy kütüphanesinin etkin bir şekilde kullanılabilmesi için, dizilerin Numpy sınıfı ile oluşturulması gerekir. Eğer diziler, Numpy sınıfı ile tanımlanmazsa dizilerde Numpy'nin sağladığı özellikler ve yordamlar mevcut olmayacaktır.

  array(dizi)

  Dizi oluşturmak için array() fonksiyonu kullanılır. Dizi daha önceden başka bir değişkende tanımlanıp yordam içinde veya tanımlanmaksızın yordam içinde de tanımlanarak kullanılabilir.

  ones()

  Tamamen birlerle dolu bir matris üretir.

  zeros()

  Tamamen sıfırlarla dolu bir matris üretir.

  arange()

  Girilen sayıya kadar girilen sayı hariç sırayla o sayıya kadar arttırımsal bir dizi oluşturur.

  random.rand()

```
#Matris Oluşturma
D = np.array([1,2,3]); #Output [1 2 3]
D = np.ones(5); #Output [ 1.  1.  1.  1.  1.]
D = np.ones ((5,5)); #Output Tüm elemanları 1 olan 5*5 matrisi
D = np.zeros(5); #Output [ 0.  0.  0.  0.  0.]
D = np.zeros((5,5)); #Output Tüm elemanları 0 olan 5*5 matrisi
D = np.arange(5); #Output [0 1 2 3 4]
D = np.random.rand(3,1); # Elemanlari 0-1 arasinda random 3*1 matris
```

0-1 değerleri arasında random elemanları olan matrisi oluşturur.

- Matris Seçme

  Tıpkı MATLAB'daki gibi, indisler kullanılarak çeşitli operatörlerle belirli elemanlar çekilebilir. Burada Numpy kütüphanesi haricinde Python'un kendi içindeki mevcut yapısının getirdiği işlevsellikler de kullanılabilir.

```
#Matris Seçme
D = np.array([[1,2,3],[4,5,6]]); #2D Matris
print(D);print(D[0:2]);print(D[0::]) #Output Ayni Matris
print(D[::][0]) #Satir Output [1 2 3]
print(D[::][:,0:1]) #Sutun Output [[1][4]]
```

- Matris Tabanlı İşlemler

  Operatörler

  Numpy kütüphanesi, matrislerde toplama çıkarma çarpma ve bölme işlemlerinin kolaylıkla yapılabilmesi için sırasıyla + - * / operatörlerine işlevsellik (operator overloading) kazandırmıştır. Normal şartlarda Numpy ile tanımlanmamış dizilerde sadece operatör kullanarak, herhangi bir döngü kullanmaksızın matris tabanlı işlemleri gerçekleştirmek mümkün değildir. Operatörlerde çarpma işleminde dikkat edilmesi gereken husus, çarpma işleminin Lineer Cebirdeki çarpma işlemini temsil eden bir işlem değil, aksine matris elemanların karşılıklı birbiriyle çarpıldığı (element-wise) bir çarpma işlemidir. Aynı çarpma işlemi *multiply* kodu ile de yerine getirilebilir.

  dot()

  Lineer Cebir'deki çarpma işlemini yerine getiren koddur. İlk matrisin satır sayısının ikinci matrisin sütun sayısına eşit olması gibi kuralları takip ederek çalışır. Tanımlanan her bir Numpy matrisinde (array() ile tanımlanmış matrisler) bu yordam mevcuttur. Python 3.5 ve üstündeki sürümlerde @ işareti kullanılarak da bu işlem gerçekleştirilebilir. Kodun okunabilirliği ve parantez karmaşasını azaltmak için dot() yerine @ ile çarpmanın yapılmasında fayda vardır.

  transpose()

  Matrisin transpozunu geri döndüren koddur. Hem Numpy nesnesinde hem de Numpy matrislerinde ilgili kod mevcuttur. Aynı zamanda Numpy matrislerinde bir özellik (property) olarak da .T biçiminde mevcuttur.

  sum()

  Numpy nesnesinde bulunan matris elemanlarının toplamını döndüren yordamdır. Axis isimli mevcut bir parametresi mevcuttur. Bu parametre girilmeksizin kullanılırsa tüm satır ve sütun elemanlarını tek bir skaler bir değer olarak döndürür. 2D matrislerde axis = 0 ile sütunların

toplamı vektörü axis = 1 olduğunda ise satırların toplamı vektörünü gönderir. Tek boyutlu bir vektörde axis, boyuttan ötürü sadece 0 değerini alacaktır, 1 değerini aldığında IndexOutOfBoundary hatası ile karşılanır, dolayısıyla vektörlerde axis parametresi kullanımı gereksizdir.

Not: axis parametresi birçok Numpy yordamında kullanılan bir parametredir ve bu parametrenin kullanıldığı diğer tüm Numpy yordamlarında axis değeri girilen parametreye göre işlemin satırsal veya sütunsal yapılmasını sağlar.

power()

Numpy nesnesinde bulunan kuvvet alma yordamıdır. Element-wisedir. Girilen matrisin, girildiği kuvvetteki değerlerini hesaplayan bir matris döndürür. Aynı şekilde ** operatörü de bu işlem için kullanılabilir.

```
#Matris Tabanlı İşlemler
A = np.array([[1,2],[4,5]]);
B = np.array([[4,5],[7,8]]);
print(A+B); #Output [[5,7][11,13]]
print(A-B); #Output [[-3,-3][-3,-3]]
print(A*B); #Output [[4,10][28,40]]
print(A/B); #Output [[ 0.25 0.4][0.57142857 0.625]]
print(A@B); #Output [[5,7][11,13]]
print(A.T); #Output [[5,7][11,13]]
print(np.sum(A)) #Output 12
print(np.sum(A,axis=0)) #Sutun Tabanli: Output [5 7]
print(np.sum(A,axis=1)) #Satir Tabanli: Output [3 9]
print(np.power(A,2)) #Output [[1 4][16 25]]
```

- Kullanışlı Fonksiyonlar

ndim()

Bir dizinin (matris/vektör) kaç boyutlu olduğu bilgisini döndüren Numpy nesnesine ait bir yordamdır.

reshape()

Vektör veya matris olsun fark etmeksizin bir diziyi girilen satır ve sütun bilgisine göre yeni bir iki boyutlu matrise çeviren Numpy nesnesine ait bir yordamdır.

concatenate()

İki farklı matrisi bir araya getirip birleştirerek yeni bir matris üreten Numpy nesnesine ait bir yordamdır. axis parametresi mevcuttur.

size()

Bir matrisin içindeki eleman sayısı bilgisini döndüren Numpy nesnesine ait bir yordamdır.

shape()

Bir matrisin kaça kaçlık (kaç satır kaç sütun) matris olduğu bilgisini döndüren Numpy nesnesine ait bir yordamdır.

nonzero()

Yordamın içindeki şartı sağlayan matris elemanlarının indislerini döndüren Numpy nesnesine ait bir yordamdır. Matlab'da find komutuna karşılık gelir. Kodlamada olası olarak yazılabilecek birçok gereksiz döngü ve işlemden kurtarması açısından sıklıkla kullanılır.

loadtxt()

Sayısal veri kaynağı olan bir metin tabanlı (txt) dosya içeriğinin, bir değişkene matris olarak atanmasını sağlar. Dosya isminin yanında delimiter ve usecols gibi parametreleri de mevcuttur. Delimiter parametresi, dosya içeriğindeki verileri birbirinden ayıran karakterin belirtilmesinde kullanılırken usecols ise dosya içeriğinden oluşturulacak matrisin sütun sayısını temsil etmektedir.

```
#Kullanışlı Fonksiyonlar
A = np.arange(6)+1; #Bir vektör tanımı
print(A); #Output : [1 2 3 4 5 6]
print("Boyut: ",np.ndim(A)); #Output Boyut :1
A = A.reshape(1,-1); #Tek satırlı 2D matris, köşeli paranteze dikkat
print(A); #Output : [[1 2 3 4 5 6]]
print("Boyut: ",np.ndim(A));#Output Boyut :2
print(A.reshape(2,3));#Output [[1 2 3][4 5 6]
print(np.concatenate((A,A),axis=0)); #Output [[1 2 3 4 5 6][1 2 3 4 5 6]]
print(np.size(A)); #Output 6
print(np.shape(A)); #Output (1,6)
B = np.random.rand(1,5);
print(B);
print(np.nonzero(B>0.5));
```

4. PYTHON İLE REGRESYON

Bu bölümde Python ile yapılan kodlamalar anlatılmıştır. Eğitim kümesi olarak Andrew Ng'nin CourseRa'daki Makine Öğrenmesi dersinde kullandığı, tek parametreli ve çok parametreli olmak üzere iki adet eğitim kümesi kullanılmıştır.

İlk olarak, tek parametreli yaklaşım olan iki sütuna sahip eğitim verisi dosyadan okunmuş ve X ve Y olmak üzere iki ayrı değişkene atanmıştır.

```
#Verilerin Yuklenmesi
testData = np.loadtxt("ex1data1.txt",delimiter=',',usecols=range(2));
m = len(testData); #Test Veri Adedi
X = np.array(testData[:,0]).reshape(m,1);
Y = np.array(testData[:,1]).reshape(m,1);
```

Ardından Şekil 1'de solda yer alan diyagram plot komuylarıyla oluşturulmuştur.

```
#Verilerin Plot ile Gosterilmesi
plt.plot(X,Y,'ro');
plt.xlabel("Nüfus(10.000)");
plt.ylabel("Kâr(10.000$)");
plt.title('Gıda Sektörünün Nüfusa Dayalı Kâr Dağılımı')
plt.show()
```

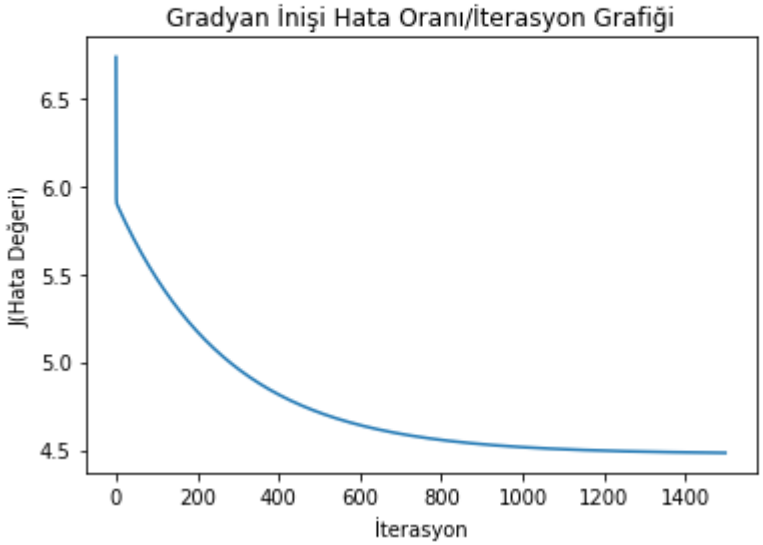
Gradyan inişi ihtiyaç duyduğu için daha önce 1 değeri verildiği söz edilen X_0 değişkeni için

eleman sayısı kadar 1'lerle dolu olan X_0 sütun matrisi, X ile concatenate() fonksiyonu ile birleştirilmiştir. Tek parametreli olduğu için iki adet Theta (θ) değeri tanımlanmış ve iterasyon ile alpha değerleri tanımlanmıştır. Tek parametre için çalışan Hata Değeri üreten costFunc() fonksiyonu ve aynı şekilde tek parametre için çalışan Gradyan inişi için gradientDescent fonksiyonları yazılmıştır. [0 0] Theta değerleri için başlayan Gradyan inişi 1500 iterasyon sonucunda uygun Theta değerlerini bulmuştur.

```
#Cost Function ve Gradyan İnişi
X = np.concatenate((np.ones(m).reshape(m,1),X),axis=1) #Sutun olarak 1 eklenir

theta = np.zeros((2,1));
iteration,alpha = 1500,0.01;

J = costFunc(theta,X,Y);
print("Theta0 = 0 Theta1 = 0 iken Hata Degeri: ",J);

J = costFunc([-1,2],X,Y);
print("Theta0 =-1 Theta1 = 2 iken Hata Degeri: ",J);

[theta,J_History] = gradientDescent(X,Y,theta,alpha,iteration);
print("Gradyan İnişe Göre İdeal Theta Değerleri: \nTheta0:",*theta[0],"Theta1:
",*theta[1]);

plt.plot(np.arange(1500),J_History)
plt.xlabel("İterasyon");
plt.ylabel("J(Hata Değeri)");
plt.title('Gradyan İnişi Hata Oranı/İterasyon Grafiği')
plt.show()

def costFunc(theta,X,Y):
    #J = (1/(2*m)) * sum(((theta(1)+theta(2)*X(:,2))-y).^2)
    J = (1/(2*m)) * np.sum( ( np.power( ( (theta[0]+np.sum(theta[1])*X[:,1])-Y[:,0]), 2)
) );
    return J;

def gradientDescent(X,Y,theta,alpha,num_iters):
    m = len(Y[:,0]);
    J_history = np.zeros((num_iters,1))
    for itr in range(0,num_iters):
        temp1 =  theta[0] - alpha*(1/m)* np.sum((theta[0]+theta[1]*X[:,1])-Y[:,0]);
        temp2 =  theta[1] - alpha*(1/m)* ((theta[0]+theta[1]*X[:,1])-
Y[:,0]).dot(X[:,1]);
        theta[0] = temp1;
        theta[1] = temp2;
        J_history[itr] = costFunc(theta,X,Y);
    return [theta,J_history];
```

Bu işlemler sonucunda ideal Theta değerleri üretilmiş ve Gradyan İnişi'nin iterasyona göre indirdiği hata oranı diyagramda gösterilmiştir.

```
Theta0 = 0 Theta1 = 0 iken Hata Degeri:  32.0727338775
Theta0 =-1 Theta1 = 2 iken Hata Degeri:  54.242455082
Gradyan İnişe Göre İdeal Theta Değerleri:
Theta0: -3.6302914394 Theta1:  1.16636235034
```

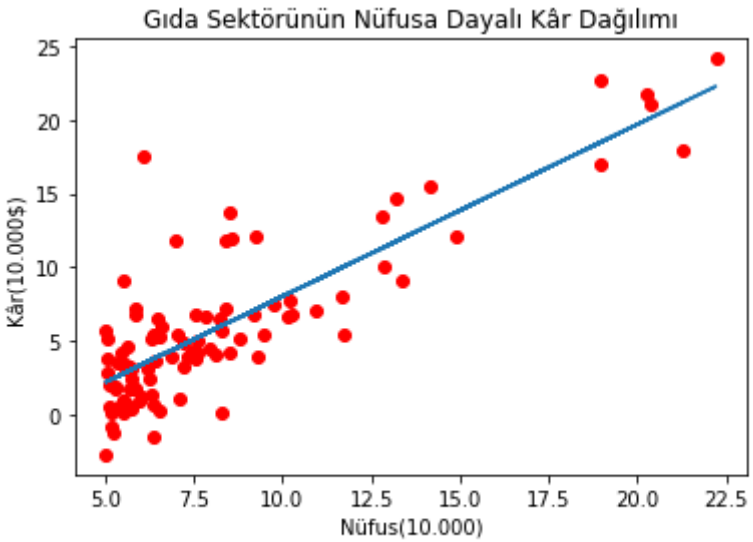


Şekil 7: Gradyan İnişinin Hata Oranını İterasyona Dayalı Düşürmesinin Görselleştirilmesi

Şekil 7'de gösterildiği gibi, Gradyan İnişi'nde yorumlanması gereken olgu belli bir iterasyondan sonra hata oranını çok düşük oranlarda (ondalık olarak binlik /onbinlik gibi) düşürmeye başlamasıdır. [0 0] Theta değerleri için hata değeri 32.072'lerde hesaplanırken Gradyan İnişi bu hata değerini 4.5'a kadar düşürmeyi başarmıştır. Şekilde 1200. iterasyonda gösterilebileceği gibi belli bir orandan sonra düşürme oranı ciddi düşmüş, bir noktadan sonra doğrusallaşmaya başlamıştır. Bunun gibi durumlarda mevcut koşullar örnek verilmesi gerekirse 1500 iterasyon yerine 1200 iterasyonla yapılması, 300 iterasyonun işlem gücü maliyetinin esgeçilmesi amacıyla tercih edilebilir.

İlk kullanılan bu eğitim kümesinde, nüfus bilgisi ve nüfusa dayalı olarak kurulan bir yemek şirketinin elde edebileceği kâr/zarar tahmin edilmeye çalışılmıştır. Nüfusun kalabalık olduğu yerlerde kurulan bir yemek şirketinin kârının da oranla arttığı gözlemlenmiş ve bu eğitim verisi tahmin için hazırlanmıştır.
İdeal Theta değerleri elde edildikten sonra ilgili hipotez, plot ile çizdirilmiş ve 20 br nüfus için elde edilebilecek kâr tahmin edilmiştir.

```
#Bulunan Hipotezin Verilerle Beraber Plot ile Gosterimi
plt.plot(X[:,1],Y,'ro',X[:,1],X.dot(theta),'-');
plt.xlabel("Nüfus(10.000)");
plt.ylabel("Kâr(10.000$)");
plt.title('Gıda Sektörünün Nüfusa Dayalı Kâr Dağılımı');
plt.show();

predict1 = np.dot(np.transpose(theta),[[1],[20]]);
print("Nüfus 20 br olduğunda Kâr Tahmini:", *predict1[0])
```




```
Nüfus 20 br olduğunda Kâr Tahmini: 19.6969555673
```

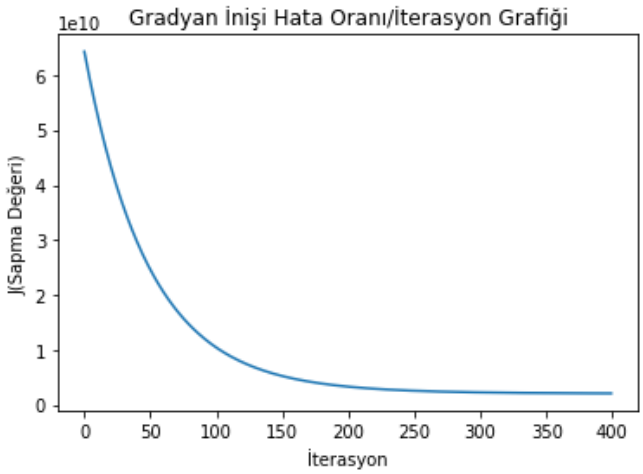
Şekil 8: Hipotezin Diyagramının Çizilmesi ve Tahmin

Çoklu parametreli eğitim kümesi olarak da iki özniteliğe sahip, evin alanı ile kaç yatak odasına sahip olduğu bilgisinden kira tahmininin yapılmasını sağlayan bir metin dosyası mevcuttur. Bu kodlamada, bir diğer önceki kodlamadan farklı olarak featureNormalization() fonksiyonu mevcuttur. BatchNormalization olarak da geçen bu işlem, tüm verilerin belli kalıplarda küçültülmesini sağlayan bir işlemdir. Özellikle çok büyük rakamlı veri kümesiyle çalışılıyorsa (milyonluk değerler gibi) verilerin Gradyan İnişi gibi işlemlere sokulmadan önce BatchNormalization işleminden geçmesi, bilgisayarın büyük rakamlardan daha ziyade küçük rakamlarla hesaplamayı daha hızlı gerçekleştirmesinden ötürü işlem gücü olarak büyük bir tasarruf sağlamaktadır. Bu işlemde tüm verilerin ortalaması ve standart sapması alınır, verinin kendisinden ortalama çıkartılıp sonuç standart sapmaya bölünerek bilgisayarın hesaplaması daha kolay, daha küçük temsili sayılar elde edilmesi sağlanır. Bunun haricinde çoklu parametreli problemler için yazılmış Gradyan İnişi ve Hata Hesaplama fonksiyonları da yazılmıştır. Yazılan bu fonksiyonlar, tek parametreli problemler için de kullanılabilir

olmaktadır. Tek parametreli yazılan kodlar, arkaplanda işlemin nasıl döndüğünün daha rahat anlaşılabilmesi amacıyla yazılmıştır.

```
#Çoklu Parametrik Yaklasim
testData = np.loadtxt("ex1data2.txt",delimiter=',',usecols=range(3));
m = len(testData); #Test Veri Adedi
X = np.array(testData[:,[0,1]]).reshape(m,2);
Y = np.array(testData[:,2]).reshape(m,1);

[X,mu,sigma] = featureNormalize(X); #Verilerin Daha Küçük Sayılarla İfade Edilmesi
X = np.concatenate((np.ones(m).reshape(m,1),X),axis=1) #Sutun olarak 1 eklenir

iteration,alpha = 400,0.01;
theta = np.zeros((3,1));

[theta,J_History] = gradientDescentMulti(X,Y,theta,alpha,iteration);
print("Gradyan İnişe Göre İdeal Theta Değerleri: \nTheta0:",*theta[0],"Theta1:
",*theta[1],"Theta2: ",*theta[2]);
plt.plot(np.arange(400),J_History)
plt.xlabel("İterasyon");
plt.ylabel("J(Sapma Değeri)");
plt.title('Gradyan İnişi Hata Oranı/İterasyon Grafiği')
plt.show()

def featureNormalize(X):
    X_norm = X;
    mu = np.mean((X),axis=0);
    sigma = np.std((X),axis=0);
    X_norm = (X_norm-mu)/sigma;
    return [X_norm,mu,sigma];

def costFuncMulti(theta,X,Y):
    J = (1/(2*m)) * np.sum( ( np.power( (X@theta) - Y,2 ) ) );
    return J;

def gradientDescentMulti(X,Y,theta,alpha,num_iters):
    m = len(Y[:,0]);
    J_history = np.zeros((num_iters,1))
    for itr in range(0,num_iters):
        delta = (1/m) * np.sum((X*(X@theta -Y)),axis=0);
        theta = (theta.T - (alpha*delta)).reshape(len(theta),1);
        J_history[itr] = costFuncMulti(theta,X,Y);
    return [theta,J_history];
```

```
Gradyan İnişe Göre İdeal Theta Değerleri:
Theta0: 334302.063993 Theta1:  99411.4494736 Theta2:  3267.01285407
```



Tüm bu kodlar sonucunda Theta değerleri hesaplanmış ve Gradyan İnişi diyagramı çizdirilmiştir. Ardından tahmin gerçekleştirilmiştir. Bu kodda 1650 m^{2}lik 3 yatak odalı evin tahmin edilen fiyatı 289221.547371 $ olarak hesaplanmıştır.

```
d = [1650,3];
d = np.array( [(d - mu) / sigma] );
d =np.concatenate([np.ones((1,1)),d],axis=1);
price = (d @ theta)[0];
print("1650 m2lik 3 yatak odalı evin tahmin edilen fiyatı: ",*price, "$")
```

5. PYTHON'DA ALTERNATİF KÜTÜPHANELER

Yukarıda yazılmış kodlar Lineer Regresyon işlerinin arkaplanda nasıl çalıştığını etkin bir şekilde anlatılabilmesi amaçlı yazılmıştır. Bunların haricinde tüm bu kodların yazılması yerine Sklearn kütüphanesi de kullanılarak tahmin işlemleri gerçekleştirilebilir. Bu kütüphaneler, gerek akademik gerek sektörel açıdan bir başarım oranını yükseltme çalışmalarından daha ziyade sadece tahmin işlemlerinin gerçekleştirilebilmesi amacıyla kullanılır. Başarım oranı yükseltmenin gerçekleştirilebilmesi için arkaplanda kodların nasıl çalıştığını bilmek ve arkaplanda çalışan o kodların gerektiğinde yazılabilme pratiğinin olması gerekir. İhtiyaç duyulan şey sadece basit bir tahmin gerçekleştirmekse bu kütüphanelerin kullanılması oldukça isabetli olacaktır.

Bunun için Sklearn kütüphanesi mevcuttur. Sklearn kütüphanesinde önceki çalışmada kullanılan eğitim kümeleri kullanılacak, bunun haricinde Lineer Regresyon konusu dışında yine bir Regresyon konusu olan Polinomsal Regresyon incelenecek, tahmin ve Theta değerleri kıyaslanacaktır.

Tek parametreli yaklaşımla ilgili kütüphane aşağıdaki şekilde kullanılmıştır.

```
#Sklearn Kütüphanesi Kullanılarak Yapılan Lineer Regresyon
from sklearn.linear_model import LinearRegression
testData = np.loadtxt("ex1data1.txt",delimiter=',',usecols=range(2));
m = len(testData); #Test Veri Adedi
X = np.array(testData[:,0]).reshape(m,1)
Y = np.array(testData[:,1]).reshape(m,1)
model = LinearRegression().fit(X,Y);
r_sq = model.score(X, Y);
print('Belirleme Katsayısı:', r_sq);
print('Intercept(Theta0):', model.intercept_);
print('Eğim(Theta1):', model.coef_);

y_pred = model.predict(20);
print('20br Nüfusa Gelen Kâr: ', y_pred);
```

Bu kodda dikkat edilebileceği üzere, veriler dosyadan aynı şekilde okunduktan sonra LinearRegression() nesnesine ait fit() yordamına gönderilmiş ve böylelikle modelin kendisi oluşturulmuştur. Model bir LinearRegression nesnesine eşitlenmiştir ve modelde sınıfta mevcut

olan score() yordamıyla da belirleme katsayısına ulaşılmıştır. Belirleme Katsayısı, 0 ile 1 arasında bir değer olup başarım oranıyla ilgili çıkarım yapılabilmesini sağlayan bir sayıdır. Bu sayının 1 veya 1'e yakın olması başarım oranının iyi olduğunu gösterebilmesiyle beraber aynı zamanda olası bir overfitting problemine işaret edebilir. Dolayısıyla Belirleme Katsayısı'nın değerlendirilmesinde tüm bu şartlar göz önünde tutulmalıdır. Bu hesaplamalar sonucunda aşağıdaki değerler elde edilmiştir:

```
Belirleme Katsayısı: 0.702031553784
Intercept(Theta0): [-3.89578088]
Eğim(Theta1): [[ 1.19303364]]
20br Nüfusa Gelen Kâr:  [[ 19.96489201]]
```

Çoklu parametreli yaklaşımda ise yine benzer işlemler yapılmıştır.

```
#Sklearn Kütüphanesi Kullanılarak Yapılan Çok Parametreli Lineer Regresyon
testData = np.loadtxt("ex1data2.txt",delimiter=',',usecols=range(3));
m = len(testData); #Test Veri Adedi
X = np.array(testData[:,[0,1]]).reshape(m,2);
Y = np.array(testData[:,2]).reshape(m,1);

model = LinearRegression().fit(X,Y);
r_sq = model.score(X, Y);
print('Belirleme Katsayısı (Determination of Coef):', r_sq);
print('Sabit (Intercept) (Theta0):', model.intercept_);
print('Eğim (Slope) (Theta1,Theta2):', model.coef_);

# y_pred = model.intercept_ + np.sum(model.coef_ * x, axis=1)
y_pred = model.predict([[1650,3]]);
print('1650 m2lik 3 yatak odalı evin tahmin edilen fiyatı: ', *y_pred[0] ,"$");
```

Ve tüm bu işlemler sonucunda ise aşağıdaki değerler elde edilmiştir:

```
Belirleme Katsayısı (Determination of Coef): 0.732945018029
Sabit (Intercept) (Theta0): [ 89597.9095428]
Eğim (Slope) (Theta1,Theta2): [[  139.21067402 -8738.01911233]]
1650 m2lik 3 yatak odalı evin tahmin edilen fiyatı:  293081.464335 $
```

Aynı zamanda Sklearn kütüphanesi ile Polinomsal Regresyon işlemler de yapılabilir. Polinomsal Regresyon'un Lineer Regresyon'dan tam olarak farkı, hipotezin polinomsal bir denklem olmasıdır. Hipotezin matematiksel denklemi en az ikinci dereceden bir denklemdir ve dolayısıyla hipotezin kaçıncı dereceden bir denklem oluşu, Theta sayısını da değiştirmektedir. (Theta Sayısı = Hipotez Derecesi + 1)

```
#Polinomsal Yaklaşım
from sklearn.preprocessing import PolynomialFeatures
X = (np.arange(15)+1).reshape(-1,1); #Polinomsal yaklasim icin 2D gerekir
Y = np.power(X,3);
Xt = PolynomialFeatures(degree=3, include_bias=False).fit_transform(X);
model = LinearRegression().fit(Xt,Y);
r_sq = model.score(Xt, Y)
intercept, coefficients = model.intercept_, model.coef_
y_pred = model.predict(Xt);
print('Belirleme Katsayısı (Determination of Coef):', r_sq);
print('Sabit (Intercept) (Theta0):', model.intercept_);
print('Eğim (Slope) (Theta1,Theta2,Theta3):', model.coef_);
cmp = np.concatenate((Y,y_pred),axis = 1);
print(cmp);
```

Polinomsal yaklaşımda, Lineer Regresyon kütüphanesinin yanı sıra PolynomialFeatures nesnesi kullanılmaktadır. Bu nesnenin içinde, hipotezin derecesini belirtmek için degree parametresi mevcuttur, include_bias parametresi ise, aslında Gradyan İniş için gerçekleştirilen x_0 değişkeni olarak girdi verisine 1 sütununun eklenip eklenmemesini belirtmek için mevcut olan bir boolean parametresidir. Bu parametre normalde girilmediğinde True olarak işaretlenmektedir. Bu çalışmada bu false olarak ele alınmıştır. Burada eğitim verisi olarak ise X için Python'dan oluşturulan 1'den 15'e kadar sayı dizisi ve Y olarak da karşılığı gelen veriye ise 3. kuvvetinin hesaplandığı bir değişken kümesi hazırlanmış ve Polinomsal yaklaşım tahmin için kullanılmıştır. Hesaplanan değişkenler şu şekildedir:

```
Belirleme Katsayısı (Determination of Coef): 1.0
Sabit (Intercept) (Theta0): [ -1.47792889e-12]
Eğim (Slope) (Theta1,Theta2,Theta3): [[  1.13398926e-13  -1.11022302e-15
  1.00000000e+00]]
```

6. SONUÇ

Lineer Regresyon ve arkaplanında yatan mantık anlatılmış, Python'da Lineer Regresyon kodları yazılmış ve Sklearn kütüphanesi ile yapılan hesaplamalarla kıyaslanılmıştır. Tüm bu kıyasın ardından aşağıdaki tablo çıkarılabilir.

Yöntem\Hesaplama	θ_0	θ_1	θ_2	Tahmin
Gradyan İniş Tek	-3.6302914394	1.16636235034	-	19.6969555673
Gradyan İniş Çok	334302.063993	99411.4494736	3267.01285407	289221.547371
Sklearn Tek	-3.89578088	1.19303364	-	19.96489201
Sklearn Çok	89597.9095428	139.21067402	-8738.01911233	293081.464335

Bu tabloda göze çarpan olgu, tahminlerin birbirine çok yakın yapılmasına karşın özellikle Çoklu parametrik hesaplamada belirgin olduğu gibi Theta değerlerinin birbirinden uzak değerlere sahip olmasıdır. Bunun sebebi olarak Sklearn kütüphanesinin tahmini Gradyan İnişi yaparak değil de yine oldukça sık kullanan ve tek satırla gerçekleştirilebilen Normal Equation yöntemiyle

yapmasıdır. Normal Equation yöntemiyle yapılan hesaplamalarda Theta değerleri Sklearn kütüphanesinin elde ettiği değerler olmaktadır.

Bu çalışmada, eğitim verisinin az sayıda küme içermesi sebebiyle (mtek = 97, mçok = 47) başarım ölçülme yapılmamıştır. Farklı eğitim verileriyle Cross Validation uygulanarak başarım kıyaslaması yapılabilir.

KAYNAKÇA

Pınar Tüfekçi – Derin Öğrenme Ders Notları

Andrew Ng – CourseRa Makine Öğrenmesi Ders Notları

https://realpython.com/linear-regression-in-python/