3.4 Çok Değişkenli Lineer Regresyon

Çoklu doğrusal regresyon iki ve daha fazla bağımsız değişken ve bir bağımlı değişken arasındaki doğrusal bağıntıyı inceler. Çoklu regresyonda bağımlı ve bağımsız değişkenler arasında bir bağıntı vardır. Bağımsız değişkenleri X, Bağımlı değişkenleri Y ile gösterelim.

$$Y = XB + E$$

- Y bağımlı değişken gözlem vektörü
- X bağımsız değişkenler gözlem matrisi
- B katsayılar vektörü
- E rasgele hata vektörü

Verilere çoklu regresyon uygulanabilmesi için, bağımsız değişkenler arasında çoklu bağımlılık (multicollinearity) olmaması gerekir.

Çok değişkenli doğrusal regresyon analizi çoklu doğrusal regresyon analizinin genellenmiş biçimidir, bağımlı değişken sayısının iki ve daha fazla olduğu durumlarda. Parametre tahminleri ve modelin geçerliliğini belirlemek için gerekli kareler toplamlarının hesaplanması çoklu regresyon yöntemi ile benzerlik göstermektedir.

Aşağıda denklemler için herhangi bir sayıda giriş değişkenine sahip olabildiğimizin gösterimini görebilirsiniz.

- $x_j^{(i)}$ = i. eğitim verisinin özelliklerinin değeri
- $x^{(i)}$ = i. eğitim verisinin özellik girdisi
- m = eğitin verilerinin sayısı
- n = özelliklerin sayısı

Bu çoklu özellikleri barındıran hipotez fonksiyonunun çok değişkenli biçimi aşağıdaki gibidir:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \ldots + \theta_n x_n$$

Matris çarpımının tanımını kullanarak, çok değişkenli hipotez işlevi kısaca şu şekilde temsil edilebilir:

$$h_\theta(x) = [\theta_0 \;\; \theta_1 \;\; \theta_2 \;\; \theta_3 \;\; \ldots \;\; \theta_n] \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = \theta^T x$$

Bu, bir eğitim örneği için hipotez fonksiyonunun bir vektörizasyonudur.

Dereceli azalma denkleminin kendisi genellikle aynı şekildedir; Sadece bütün özelliklerimiz için bunu tekrarlamalıyız:

$$\theta_j := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} h_\theta((x^{(i)}) - y^{(i)}) \cdot x_0^{(i)}$$

$$\theta_j := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} h_\theta((x^{(i)}) - y^{(i)}) \cdot x_1^{(i)}$$

$$\theta_j := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} h_\theta((x^{(i)}) - y^{(i)}) \cdot x_2^{(i)}$$

$$\ldots$$

$$\ldots$$

devam edin yakınsayana kadar

$$\ldots$$

devam edin yakınsayana kadar

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} h_\theta((x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \quad for\, j := 0 \ldots n$$

Buraya kadar anlatılanları ilerleyen bölümlerde örnekler üzerinden tekrar inceleyeceğiz.

3.4.1 Özellik Ölçeklendirme ve Normalleştirme

Basit bir dille ifade edecek olursak, makine öğrenmesi problemlerinde veri setimizdeki her bir örnek hep aynı aralıkta olmadığı için, dereceli azalma gibi ağırlıkların belirlendiği algoritmalarda bazı değerler diğerlerinden önce daha hızlı bir şekilde güncellenebilir. Bu duruma engel olmak amacıyla giriş değişkenlerimizin aralıklarını değiştirebiliriz ve böylece verilerimizin hepsi kabaca aynı aralıkta olur. Verilerin ideal aralıkları:

$$-1 \leq x(i) \leq 1 \; veya \; -0.5 \leq x(i) \leq 0.5'$$

Bunu sağlamak için iki yöntem vardır, özellik ölçeklendirme(feature scaling) ve ortalama normalleştirme(mean normalization). Özellik ölçeklendirme, giriş değerlerini giriş değişkeninin aralık (maksimum değer eksi en düşük değerinden) aralığına bölerek yeni bir bir aralığına neden olur. Ortalama normalleştirme, bir girdi değişkeni için ortalama değerin bu değerlerden çıkarılmasıdır. Girdi değişkeni sıfırdan yeni bir ortalama değer ile sonuçlanan değişkendir. Bu yöntemlerin her ikisini de uygulamak için giriş değerlerini:

$$x_i := \frac{x_i - \mu_i}{s_i}$$

Burada μi, özellik (i) için tüm değerlerin ortalamasıdır ve si, değer aralığıdır (maks – min) veya si standart sapmadır.

Standardizasyonun (veya Z Puanı Normalizasyonunun) sonucu, özelliklerin standart bir normal dağılım özelliklerine sahip olacak şekilde yeniden ölçeklendirilmesidir.

$$\mu = 0$$
$$\sigma = 1$$

Burada μ ortalama ve σ standart sapma; Örneklerin standart puanları (z puanları) şu şekilde hesaplanır:

$$z = \frac{x - \mu}{\sigma}$$

Özelliklerin standart sapması 1 ile 0 civarında ortalanacak şekilde standartlaştırılması, sadece farklı birimleri olan ölçümleri karşılaştırdığımızda değil aynı zamanda birçok makine öğrenme algoritması için genel bir gereklilik olarak da önemlidir. Dereceli azalmayı önemli bir örnek olarak düşünebiliriz (lojistik regresyonda, SVMlerde, sinir ağlarında vb. sıklıkla kullanılan bir optimizasyon algoritmasıdır); Özellikler farklı ölçeklerde olmakla birlikte, ağırlık değerleri güncelleme işleminde wj özellik değerleri geçerli olduğundan bazı ağırlıklar diğerlerinden daha hızlı güncellenebilir.

$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j} = \eta \sum_i (t^{(i)} - o^{(i)}) x_j^{(i)},$$

Böylece

$$Wj := wj + \Delta wj$$

3.4.2 Dereceli Azalma Çeşitleri

3.4.2.1 Dereceli Azalma Ne Zaman Doğru Çalışır?

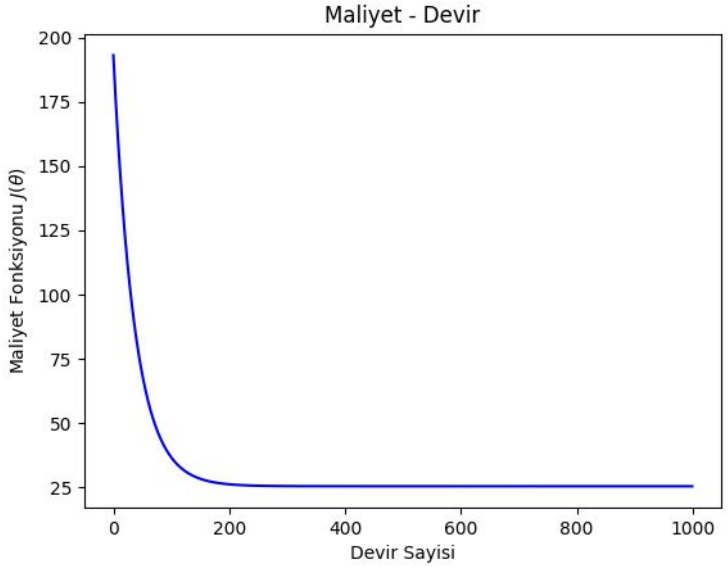
Derecel Azalma çeşitlerine bakmadan önce dereceli azalmanın ne zaman doğru çalıştığını anlayalım. Bir önceki örneğimizde maliyet fonksiyonu grafiğimizi çizip, dereceli azalmanın doğru çalıştığını söylemiştik. Kaldığımız yerden devam edelim. Bu örneğimizde öğrenme oranı alfa değerini rastgele belirledik; 0.0001. Bu sayı bize neyi ifade ediyor, evet biliyoruz optimum ağırlığa doğru ne kadar hızlı veya yavaş hareket edeceğimizi belirtiyor, ancak neden alfa'ya bu değeri verdik. Doğru değeri verdiğimizi nereden bilebiliriz ya da problemimizde doğru alfa değerini nasıl bildik?

Soruyu şöyle değiştirmeliyiz: Dereceli Azalma Ne Zaman Doğru Çalışır? Şimdi sorunumuz daha mantıklı hale geldi. Dereceli Azalma doğru çalıştığında alfa değerini doğru bilmişiz demek değil midir?

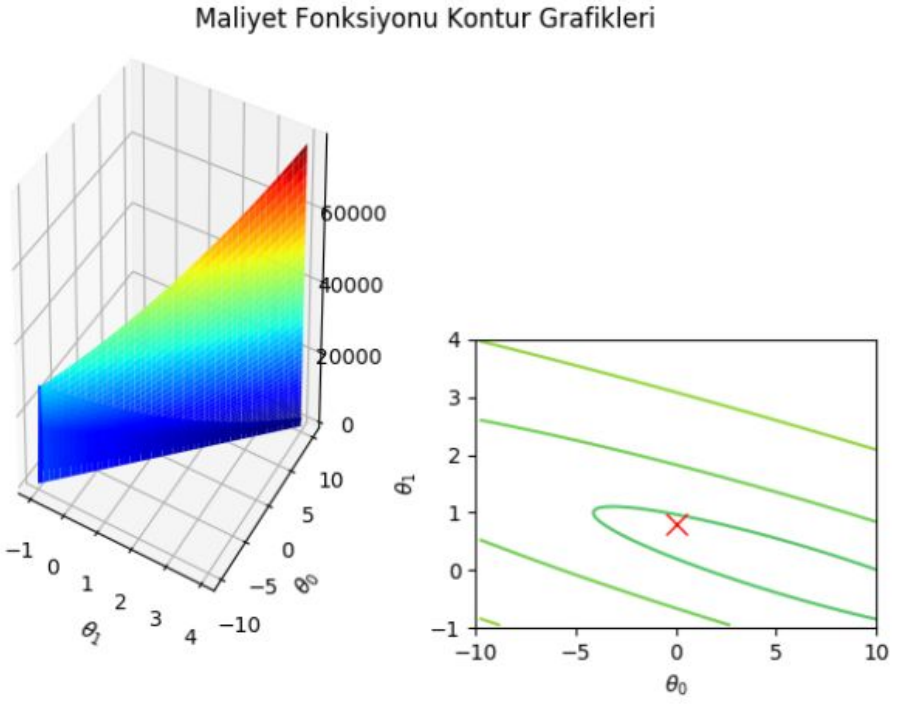
- Maliyet fonksiyonu her adımda azalıyorsa, Dereceli Azalma doğru çalışıyor demektir.

O zaman her devirde (adımda) maliyet fonksiyonuna karşılık gelen grafiğe bakarsak, dereceli azalma algoritmamızın ve bulduğumuz alfa değerinin ne denli doğru olduğunu görebiliriz.

Şimdi programımızı çalıştırdığımızda, maliyet fonksiyonu – devir sayısı grafiğini görüyoruz. Grafiği dikkatlice incelerseniz aslında alfa değerini iyi seçmiş olduğumuzu görebilirsiniz.



Programımızı sonlandırmadan önce daha önce bahsettiğimiz kontur grafiklerini hatırlayalım. Kontur grafikleri sayesinde maliyet fonksiyonu ve parametreler arasındaki ilişkiyi görselleştirebiliyorduk. Şimdi biz de bu örnek için kontur grafiklerine bir göz atalım.



- Eğer alfa değerini çok küçük seçerseniz, dereceli azalma algoritması çok yavaş hareket edecektir.
- Eğer alfa değerini çok büyük seçerseniz dereceli azalma minimum'u geçip gidebilir. Yakınsama da başarız olur hatta aradığımız minimum değerinden sapabilir.
- Dereceli azalma, öğrenme oranı alfa sabit olsa bile, yerel bir minimuma yaklaşabilir. Yerel bir minimuma yaklaştığımızda dereceli azalma otomatik olarak daha küçük adımlar atacaktır. Yani, zamanla alfa'nın azaltılmasına gerek yoktur.

3.4.2.2 Batch Gradient

Dereceli Azalma konusunda başka kaynaklara göz attıysanız, birçok farklı çeşidinin olduğunu görmüşsünüzdür. Şimdi Batch Gradient Descent ile başlayalım. Batch'in türkçe karşılığının bir defada alınan miktar olduğunu biliyoruz. Batch Gradient Descent'e biz Toplu Dereceli Azalma diyeceğiz.

- Toplu Dereceli Azalma, dereceli azalma algoritmasının her bir adımında (devirinde) bütün eğitim örneklerini kullanır.

Yaptığımız örnek, toplu dereceli azalma'ya bir örnektir. Elimizde var olan bütün öngörücü değişkenlerini kullanarak algoritmayı implement ettik. Formulümüze tekrar bakalım ve hatırlayalım:

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Bütün eğitim örneklerini kullanmak demek, yukarıya baktığınızda, maliyet fonksiyonundaki toplamda yer alan m değerini göstermektedir. Biz m değerini maliyeti hesaplarken m = len(x) olarak seçmiştik. Bu da elimizdeki tüm örnekleri kullandığımızı gösteriyor.

Unutmadan, Batch Gradient Descent, Vanilla Gradient Descent diye de bilinir.

Şimdi biraz daha basitçe anlatalım.

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

$$\begin{gathered} \textit{Yakınsayana Kadar Devam } \{ \\ \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \\ \} \\ j = 0 \textit{ ve } j = 1 \textit{ için} \end{gathered}$$

Yukarıdaki algoritma, Dereceli Azalmayı (Toplu Dereceli Azalmayı) gerçekleştirmek için maliyet fonksiyonu J'nin gradyanını hesaplamamız gerektiğini ve maliyet fonksiyonunun gradyanını hesaplamak için de, herbir örneğin toplamını (üstteki maliyet fonksiyonu) toplamamız gerektiğini söylüyor. Peki eğer, 5 milyon örneğe sahipsek bu durumda bu işlemi 5 milyon kez tekrar etmemiz gerekir.

Yani bizim şimdi sadece minimum düzeye doğru tek bir adım atmak için, her maliyeti 5 milyon kez hesaplamamız mı gerekiyor?

3.4.2.3 Stochastic Gradient

Tabiki böyle yapmadıklarını biliyorsunuz. Böyle bir durumda, her adımda her örneği kullanmak bize hem zaman açısından hem de yer açısından çok büyük olumsuzluklar getirecektir. Evet anladınız sanırım. Bütün örnekleri kullanmak zorunda mıyız? Değiliz. Sırada Stochastic Gradient Descent yani Olasılıksak Dereceli Azalma var.

- Olasılıksak Dereceli Azalma, bütün örneklerin maliyet eğrisinin toplamını kullanmak yerine, her yinelemede bir örnekteki maliyet eğimini kullanır.

Olasılıksal Dereceli Azalma'da dikkat edilmesi gereken bir kaç nokta var,

- Döngüden önce, eğitim örneklerini rasgele karıştırmanız gerekir.
- Her seferinde yalnızca bir örnek kullandığı için, minimum değere giden yol, toplu dereceli daha gürültülüdür. Ancak, bize aynı minimum değeri, kısa bir eğitim süresinde verdiği sürece sorun değildir.

3.4.2.4 Mini - Batch Gradient

Son olarak Mini Batch Gradient Descent'den bahsedelim.

- Mini Toplu Dereceli Azalma, her yinelemede toplam m örnek içerisinden n örnek seçerek kullanır. (n<m)

3.4.3 Dereceli Azalma Optimizasyon Algoritmaları

Şimdi daha derine inelim. Dereceli azalma çeşitleri ve optimizasyon algoritmalarından bahsedelim.

Dereceli Azalma, optimizasyonu gerçekleştiren en popüler algoritmalardan biridir. Dereceli Azalma (Gradyan İnişi) üzerinde bu kadar durmamızın nedeni sadece doğrusal regresyon problemlerinde değil başta sinir ağları olmak üzere birçok makine öğrenmesi probleminde kullanılmasıdır. Aynı zamanda, birçok derin öğrenme kütüphanesi (lasagne, caffe ve keras), dereceli azalmayı optimize etmek için çeşitli algoritmaların uygulamalarını içerir. Bununla birlikte, bu algoritmalar, çoğu zaman kara kutu iyileştirici (black-box optimizers), olarak kullanılır. “Kara Kutu” optimizasyonu, bir optimizasyon algoritmasının sözde bir kara kutu arabirimi yoluyla bir hedef fonksiyonu optimize etmesi yani en aza indirgemesi beklendiği bir problem durumuna işaret eder. Optimizasyonun amacı, önceden tanımlanmış bir zaman içinde mümkün olan en iyi f(x) değerini bulmaktır.

Dereceli Azalma, bir modelin parametreleri(beta/teta) tarafından parametrelendirilen bir hedef fonksiyonu(maliyet fonksiyonunu) J(θ)’yı en aza indirmenin bir yoludur. Başka bir deyişle, bir vadiye varana kadar yokuş aşağı nesnel fonksiyon tarafından yaratılan yüzeyin eğim yönünü takip etmektir.

Dereceli Azalmanın üç farklı çeşidi vardır; bunlar, hedef fonksiyonun gradyanını(derecesini) hesaplamak için ne kadar veri kullandığımız konusunda farklılık gösterir. Veri miktarına bağlı olarak, parametre güncellemesinin doğruluğu ile bir güncelleme gerçekleştirmek için gereken süre arasında bir denge kurarız.

Toplu Dereceli Azalma (Batch Gradient Descent)

Vanilya Dereceli Azalması olarak da bilinir. Maliyet fonksiyonunun dereceli azalmasını yani gradyan inişini tüm eğitim verisi seti için hesaplar.

Sadece bir güncelleme gerçekleştirmek için tüm veri kümesinin dereceli azalmasını hesaplamamız gerektiğinden, toplu eğim inişi çok yavaş olabilir ve belleğe sığmayan veri kümeleri için de zor olabilir.

Olasılıksal Dereceli Azalma (Stochastic Gradient Descent)

Stochastic Gradient Descent, Batch Gradient Descent’in aksine her eğitim örneği x(i) ve etiket y(i) için bir parametre güncellemesi gerçekleştirir.

Olasılıksal Dereceli Azalma, bir kerede bir güncelleme gerçekleştirir. Bu nedenle genellikle daha hızlıdır ve çevrimiçi öğrenme için de kullanılabilir.

Mini-Toplu Dereceli Azalma (Mini-Batch Gradient Descent)

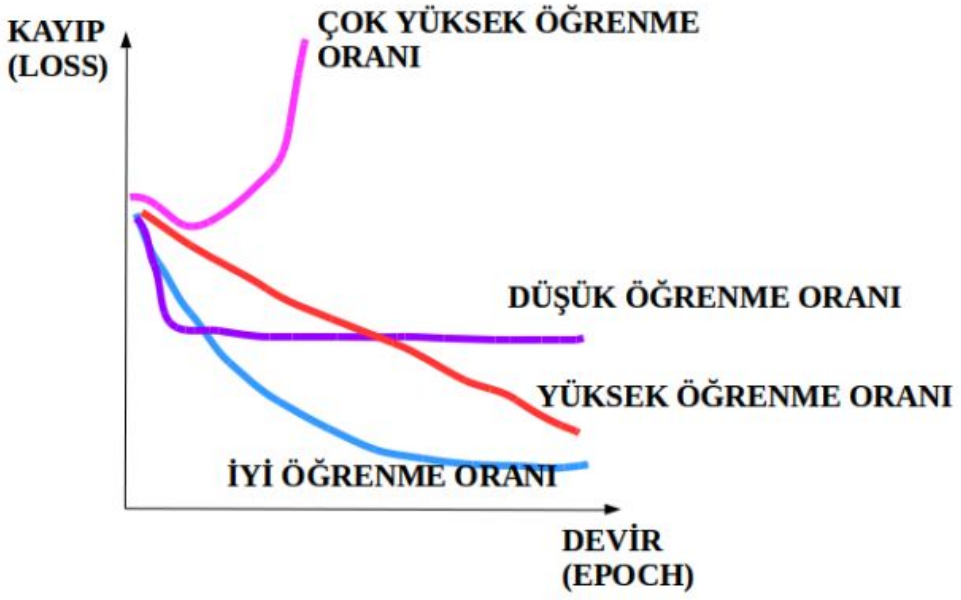
Mini-Batch Gradient Descent diğer dereceli azalma algoritmalarının en iyisini alıyor ve her bir mini paket için bir güncelleme gerçekleştiriyordur.

Eğitim setinden n boyutta paketler seçmek parametre güncellemelerinin varyansını azaltır; bu da daha kararlı yakınsama sağlayabilir. Mini toplu

dereceli azalma çok verimlidir. Yaygın mini paket boyutları 50 ile 256 arasında değişir, ancak farklı uygulamalar için değişiklik gösterebilir.

Dereceli Azalma yöntemlerinin sebep olduğu birçok sorun olabilir. Doğru yakınsama için öğrenme oranını seçmek zor olabilir. Bütün ağırlıklar aynı anda güncellenmeyebilir. Bu problemlere çözüm olarak birçok dereceli azalma optimizasyon algoritması kullanılabilir. Bu algoritmalardan bahsetmeden önce şuna bir bakalım. Dereceli Azalma Algoritmasının doğru çalıştığını anlamak için devir sayısı ile maliyet fonksiyonu arasındaki grafiğe bakıyoruz. Hangi grafik doğrudur? Öğrenme Oranını nasıl seçmeliyiz, doğru öğrenme oranını(alfayı) seçebilmiş miyiz? Bu soruların cevabına aşağıdaki grafiği inceleyerek ulaşabilirsiniz. Düşük öğrenme oranına benzer bir grafik elde ederseniz anlarsınız ki alfayı büyütmelisiniz ve ya çok yüksek öğrenme oranına benzer bir eğri elde ettiyseniz alfa değerini azaltmalısınız.

Unutmadan, maliyet fonksiyonuna kayıp fonksiyonu (loss function) da denir.



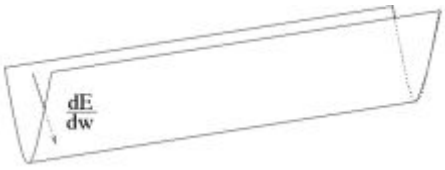
Dereceli Azalma Optimizasyon Algoritmaları
Bu algoritmalardan bazıları:

- Momentum
- Nesterov accelerate gradient
- Adagrad
- Adadelta
- Rmsprop
- Adam

Zamanla hepsinden bahsedebiliriz. Şimdi Momentum ile başlayalım.

3.4.3.1 Momentum

Maliyet fonksiyonu uzun ve dar bir vadi şeklinde ise, dereceli azalma vadi duvarlarının aşağısına hızlı bir şekilde ilerlerken vadi tabanında ise çok yavaş ilerler.



Özellikle Olasılıksal Dereceli Azalmalarda yüksek varyanslı salınımlar yakınsamaya ulaşmayı zorlaştırıyor. Bu nedenle Momentum denilen bir teknik icat edilmiştir.

Bu Momentum Algoritması ile ilgili yönde gezinerek Dereceli Azalmayı hızlandırdı ve ilgisiz yönlerde salınımı yumuşatmıştır.

Yani yaptığı tek şey, bir γ güncelleme vektörüne(momentum vektörü/terimi) geçmiş adımın güncelleme vektörünü eklemektir. Bundan sonra da parametreleri günceller.

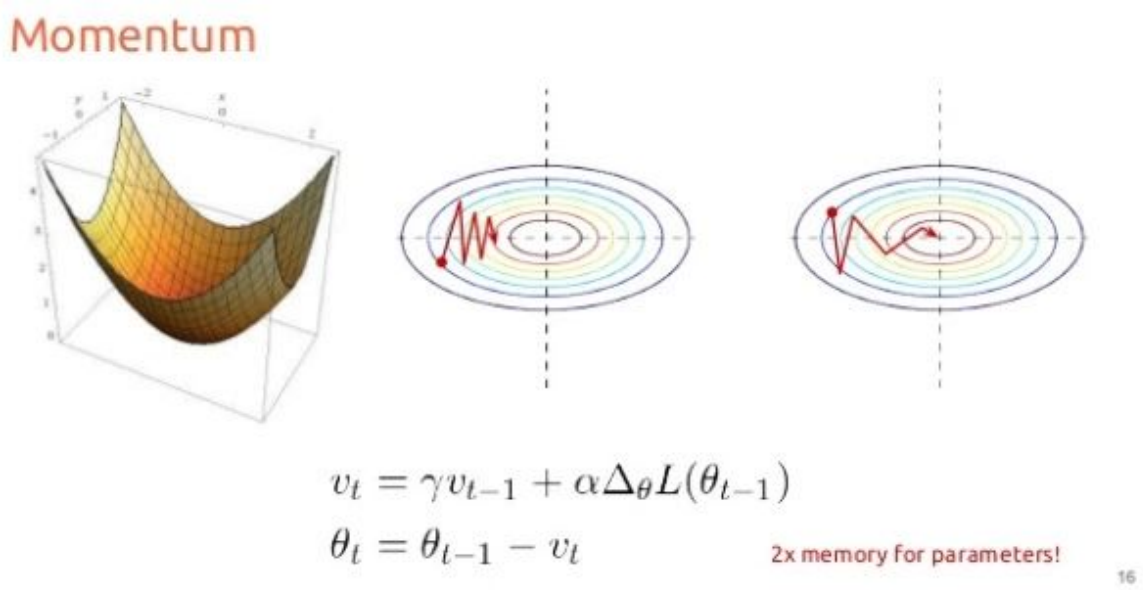
$$v_t = \gamma v_{t-1} - \alpha \nabla_\beta J(\beta)$$
$$\beta = \beta + v_t$$

γ momentum terimi genellikle 0.9 veya daha büyük bir değere ayarlanır.

En başta belirtildiği gibi Maliyet fonksiyonu uzun ve dar bir vadi şeklinde ise, dereceli azalma vadi duvarlarının aşağısına hızlı bir şekilde ilerlerken vadi tabanında ise çok yavaş ilerler. Parametreleri, önceki güncellemenin ve dereceli azalmanın güncellemesinin bir birleşimi kullanılarak güncelleyerek bu sorunu hafifletebiliriz. Bu ağırlıklara ivme kazandırmaya benzer.

Böylece Momentum güncellemesi ile, momentum terimi tutarlı bir dereceye sahip herhangi bir yönde hız oluşturacaktır.

Momentum kullanarak, gereksiz parametre güncellemeleri azaltılır ve bu da daha hızlı ve istikrarlı yakınsama ve azalmış salınımlara neden olur.



3.4.4 Baştan Sona Çok Değişkenli Lineer Regresyon Örneği

Önce veri setini anlayalım, sonra problemimizin tanımını yapacağız. Yine ev fiyatları tahmini gerçekleştireceğiz. Aslında veri setini eğitim ve test verileri olarak ayırıp, modeli eğitip sonra test etmiyoruz. Ancak yaptığımız şey aslında aynı. Elimizdeki verilere en uygun modeli oluşturmaya çalışıyoruz. Öyleyse başlayalım. İlk kolon evin büyüklüğünü, ikinci kolon evdeki yatak odası sayısını ve son kolon ise evin fiyatını göstermektedir. Bir önceki örnekte evin fiyatını tahmin etmek için sadece bir değişkeni, özelliği kulanmıştık. Şimdi elimizde iki tane öngörücü değişken var; evin büyüklüğü ve yatak odası sayısı. Bir önceki örneğimiz tek değişkenli lineer regresyon problemiydi, şimdi de karşımızda çok değişkenli lineer regresyon problemi var.

Daha önce, tek değişkenli bir regresyon probleminde dereceli azalmayı uyguladık. Şimdi tek fark, X matrisinde yani girdimizde bir başka özellik daha olmasıdır. Hipotez fonksiyonu ve toplu dereceli azalmada güncelleme kuralları değişmeden kalır. Bir önceki örnekte fark ettiyseniz özelliklerin sayı değerleri çok büyüktü. Bunların kontur grafiklerini çizmeye çalıştıysanız eğer görmek istediğiniz grafiklere ulaşamamışsınızdır. Bunun sebebi özelliklerin normalize edilmeyişidir. Özellikleri ölçeklediğimiz zaman dereceli azalmanın çok daha hızlı bir şekilde yakınsamaya neden olduğunu görürsünüz ve böyle maliyet fonksiyonu – parametrelerin ilişkisini gösteren grafikler istediğiniz gibi olur. Neden özellikleri ölçeklendirmeliyiz, bunun sebebi çok açık, dereceli azalma çalıştığında bazı ağırlıkları diğerlerinden daha hızlı güncelleyecektir. Giriş değerlerinin her birini kabaca aynı aralıkta tutarak eğim açılımı hızlandırabilir. Bunu yapmanın iki yolu vardır.

Özellik ölçeklendirme, giriş değerlerini giriş değişkeninin aralık (maksimum değer eksi en düşük değerinden) aralığına bölerek yeni bir bir aralığına neden olur. **Ortalama normalleştirme**, bir girdi değişkeni için ortalama değerin bu değerlerden çıkarılmasıdır. Girdi değişkeni sıfırdan yeni bir ortalama değer ile sonuçlanan

değişkendir.

Çok değişkenli durumda, maliyet fonksiyonu aşağıdaki vektörize edilmiş biçimde yazılabilir:

$$J(\theta) = \frac{1}{2m} (X\theta - \vec{y})^T (X\theta - \vec{y})$$

X ve y vektörleri şöyle:

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}.$$

Aslında iki formun da eşdeğer olduğunu fark etmişsinizdir.

Yeni örneğimize geri dönelim. Çok değişkenli demek, çıktımızı tahmin etmek için kullandığımız girdi özelliklerimizin birden fazla olması anlamına gelir. Problemin ilk versiyonunda X = ev büyüklüğü ve y = ev fiyatını göstermekteydi. Şimdi yeni örneğimizde çıktıyı tahmin etmeye çalıştığımız X değişkenini şu hale geliyor, x1 = ev büyüklüğü, x2 = yatak odası sayısı.

Yeni Hipotezimiz de:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

X değişkenleri arttırdıkça hipotez şöyle değişebilir:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

Bu problemde de Dereceli Azalmayı Kullanacağız.

Maliyet Fonksiyonumuz:

$$J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Dereceli Azalma Algoritmamız:

$$
\begin{aligned}
&\textit{Tekrar Et } \{ \\
&\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n) \\
&\textit{ayni anda guncelle } j = 0, \ldots, n \}
\end{aligned}
$$

Maliyet fonksiyonunun kısmi türevini açarsak:

$$
\begin{aligned}
&\textit{Tekrar Et } \{ \\
&\theta_j := \theta_j - \alpha \frac{1}{m} \sum_i^m (h_\theta(x^i) - y^i) x_j^i \\
&\textit{ayni anda guncelle } j = 0, \ldots, n \}
\end{aligned}
$$

Her bir j için açarsak:

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_i^m (h_\theta(x^i) - y^i) x_0^i$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_i^m (h_\theta(x^i) - y^i) x_1^i$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_i^m (h_\theta(x^i) - y^i) x_2^i$$

Şimdi size her yer formul olmuş ve her şey çok karışık, hiçbir şey anlamıyormuşsunuz gibi geliyorsa, dikkatinizi toplayın ve tekrar bakın.

Daha rahat görebilmeniz için bir önceki örneğimizdeki formuller ile karşılaştıralım.

Hipotez:

$$
\begin{aligned}
&\textit{Eski Hipotez} : h_\theta(x) = \theta_0 + \theta_1 x \\
&\textit{Yeni Hipotez} : h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 + \theta_3 x + \theta_4 x_4
\end{aligned}
$$

Maliyet Fonksiyonu:

$$\textit{Eski Maliyet Fonksiyonu} : J(\theta_0, \theta_1) = \frac{1}{2m} \sum_i^m (h_\theta(x^i) - y^i)^2$$

$$\textit{Yeni Maliyet Fonksiyonu} : J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_i^m (h_\theta(x^i) - y^i)^2$$

Dereceli Azalma:

$$Eski\ Dereceli\ Azalma : \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)\ \ j = 0\ ve\ j = 1\ icin$$

$$Yeni\ Dereceli\ Azalma : \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)\ \ j = 0, \ldots, n\ icin$$

Bir de böyle bakın:

$$Eski\ Hipotez : h_\theta(x) = \theta_0 + \theta_1 x$$

$$Yeni\ Hipotez : h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 + \theta_3 x + \theta_4 x_4$$

$$Eski\ Maliyet\ Fonksiyonu : J(\theta_0, \theta_1) = \frac{1}{2m} \sum_i^m (h_\theta(x^i) - y^i)^2$$

$$Yeni\ Maliyet\ Fonksiyonu : J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_i^m (h_\theta(x^i) - y^i)^2$$

$$Eski\ Dereceli\ Azalma : \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)\ \ j = 0\ ve\ j = 1\ icin$$

$$Yeni\ Dereceli\ Azalma : \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)\ \ j = 0, \ldots, n\ icin$$

Değişen tek şey θ sayıları !

Şimdi örneğimize geri dönelim. Hatırlarsanız ilk kolon evin büyüklüğünü, ikinci kolon evdeki yatak odası sayısını ve son kolon ise evin fiyatını göstermektedir. İlk iki kolondaki verileri kullanarak son kolondaki verilere en iyi uyacak doğruyu çizeceğiz. Bunun için hipotez fonksiyonuna ihtiyacımız var.

İki değişkenimiz olduğu için Hipotezimiz:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Amaç: Hipotezi Bulmak

1. Hipotezi Bulmak İçin Tetaları(parametreleri) Bulmamız Gerekiyor.
2. Tetaları Bulmak için Maliyet Fonksiyonunu Kullanacağız.
    a. Maliyeti En Aza İndirgemeliyiz.
        i. En az Hata ile parametreleri bulursak, en iyi hipotezi buluruz.
        ii. Maliyeti Hesaplarken (yani hatayı), Squared Error, Mean Squarred Error(MSE), Mean Absolute Error(MAE), Root Mean Squared Error(RMSE) Metriklerini Kullanabiliriz.
3. Maliyeti En Aza indirgemek için Dereceli Azalmayı Kullacağız.
    a. Özellikleri Ölçeklendirmeliyiz.
        i. Dereceli Azalmanın Bazı Ağırlıkları Daha Önce Güncellemesini Önlemek için.
    b. Dereceli Azalmayı Üç Ayrı Şekilde Gerçekleştirebiliriz:
        i. Batch-Gradient Descent
        ii. Mini-Batch Gradient Descent
        iii. Mini-Batch Gradient Descent with Momentum Optimization

3.4.4.1 Toplu Dereceli Azalma ile Örnek

calistir.py

```
# coding=utf-8
""" Batch Gradient Descent
Toplu Dereceli Azalma"""

import pandas as pd
import numpy as np

from dereceli_azalma import dereceli_azalma
from ozellik_normalizasyonu import ozellik_normalizasyonu
from ciz import maliyet_devir_grafigi_ciz

u""" 1- Veri Setinin Yuklenmesi """
print '1- Veri Seti Yukleniyor ...'
# Veri Setini Csv Dosyasindan Okuma
veri_seti = pd.read_csv('regresyon_veri_2.txt', delimiter=',',
header=None, names=['Buyukluk', 'Oda', 'Fiyat'])

# Tahmin Etmeye Calisacagimiz Verileri Okuma
y = veri_seti['Fiyat'].as_matrix()
del veri_seti['Fiyat']

# Toplam Girdi Sayisi
m = len(y)
```

```
# Ongorucu Veriler, ilk iki degisken: buyukluk ve oda sayisi
X_veri = veri_seti.as_matrix()

u""" 2- Ozellik Normalizasyonu """
print '2- Ozellik Normalizasyonu Yapiliyor ...'
x, ortalama, standart_sapma = ozellik_normalizasyonu(X_veri)

u""" 3- Kesme Degerinin Eklenmesi"""
print '3- Kesme Degeri Ekleniyor ...'
X = np.ones(shape=(m, (len(X_veri[0]) + 1)))
X[:, 1:(len(X_veri[0]) + 1)] = x

# Tahmin Edilecek Hedef Degiskeni Atama ve Boyutunu Ayarlama, yani
vektor haline donusturme
y.shape = (m, 1)

u""" 4- Ogrenme Orani ve Devir Sayisinin Ayarlanmasi"""
print '4- Ogrenme Orani ve Devir Sayisi Ayarlaniyor ...'
toplam_devir = 1000
alfa = 0.01

print 'Devir Sayisi ', toplam_devir
print 'Alfa ', alfa

beta_mse = np.zeros((len(X[0]), 1))
beta_mae = np.zeros((len(X[0]), 1))
beta_sqrt_err = np.zeros((len(X[0]), 1))
beta_rmse = np.zeros((len(X[0]), 1))

u""" 5.1 - Dereceli Azalmanin Hesaplanmasi"""
print '5.1- Dereceli Azalma Hesaplaniyor ...'
beta_mse, maliyet_mse = dereceli_azalma(X, y, beta_mse, alfa,
toplam_devir, 'mse')
beta_mae, maliyet_mae = dereceli_azalma(X, y, beta_mae, alfa,
toplam_devir, 'mae')
beta_sqrt_err, maliyet_sqrt_err = dereceli_azalma(X, y,
beta_sqrt_err, alfa, toplam_devir, 'squared_error')
beta_rmse, maliyet_rmse = dereceli_azalma(X, y, beta_rmse, alfa,
toplam_devir, 'rmse')

u""" 6- Dereceli Azalmanin Dogru Calisip Calismadiginin
Kontrolu"""
print '6- Maliyet Fonksiyonu - Devir Sayisi Grafigi Kaydediliyor
...'
maliyet_devir_grafigi_ciz(toplam_devir, maliyet_mse, maliyet_mae,
maliyet_sqrt_err, maliyet_rmse,
                          baslik='Dereceli Azalma')
```

```
u""" 7- Sonuc"""
print 'Hesaplanan Beta (MSE) ', beta_mse
print 'Hesaplanan Beta (MAE) ', beta_mae
print 'Hesaplanan Beta (SQRT ERR) ', beta_sqrt_err
print 'Hesaplanan Beta (RMSE) ', beta_rmse

print '7- Hipotez (SQRT ERR\'e gore)'
print "J(theta) = " + str(beta_sqrt_err[0][0]) + " + " +
str(beta_sqrt_err[1][0]) + " x0 + " + str(
    beta_sqrt_err[2][0]) + " x1"

print '8- Ev Fiyati Tahmini Yapalim ... (SQRT ERR\'e gore)'

# X Verilerinde Normalizasyon Yaptigimiz Icin Direkt Kullanamayiz.
# Tahmin etmek icin X ongorucu degiskenlerimizi buldugumuz
# ortalama ve standart sapmalara gore normalize edip kullaniyoruz
x0 = 1
x1 = (1650 - ortalama[0]) / standart_sapma[0]
x2 = (3 - ortalama[1]) / standart_sapma[1]
tahmin = np.dot([x0, x1, x2], beta_sqrt_err)

print 'Ev Buyuklugu 1650, 3 yatak odali ise Fiyati : ', tahmin[0]

x0 = 1
x1 = (1604 - ortalama[0]) / standart_sapma[0]
x2 = (3 - ortalama[1]) / standart_sapma[1]
tahmin = np.dot([x0, x1, x2], beta_sqrt_err)

print 'Ev Buyuklugu 1604, 3 yatak odali ise Fiyati : ', tahmin[0]

x0 = 1
x1 = (1000 - ortalama[0]) / standart_sapma[0]
x2 = (2 - ortalama[1]) / standart_sapma[1]
tahmin = np.dot([x0, x1, x2], beta_sqrt_err)

print 'Ev Buyuklugu 1000, 2 yatak odali ise Fiyati : ', tahmin[0]
```

dereceli_azalma.py

```
# coding=utf-8
""" Gradient Descent
 Dereceli Azalma """

import numpy as np
from maliyet_hesapla import squarred_error_maliyet_hesapla,
```

```
mae_maliyet_fonksiyonu_hesapla, \
    mse_maliyet_fonksiyonu_hesapla, \
    rmse_maliyet_fonksiyonu_hesapla


def dereceli_azalma(x, y, beta, alfa, devir_sayisi,
maliyet_fonksiyonu):
    """

    :param x: ozellik degiskenleri
    :param y: hedef degisken
    :param beta: parametreler
    :param alfa: ogrenme orani
    :param devir_sayisi: devir sayisi
    :param maliyet_fonksiyonu: hesaplanacak maliyet fonksiyonu
turu
    :return: beta ve maliyet fonksiyonu
    """
    maliyet_J = np.zeros(shape=(devir_sayisi, 1))

    for devir in range(devir_sayisi):

        m = y.size

        tahminler = x.dot(beta)
        beta_size = beta.size

        for i in range(beta_size):
            X_i = x[:, i]
            X_i.shape = (m, 1)

            beta[i][0] = beta[i][0] - alfa * (1.0 / m) *
((tahminler - y) * X_i).sum()

        if maliyet_fonksiyonu == 'mse':
            maliyet_J[devir, 0] =
mse_maliyet_fonksiyonu_hesapla(x, y, beta)
        if maliyet_fonksiyonu == 'mae':
            maliyet_J[devir, 0] =
mae_maliyet_fonksiyonu_hesapla(x, y, beta)
        if maliyet_fonksiyonu == 'rmse':
            maliyet_J[devir, 0] =
rmse_maliyet_fonksiyonu_hesapla(x, y, beta)
        if maliyet_fonksiyonu == 'squared_error':
            maliyet_J[devir, 0] =
squarred_error_maliyet_hesapla(x, y, beta)
    return beta, maliyet_J
```

maliyet_hesapla.py

```python
# coding=utf-8
""" Compute Cost Function
 Maliyet Fonksiyonunu Hesapla"""

import numpy as np


def squarred_error_maliyet_hesapla(x, y, teta):
    """

    :param x: Ongorucu Degisken, Ev Buyuklukleri
    :param y: Hedef Degisken, Ev Fiyatlari
    :param teta: parametre
    :return: maliyet
    """
    m = len(x)
    toplamin_acilimi = np.power(((x * teta.T) - y), 2)
    maliyet = np.sum(toplamin_acilimi) / (2 * m)
    return maliyet


def mae_maliyet_fonksiyonu_hesapla(x, y, teta):
    """

    :param x: Ongorucu Degisken, Ev Buyuklukleri
    :param y: Hedef Degisken, Ev Fiyatlari
    :param teta: parametre
    :return: maliyet
    """
    m = len(x)
    toplamin_acilimi = np.abs(((x * teta.T) - y))
    maliyet = np.sum(toplamin_acilimi) / m
    return maliyet


def mse_maliyet_fonksiyonu_hesapla(x, y, teta):
    """

    :param x: Ongorucu Degisken, Ev Buyuklukleri
    :param y: Hedef Degisken, Ev Fiyatlari
    :param teta: parametre
    :return: maliyet
    """
```

```
    m = len(x)
    toplamin_acilimi = np.power(((x * teta.T) - y), 2)
    maliyet = np.sum(toplamin_acilimi) / m
    return maliyet


def rmse_maliyet_fonksiyonu_hesapla(x, y, teta):
    """

    :param x: Ongorucu Degisken, Ev Buyuklukleri
    :param y: Hedef Degisken, Ev Fiyatlari
    :param teta: parametre
    :return: maliyet
    """
    m = len(x)
    toplamin_acilimi = np.power(((x * teta.T) - y), 2)
    maliyet = np.sqrt(np.sum(toplamin_acilimi) / m)
    return maliyet
```

ozellik_normalizasyonu.py

```
# coding=utf-8
""" Feature Normalization
 Ozellik Normalizasyonu"""
import numpy as np


def ozellik_normalizasyonu(x):
    """

    :param x: normallestirilecek ozellikler
    :return: normalize olmus X, ortalamasi ve sapmasi
    """
    butun_ortalamalar = []
    butun_standart_sapmalar = []
    X_normalize = x

    n_c = x.shape[1]
    for i in range(n_c):
        m = np.mean(x[:, i])
        s = np.std(x[:, i])

        butun_ortalamalar.append(m)
        butun_standart_sapmalar.append(s)
        X_normalize[:, i] = (X_normalize[:, i] - m) / s
```

```
    return X_normalize, butun_ortalamalar, butun_standart_sapmalar
```

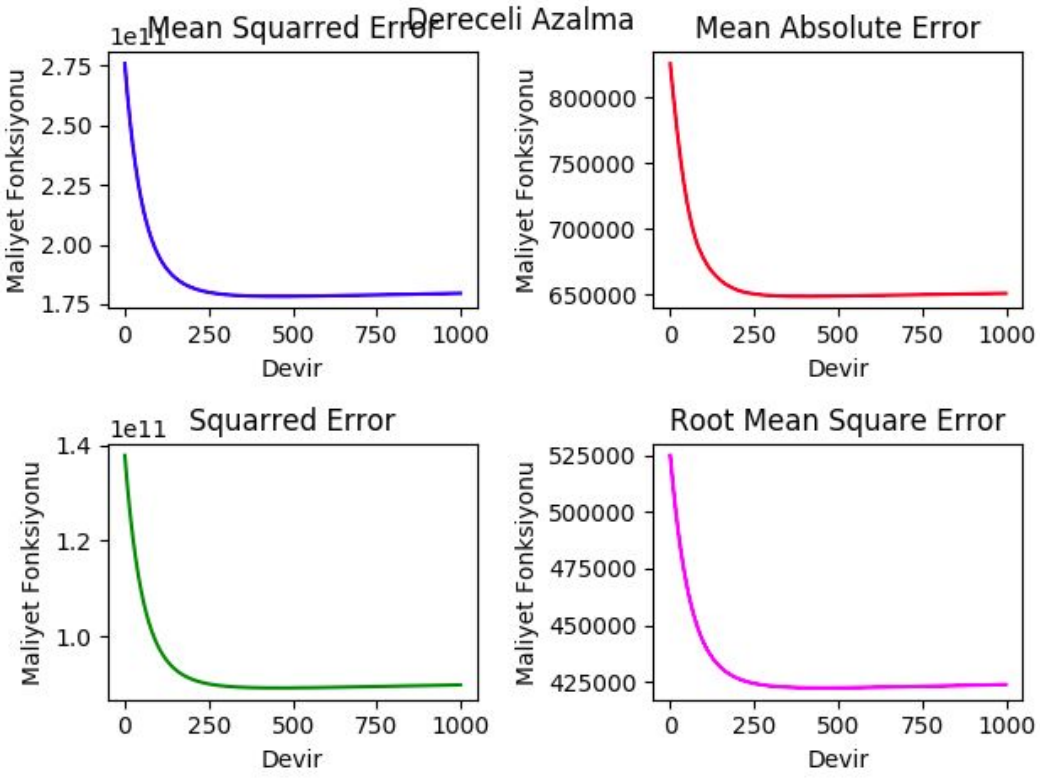
ciz.py

```
# coding=utf-8
""" Plotting Data
Veri Setinin Grafiginin Cizilmesi"""

import matplotlib.pyplot as plt
import numpy as np


def maliyet_devir_grafigi_ciz(toplam_devir, mse, mae,
squared_error, rmse, baslik):
    """

  :param toplam_devir: Toplam Devir Sayisi
    :param mse: Mean Squared Error
    :param mae: Mean Absolute Error
    :param squared_error: Squared Error
    :param rmse: Root Mean Squared Error
    :param baslik: Grafigin Basligi
    """
    fig = plt.figure(1)
    fig.suptitle(baslik)

    ax1 = fig.add_subplot(221)
    ax1.set_title("Mean Squarred Error")
    ax1.set_xlabel("Devir")
    ax1.set_ylabel("Maliyet Fonksiyonu")
    ax1.plot(np.arange(toplam_devir), mse, 'blue')

    ax2 = fig.add_subplot(222)
    ax2.set_title("Mean Absolute Error")
    ax2.set_xlabel("Devir")
    ax2.set_ylabel("Maliyet Fonksiyonu")
    ax2.plot(np.arange(toplam_devir), mae, 'red')

    ax3 = fig.add_subplot(223)
    ax3.set_title("Squarred Error")
    ax3.set_xlabel("Devir")
    ax3.set_ylabel("Maliyet Fonksiyonu")
    ax3.plot(np.arange(toplam_devir), squared_error, 'green')
    plt.tight_layout()

    ax4 = fig.add_subplot(224)
```

```
    ax4.set_title("Root Mean Square Error")
    ax4.set_xlabel("Devir")
    ax4.set_ylabel("Maliyet Fonksiyonu")
    ax4.plot(np.arange(toplam_devir), rmse, 'magenta')
    plt.tight_layout()

    fig.savefig(str(baslik).strip() + ".png")
```

Programın çıktısı:

```
1- Veri Seti Yukleniyor ...
2- Ozellik Normalizasyonu Yapiliyor ...
3- Kesme Degeri Ekleniyor ...
4- Ogrenme Orani ve Devir Sayisi Ayarlaniyor ...
Devir Sayisi  1000
Alfa  0.01
5.1- Dereceli Azalma Hesaplaniyor ...
6- Maliyet Fonksiyonu - Devir Sayisi Grafigi Kaydediliyor ...
Hesaplanan Beta (MSE)  [[267101.52754383]
 [120807.12427982]
 [ 61359.72765548]]
Hesaplanan Beta (MAE)  [[267101.52754383]
 [120807.12427982]
 [ 61359.72765548]]
Hesaplanan Beta (SQRT ERR)  [[267101.52754383]
 [120807.12427982]
 [ 61359.72765548]]
Hesaplanan Beta (RMSE)  [[267101.52754383]
 [120807.12427982]
 [ 61359.72765548]]
7- Hipotez (SQRT ERR'e gore)
J(theta) = 267101.5275438266 + 120807.12427981633 x0 +
61359.72765548302 x1
8- Ev Fiyati Tahmini Yapalim ... (SQRT ERR'e gore)
Ev Buyuklugu 1650, 3 yatak odali ise Fiyati :  166771.128835
Ev Buyuklugu 1604, 3 yatak odali ise Fiyati :  162278.30148
Ev Buyuklugu 1000, 2 yatak odali ise Fiyati :  42837.7250495
```



3.4.4.2 Mini-Paket Dereceli Azalma ile Örnek

Bir önceki bölümde Çok Değişkenli Lineer Regresyon probleminde Toplu Dereceli Azalma kullanmıştık. Şimdi ise dereceli azalmayı kullanırken bütün örnekler yerine daha küçük paketler seçerek kullanalım. Buna hatırlarsanız Mini Paket Dereceli Azalma deniyordu.

calistir.py

```
# coding=utf-8
""" Main
Calistir """

import pandas as pd
import numpy as np

from dereceli_azalma import dereceli_azalma_mini_paket
from ozellik_normalizasyonu import ozellik_normalizasyonu
from ciz import maliyet_devir_grafigi_ciz

u""" 1- Veri Setinin Yuklenmesi """
print '1- Veri Seti Yukleniyor ...'
# Veri Setini Csv Dosyasindan Okuma
veri_seti = pd.read_csv('regresyon_veri_2.txt', delimiter=',',
```

```
header=None, names=['Buyukluk', 'Oda', 'Fiyat'])

# Tahmin Etmeye Calisacagimiz Verileri Okuma
y = veri_seti['Fiyat'].as_matrix()
del veri_seti['Fiyat']

# Toplam Girdi Sayisi
m = len(y)

# Ongorucu Veriler, ilk iki degisken: buyukluk ve oda sayisi
X_veri = veri_seti.as_matrix()

u""" 2- Ozellik Normalizasyonu """
print '2- Ozellik Normalizasyonu Yapiliyor ...'
x, ortalama, standart_sapma = ozellik_normalizasyonu(X_veri)

u""" 3- Kesme Degerinin Eklenmesi"""
print '3- Kesme Degeri Ekleniyor ...'
X = np.ones(shape=(m, (len(X_veri[0]) + 1)))
X[:, 1:(len(X_veri[0]) + 1)] = x

# Tahmin Edilecek Hedef Degiskeni Atama ve Boyutunu Ayarlama, yani
vektor haline donusturme
y.shape = (m, 1)

u""" 4- Ogrenme Orani ve Devir Sayisinin Ayarlanmasi"""
print '4- Ogrenme Orani ve Devir Sayisi Ayarlaniyor ...'
toplam_devir = 1000
alfa = 0.005

print 'Devir Sayisi ', toplam_devir
print 'Alfa ', alfa

beta_mse = np.zeros((len(X[0]), 1))
beta_mae = np.zeros((len(X[0]), 1))
beta_sqrt_err = np.zeros((len(X[0]), 1))
beta_rmse = np.zeros((len(X[0]), 1))

paket = 50

u""" 5.2 - Mini Paket Dereceli Azalmanin Hesaplanmasi"""
print '5.2- Mini Paket Dereceli Azalma Hesaplaniyor ...'
beta_mse, maliyet_mse = dereceli_azalma_mini_paket(X, y, beta_mse,
alfa, toplam_devir, paket, 'mse')
beta_mae, maliyet_mae = dereceli_azalma_mini_paket(X, y, beta_mae,
alfa, toplam_devir, paket, 'mae')
beta_sqrt_err, maliyet_sqrt_err = dereceli_azalma_mini_paket(X, y,
beta_sqrt_err, alfa, toplam_devir, paket,
```

```
'squared_error')
beta_rmse, maliyet_rmse = dereceli_azalma_mini_paket(X, y,
beta_rmse, alfa, toplam_devir, paket, 'rmse')

u""" 6- Dereceli Azalmanin Dogru Calisip Calismadiginin
Kontrolu"""
print '6- Maliyet Fonksiyonu - Devir Sayisi Grafigi Kaydediliyor
...'
maliyet_devir_grafigi_ciz(toplam_devir, maliyet_mse, maliyet_mae,
maliyet_sqrt_err, maliyet_rmse,
                          baslik='Mini Paket Dereceli Azalma')

u""" 7- Sonuc"""
print 'Hesaplanan Beta (MSE) ', beta_mse
print 'Hesaplanan Beta (MAE) ', beta_mae
print 'Hesaplanan Beta (SQRT ERR) ', beta_sqrt_err
print 'Hesaplanan Beta (RMSE) ', beta_rmse

print '7- Hipotez (SQRT ERR\'e gore)'
print "J(theta) = " + str(beta_sqrt_err[0][0]) + " + " +
str(beta_sqrt_err[1][0]) + " x0 + " + str(
    beta_sqrt_err[2][0]) + " x1"

print '8- Ev Fiyati Tahmini Yapalim ... (SQRT ERR\'e gore)'

# X Verilerinde Normalizasyon Yaptigimiz Icin Direkt Kullanamayiz.
# Tahmin etmek icin X ongorucu degiskenlerimizi buldugumuz
# ortalama ve standart sapmalara gore normalize edip kullaniyoruz
x0 = 1
x1 = (1650 - ortalama[0]) / standart_sapma[0]
x2 = (3 - ortalama[1]) / standart_sapma[1]
tahmin = np.dot([x0, x1, x2], beta_sqrt_err)

print 'Ev Buyuklugu 1650, 3 yatak odali ise Fiyati : ', tahmin[0]

x0 = 1
x1 = (1604 - ortalama[0]) / standart_sapma[0]
x2 = (3 - ortalama[1]) / standart_sapma[1]
tahmin = np.dot([x0, x1, x2], beta_sqrt_err)

print 'Ev Buyuklugu 1604, 3 yatak odali ise Fiyati : ', tahmin[0]

x0 = 1
x1 = (1000 - ortalama[0]) / standart_sapma[0]
x2 = (2 - ortalama[1]) / standart_sapma[1]
tahmin = np.dot([x0, x1, x2], beta_sqrt_err)
```

```
print 'Ev Buyuklugu 1000, 2 yatak odali ise Fiyati : ', tahmin[0]
```

dereceli_azalma.py

```
# coding=utf-8
""" Mini Batch Gradient Descent
Mini Paket Dereceli Azalma """

import numpy as np
from maliyet_hesapla import squarred_error_maliyet_hesapla,
mae_maliyet_fonksiyonu_hesapla, \
    mse_maliyet_fonksiyonu_hesapla, \
    rmse_maliyet_fonksiyonu_hesapla


def mini_paket_al(x, y, mini_paket_buyuklugu):
    """

    :param x: ozellik degiskenleri
    :param y: hedef degisken
    :param mini_paket_buyuklugu:
    :return: veri setinden secilen paket(mini batch)
    """
    mini_paketler = []

    # X, y = np.random.shuffle(X,y)

    for i in range(0, x.shape[0], mini_paket_buyuklugu):
        x_mini = x[i:i + mini_paket_buyuklugu]
        y_mini = y[i:i + mini_paket_buyuklugu]

        mini_paketler.append((x_mini, y_mini))

    return mini_paketler


def dereceli_azalma_mini_paket(x, y, beta, alfa, devir_sayisi,
paket_buyuklugu, maliyet_fonksiyonu):
    """

   :param x: ozellik degiskenleri
   :param y: hedef degisken
   :param beta: parametreler
   :param alfa: ogrenme orani
   :param devir_sayisi: devir sayisi
   :param paket_buyuklugu: secilen mini paketin buyuklugu
   :param maliyet_fonksiyonu: hesaplanacak maliyet fonksiyonu turu
```

```
    :return: beta ve maliyet fonksiyonu
    """


    maliyet_J = np.zeros(shape=(devir_sayisi, 1))

    for devir in range(devir_sayisi):

        for paketler in mini_paket_al(x, y, paket_buyuklugu):

            x = paketler[0]
            y = paketler[1]
            m = y.size

            tahminler = x.dot(beta)
            beta_size = beta.size

            for i in range(beta_size):
                X_i = x[:, i]
                X_i.shape = (m, 1)

                beta[i][0] = beta[i][0] - alfa * (1.0 / m) *
((tahminler - y) * X_i).sum()

            if maliyet_fonksiyonu == 'mse':
                maliyet_J[devir, 0] =
mse_maliyet_fonksiyonu_hesapla(x, y, beta)
            if maliyet_fonksiyonu == 'mae':
                maliyet_J[devir, 0] =
mae_maliyet_fonksiyonu_hesapla(x, y, beta)
            if maliyet_fonksiyonu == 'rmse':
                maliyet_J[devir, 0] =
rmse_maliyet_fonksiyonu_hesapla(x, y, beta)
            if maliyet_fonksiyonu == 'squared_error':
                maliyet_J[devir, 0] =
squarred_error_maliyet_hesapla(x, y, beta)
    return beta, maliyet_J
```

maliyet_hesapla.py

```
# coding=utf-8
""" Compute Cost Function
 Maliyet Fonksiyonunu Hesapla"""

import numpy as np
```

```python
def squarred_error_maliyet_hesapla(x, y, teta):
    """

    :param x: Ongorucu Degisken, Ev Buyuklukleri
    :param y: Hedef Degisken, Ev Fiyatlari
    :param teta: parametre
    :return: maliyet
    """
    m = len(x)
    toplamin_acilimi = np.power(((x * teta.T) - y), 2)
    maliyet = np.sum(toplamin_acilimi) / (2 * m)
    return maliyet


def mae_maliyet_fonksiyonu_hesapla(x, y, teta):
    """

    :param x: Ongorucu Degisken, Ev Buyuklukleri
    :param y: Hedef Degisken, Ev Fiyatlari
    :param teta: parametre
    :return: maliyet
    """
    m = len(x)
    toplamin_acilimi = np.abs(((x * teta.T) - y))
    maliyet = np.sum(toplamin_acilimi) / m
    return maliyet


def mse_maliyet_fonksiyonu_hesapla(x, y, teta):
    """

    :param x: Ongorucu Degisken, Ev Buyuklukleri
    :param y: Hedef Degisken, Ev Fiyatlari
    :param teta: parametre
    :return: maliyet
    """
    m = len(x)
    toplamin_acilimi = np.power(((x * teta.T) - y), 2)
    maliyet = np.sum(toplamin_acilimi) / m
    return maliyet


def rmse_maliyet_fonksiyonu_hesapla(x, y, teta):
    """

    :param x: Ongorucu Degisken, Ev Buyuklukleri
    :param y: Hedef Degisken, Ev Fiyatlari
```

```
    :param teta: parametre
    :return: maliyet
    """
    m = len(x)
    toplamin_acilimi = np.power(((x * teta.T) - y), 2)
    maliyet = np.sqrt(np.sum(toplamin_acilimi) / m)
    return maliyet
```

ozellik_normalizasyonu.py

```
# coding=utf-8
""" Feature Normalization
 Ozellik Normalizasyonu"""
import numpy as np


def ozellik_normalizasyonu(x):
    """

    :param x: normallestirilecek ozellikler
    :return: normalize olmus X, ortalamasi ve sapmasi
    """
    butun_ortalamalar = []
    butun_standart_sapmalar = []
    X_normalize = x

    n_c = x.shape[1]
    for i in range(n_c):
        m = np.mean(x[:, i])
        s = np.std(x[:, i])

        butun_ortalamalar.append(m)
        butun_standart_sapmalar.append(s)
        X_normalize[:, i] = (X_normalize[:, i] - m) / s

    return X_normalize, butun_ortalamalar, butun_standart_sapmalar
```

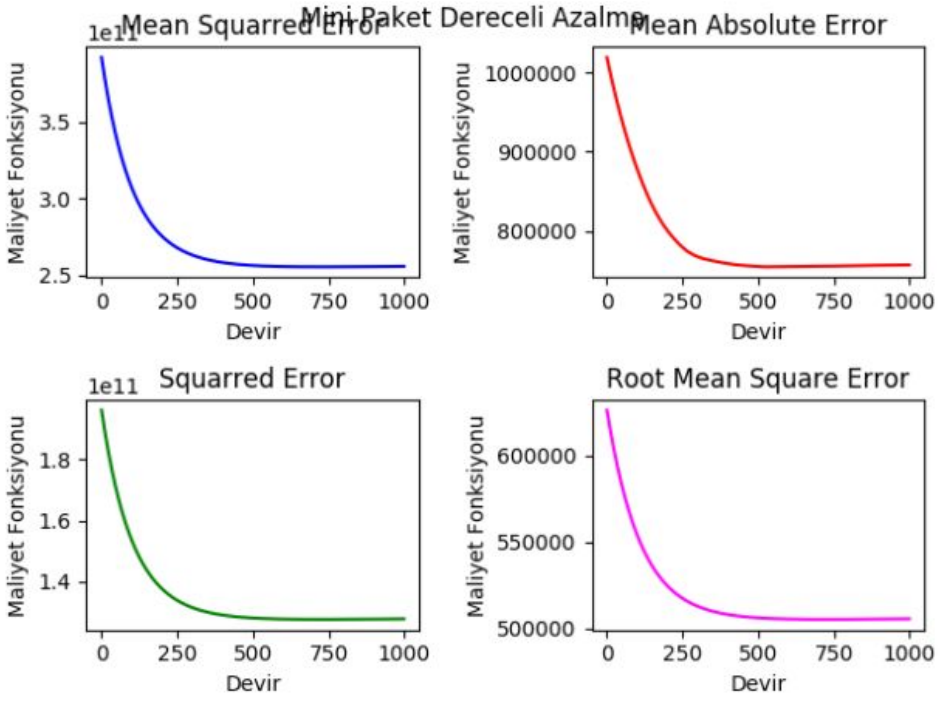
ciz.py

```
# coding=utf-8
""" Plotting Data
Veri Setinin Grafiginin Cizilmesi"""

import matplotlib.pyplot as plt
```

```
import numpy as np


def maliyet_devir_grafigi_ciz(toplam_devir, mse, mae,
squared_error, rmse, baslik):
    """

    :param toplam_devir: Toplam Devir Sayisi
    :param mse: Mean Squared Error
    :param mae: Mean Absolute Error
    :param squared_error: Squared Error
    :param rmse: Root Mean Squared Error
    :param baslik: Grafigin Basligi
    """
    fig = plt.figure(1)
    fig.suptitle(baslik)

    ax1 = fig.add_subplot(221)
    ax1.set_title("Mean Squarred Error")
    ax1.set_xlabel("Devir")
    ax1.set_ylabel("Maliyet Fonksiyonu")
    ax1.plot(np.arange(toplam_devir), mse, 'blue')

    ax2 = fig.add_subplot(222)
    ax2.set_title("Mean Absolute Error")
    ax2.set_xlabel("Devir")
    ax2.set_ylabel("Maliyet Fonksiyonu")
    ax2.plot(np.arange(toplam_devir), mae, 'red')

    ax3 = fig.add_subplot(223)
    ax3.set_title("Squarred Error")
    ax3.set_xlabel("Devir")
    ax3.set_ylabel("Maliyet Fonksiyonu")
    ax3.plot(np.arange(toplam_devir), squared_error, 'green')
    plt.tight_layout()

    ax4 = fig.add_subplot(224)
    ax4.set_title("Root Mean Square Error")
    ax4.set_xlabel("Devir")
    ax4.set_ylabel("Maliyet Fonksiyonu")
    ax4.plot(np.arange(toplam_devir), rmse, 'magenta')
    plt.tight_layout()

    fig.savefig(str(baslik).strip() + ".png")
```

Programın Çıktısı:

```
1- Veri Seti Yukleniyor ...
2- Ozellik Normalizasyonu Yapiliyor ...
3- Kesme Degeri Ekleniyor ...
4- Ogrenme Orani ve Devir Sayisi Ayarlaniyor ...
Devir Sayisi  1000
Alfa  0.005
5.2- Mini Paket Dereceli Azalma Hesaplaniyor ...
6- Maliyet Fonksiyonu - Devir Sayisi Grafigi Kaydediliyor ...
Hesaplanan Beta (MSE)  [[263240.17401489]
 [103160.30003545]
 [ 77686.37799889]]
Hesaplanan Beta (MAE)  [[263240.17401489]
 [103160.30003545]
 [ 77686.37799889]]
Hesaplanan Beta (SQRT ERR)  [[263240.17401489]
 [103160.30003545]
 [ 77686.37799889]]
Hesaplanan Beta (RMSE)  [[263240.17401489]
 [103160.30003545]
 [ 77686.37799889]]
7- Hipotez (SQRT ERR'e gore)
J(theta) = 263240.1740148875 + 103160.30003545314 x0 +
77686.37799888523 x1
8- Ev Fiyati Tahmini Yapalim ... (SQRT ERR'e gore)
Ev Buyuklugu 1650, 3 yatak odali ise Fiyati :  186534.470867
Ev Buyuklugu 1604, 3 yatak odali ise Fiyati :  182697.930425
Ev Buyuklugu 1000, 2 yatak odali ise Fiyati :  55790.6827138
```



3.4.4.3 Mini-Paket Dereceli Azalma ve Momentum ile Örnek

calistir.py

```
# coding=utf-8
""" Main
Calistir """

import pandas as pd
import numpy as np

from dereceli_azalma import
dereceli_azalma_mini_paket_momentum_ile
from ozellik_normalizasyonu import ozellik_normalizasyonu
from ciz import maliyet_devir_grafigi_ciz

u""" 1- Veri Setinin Yuklenmesi """
print  '1- Veri Seti Yukleniyor ...'
# Veri Setini Csv Dosyasindan Okuma
veri_seti = pd.read_csv('regresyon_veri_2.txt', delimiter=',',
header=None, names=['Buyukluk', 'Oda', 'Fiyat'])

# Tahmin Etmeye Calisacagimiz Verileri Okuma
y = veri_seti['Fiyat'].as_matrix()
del veri_seti['Fiyat']

# Toplam Girdi Sayisi
m = len(y)

# Ongorucu Veriler, ilk iki degisken: buyukluk ve oda sayisi
X_veri = veri_seti.as_matrix()

u""" 2- Ozellik Normalizasyonu """
print  '2- Ozellik Normalizasyonu Yapiliyor ...'
x, ortalama, standart_sapma = ozellik_normalizasyonu(X_veri)

u""" 3- Kesme Degerinin Eklenmesi"""
print  '3- Kesme Degeri Ekleniyor ...'
X = np.ones(shape=(m, (len(X_veri[0]) + 1)))
X[:, 1:(len(X_veri[0]) + 1)] = x

# Tahmin Edilecek Hedef Degiskeni Atama ve Boyutunu Ayarlama, yani
vektor haline donusturme
y.shape = (m, 1)

u""" 4- Ogrenme Orani ve Devir Sayisinin Ayarlanmasi"""
print  '4- Ogrenme Orani ve Devir Sayisi Ayarlaniyor ...'
toplam_devir = 1000
```

```
alfa = 0.001

print  'Devir Sayisi ', toplam_devir
print  'Alfa ', alfa

beta_mse = np.zeros((len(X[0]), 1))
beta_mae = np.zeros((len(X[0]), 1))
beta_sqrt_err = np.zeros((len(X[0]), 1))
beta_rmse = np.zeros((len(X[0]), 1))

paket = 50

u""" 5.3 - Momentum ile Mini Paket Dereceli Azalmanin
Hesaplanmasi"""
print  '5.3- Momentum Mini Paket Dereceli Azalma Hesaplaniyor ...'
beta_mse, maliyet_mse = dereceli_azalma_mini_paket_momentum_ile(X,
y, beta_mse, alfa, toplam_devir, paket, 'mse')
beta_mae, maliyet_mae = dereceli_azalma_mini_paket_momentum_ile(X,
y, beta_mae, alfa, toplam_devir, paket, 'mae')
beta_sqrt_err, maliyet_sqrt_err =
dereceli_azalma_mini_paket_momentum_ile(X, y, beta_sqrt_err, alfa,
toplam_devir,

paket,

'squared_error')
beta_rmse, maliyet_rmse =
dereceli_azalma_mini_paket_momentum_ile(X, y, beta_rmse, alfa,
toplam_devir, paket, 'rmse')

u""" 6- Dereceli Azalmanin Dogru Calisip Calismadiginin
Kontrolu"""
print  '6- Maliyet Fonksiyonu - Devir Sayisi Grafigi Kaydediliyor
...'
maliyet_devir_grafigi_ciz(toplam_devir, maliyet_mse, maliyet_mae,
maliyet_sqrt_err, maliyet_rmse,
                          baslik='Mini Paket Dereceli Azalma
Momentum Optimizasyon ile')

u""" 7- Sonuc"""
print  'Hesaplanan Beta (MSE) ', beta_mse
print  'Hesaplanan Beta (MAE) ', beta_mae
print  'Hesaplanan Beta (SQRT ERR) ', beta_sqrt_err
print  'Hesaplanan Beta (RMSE) ', beta_rmse

print  '7- Hipotez (SQRT ERR\'e gore)'
print  "J(theta) = " + str(beta_sqrt_err[0][0]) + " + " +
str(beta_sqrt_err[1][0]) + " x0 + " + str(
```

```
    beta_sqrt_err[2][0]) + " x1"

print  '8- Ev Fiyati Tahmini Yapalim ... (SQRT ERR\'e gore)'

# X Verilerinde Normalizasyon Yaptigimiz Icin Direkt Kullanamayiz.
# Tahmin etmek icin X ongorucu degiskenlerimizi buldugumuz
# ortalama ve standart sapmalara gore normalize edip kullaniyoruz
x0 = 1
x1 = (1650 - ortalama[0]) / standart_sapma[0]
x2 = (3 - ortalama[1]) / standart_sapma[1]
tahmin = np.dot([x0, x1, x2], beta_sqrt_err)

print  'Ev Buyuklugu 1650, 3 yatak odali ise Fiyati : ', tahmin[0]

x0 = 1
x1 = (1604 - ortalama[0]) / standart_sapma[0]
x2 = (3 - ortalama[1]) / standart_sapma[1]
tahmin = np.dot([x0, x1, x2], beta_sqrt_err)

print  'Ev Buyuklugu 1604, 3 yatak odali ise Fiyati : ', tahmin[0]

x0 = 1
x1 = (1000 - ortalama[0]) / standart_sapma[0]
x2 = (2 - ortalama[1]) / standart_sapma[1]
tahmin = np.dot([x0, x1, x2], beta_sqrt_err)

print  'Ev Buyuklugu 1000, 2 yatak odali ise Fiyati : ', tahmin[0]
```

dereceli_azalma.py

```
# coding=utf-8
""" Mini Batch Gradient Descent with Momentum
Mini Paket Dereceli Azalma Momentum ile """

import numpy as np
from maliyet_hesapla import squarred_error_maliyet_hesapla,
mae_maliyet_fonksiyonu_hesapla, \
    mse_maliyet_fonksiyonu_hesapla, \
    rmse_maliyet_fonksiyonu_hesapla


def mini_paket_al(x, y, mini_paket_buyuklugu):
    """

    :param x: ozellik degiskenleri
    :param y: hedef degisken
```

```
    :param mini_paket_buyuklugu:
    :return: veri setinden secilen paket(mini batch)
    """
    mini_paketler = []

    # X, y = np.random.shuffle(X,y)

    for i in range(0, x.shape[0], mini_paket_buyuklugu):
        x_mini = x[i:i + mini_paket_buyuklugu]
        y_mini = y[i:i + mini_paket_buyuklugu]

        mini_paketler.append((x_mini, y_mini))

    return mini_paketler


def dereceli_azalma_mini_paket_momentum_ile(x, y, beta, alfa,
devir_sayisi, paket_buyuklugu, maliyet_fonksiyonu):
    """

    :param x: ozellik degiskenleri
    :param y: hedef degisken
    :param beta: parametreler
    :param alfa: ogrenme orani
    :param devir_sayisi: devir sayisi
    :param paket_buyuklugu: secilen mini paketin buyuklugu
    :param maliyet_fonksiyonu: hesaplanacak maliyet fonksiyonu
turu
    :return: beta ve maliyet fonksiyonu
    """
    maliyet_J = np.zeros(shape=(devir_sayisi, 1))
    momentum_sabiti = 0.9
    hiz = 1
    for devir in range(devir_sayisi):

        for paketler in mini_paket_al(x, y, paket_buyuklugu):

            x = paketler[0]
            y = paketler[1]
            m = y.size

            tahminler = x.dot(beta)
            beta_size = beta.size

            for i in range(beta_size):
                X_i = x[:, i]
                X_i.shape = (m, 1)
```

```
                hiz = momentum_sabiti * hiz - alfa * (1.0 / m) *
((tahminler - y) * X_i).sum()
                beta[i][0] = beta[i][0] + hiz

                if maliyet_fonksiyonu == 'mse':
                    maliyet_J[devir, 0] =
mse_maliyet_fonksiyonu_hesapla(x, y, beta)
                if maliyet_fonksiyonu == 'mae':
                    maliyet_J[devir, 0] =
mae_maliyet_fonksiyonu_hesapla(x, y, beta)
                if maliyet_fonksiyonu == 'rmse':
                    maliyet_J[devir, 0] =
rmse_maliyet_fonksiyonu_hesapla(x, y, beta)
                if maliyet_fonksiyonu == 'squared_error':
                    maliyet_J[devir, 0] =
squarred_error_maliyet_hesapla(x, y, beta)
    return beta, maliyet_J
```

maliyet_hesapla.py

```
# coding=utf-8
""" Compute Cost Function
 Maliyet Fonksiyonunu Hesapla"""

import numpy as np


def squarred_error_maliyet_hesapla(x, y, teta):
    """

    :param x: Ongorucu Degisken, Ev Buyuklukleri
    :param y: Hedef Degisken, Ev Fiyatlari
    :param teta: parametre
    :return: maliyet
    """
    m = len(x)
    toplamin_acilimi = np.power(((x * teta.T) - y), 2)
    maliyet = np.sum(toplamin_acilimi) / (2 * m)
    return maliyet


def mae_maliyet_fonksiyonu_hesapla(x, y, teta):
    """

    :param x: Ongorucu Degisken, Ev Buyuklukleri
```

```
    :param y: Hedef Degisken, Ev Fiyatlari
    :param teta: parametre
    :return: maliyet
    """
    m = len(x)
    toplamin_acilimi = np.abs(((x * teta.T) - y))
    maliyet = np.sum(toplamin_acilimi) / m
    return maliyet


def mse_maliyet_fonksiyonu_hesapla(x, y, teta):
    """

    :param x: Ongorucu Degisken, Ev Buyuklukleri
    :param y: Hedef Degisken, Ev Fiyatlari
    :param teta: parametre
    :return: maliyet
    """
    m = len(x)
    toplamin_acilimi = np.power(((x * teta.T) - y), 2)
    maliyet = np.sum(toplamin_acilimi) / m
    return maliyet


def rmse_maliyet_fonksiyonu_hesapla(x, y, teta):
    """

    :param x: Ongorucu Degisken, Ev Buyuklukleri
    :param y: Hedef Degisken, Ev Fiyatlari
    :param teta: parametre
    :return: maliyet
    """
    m = len(x)
    toplamin_acilimi = np.power(((x * teta.T) - y), 2)
    maliyet = np.sqrt(np.sum(toplamin_acilimi) / m)
    return maliyet
```

ozellik_normalizasyonu.py

```
# coding=utf-8
""" Feature Normalization
 Ozellik Normalizasyonu"""
import numpy as np


def ozellik_normalizasyonu(x):
```

```
    """

    :param x: normallestirilecek ozellikler
    :return: normalize olmus X, ortalamasi ve sapmasi
    """
    butun_ortalamalar = []
    butun_standart_sapmalar = []
    X_normalize = x

    n_c = x.shape[1]
    for i in range(n_c):
        m = np.mean(x[:, i])
        s = np.std(x[:, i])

        butun_ortalamalar.append(m)
        butun_standart_sapmalar.append(s)
        X_normalize[:, i] = (X_normalize[:, i] - m) / s

    return X_normalize, butun_ortalamalar, butun_standart_sapmalar
```

ciz.py

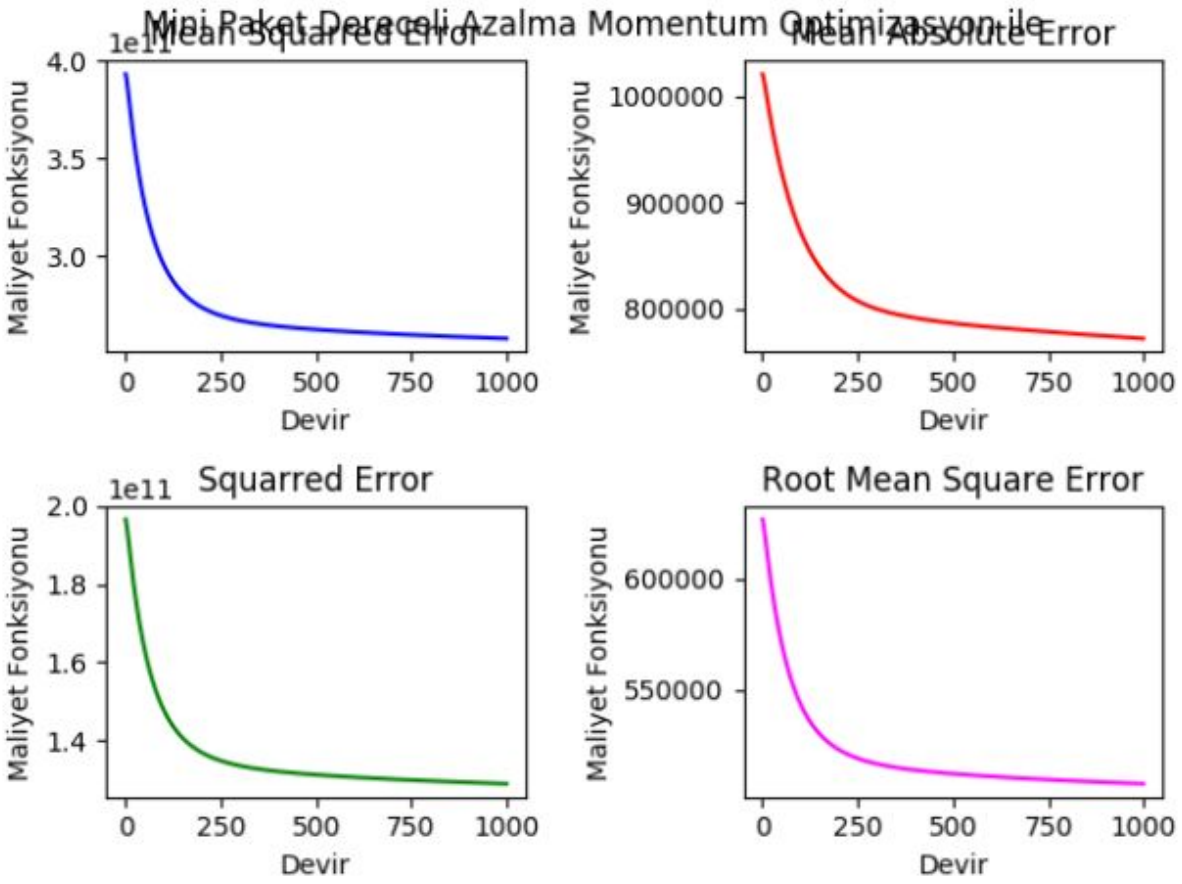
```
# coding=utf-8
""" Plotting Data
Veri Setinin Grafiginin Cizilmesi"""

import matplotlib.pyplot as plt
import numpy as np


def maliyet_devir_grafigi_ciz(toplam_devir, mse, mae,
squared_error, rmse, baslik):
    """

    :param toplam_devir: Toplam Devir Sayisi
    :param mse: Mean Squared Error
    :param mae: Mean Absolute Error
    :param squared_error: Squared Error
    :param rmse: Root Mean Squared Error
    :param baslik: Grafigin Basligi
    """
    fig = plt.figure(1)
    fig.suptitle(baslik)

    ax1 = fig.add_subplot(221)
    ax1.set_title("Mean Squarred Error")
```

```
    ax1.set_xlabel("Devir")
    ax1.set_ylabel("Maliyet Fonksiyonu")
    ax1.plot(np.arange(toplam_devir), mse, 'blue')

    ax2 = fig.add_subplot(222)
    ax2.set_title("Mean Absolute Error")
    ax2.set_xlabel("Devir")
    ax2.set_ylabel("Maliyet Fonksiyonu")
    ax2.plot(np.arange(toplam_devir), mae, 'red')

    ax3 = fig.add_subplot(223)
    ax3.set_title("Squarred Error")
    ax3.set_xlabel("Devir")
    ax3.set_ylabel("Maliyet Fonksiyonu")
    ax3.plot(np.arange(toplam_devir), squared_error, 'green')
    plt.tight_layout()

    ax4 = fig.add_subplot(224)
    ax4.set_title("Root Mean Square Error")
    ax4.set_xlabel("Devir")
    ax4.set_ylabel("Maliyet Fonksiyonu")
    ax4.plot(np.arange(toplam_devir), rmse, 'magenta')
    plt.tight_layout()

    fig.savefig(str(baslik).strip() + ".png")
```

Programın Çıktısı:

```
1- Veri Seti Yukleniyor ...
2- Ozellik Normalizasyonu Yapiliyor ...
3- Kesme Degeri Ekleniyor ...
4- Ogrenme Orani ve Devir Sayisi Ayarlaniyor ...
Devir Sayisi  1000
Alfa  0.001
5.3- Momentum Mini Paket Dereceli Azalma Hesaplaniyor ...
6- Maliyet Fonksiyonu - Devir Sayisi Grafigi Kaydediliyor ...
Hesaplanan Beta (MSE)  [[200884.97208343]
 [156012.12458164]
 [120322.20054088]]
Hesaplanan Beta (MAE)  [[200884.97208343]
 [156012.12458164]
 [120322.20054088]]
Hesaplanan Beta (SQRT ERR)  [[200884.97208343]
 [156012.12458164]
 [120322.20054088]]
Hesaplanan Beta (RMSE)  [[200884.97208343]
```

```
 [156012.12458164]
 [120322.20054088]]
7- Hipotez (SQRT ERR'e gore)
J(theta) = 200884.97208343152 + 156012.12458163846 x0 +
120322.20054087753 x1
8- Ev Fiyati Tahmini Yapalim ... (SQRT ERR'e gore)
Ev Buyuklugu 1650, 3 yatak odali ise Fiyati :  85886.2780293
Ev Buyuklugu 1604, 3 yatak odali ise Fiyati :  80084.173682
Ev Buyuklugu 1000, 2 yatak odali ise Fiyati :  -114633.951987
```



Bundan önceki üç bölümde çok değişkenli lineer regresyonun üç farklı algoritma ile gerçekleştirilmiş programlarını vermiştik. Bunlar sırayla:

- Toplu Dereceli Azalma
- Mini Toplu Dereceli Azalma
- Momentum Optimizasyonu ile Mini Toplu Dereceli Azalma

Şimdi bu üç farklı yöntemi birbirleriyle karşılaştırmak için Ev Fiyatı Tahminlerine bakalım.

Toplu Dereceli Azalma

('Ev Buyuklugu 1650, 3 yatak odali ise Fiyati : ', 266244.76416209328)
('Ev Buyuklugu 1604, 3 yatak odali ise Fiyati : ', 259246.93573893074)
('Ev Buyuklugu 1000, 2 yatak odali ise Fiyati : ', 140624.41778667527)

Mini Toplu Dereceli Azalma

('Ev Buyuklugu 1650, 3 yatak odali ise Fiyati : ', 265148.33582330809)
('Ev Buyuklugu 1604, 3 yatak odali ise Fiyati : ', 258838.15646738984)
('Ev Buyuklugu 1000, 2 yatak odali ise Fiyati : ', 131737.35983648058)

Momentum Optimizasyonu ile Mini Toplu Dereceli Azalma

('Ev Buyuklugu 1650, 3 yatak odali ise Fiyati : ', 120004.4445724087)
('Ev Buyuklugu 1604, 3 yatak odali ise Fiyati : ', 110142.80850545214)
('Ev Buyuklugu 1000, 2 yatak odali ise Fiyati : ', -160942.42948644597)

Momentum Optimizasyonu ile Mini Toplu Dereceli Azalma'nın sonuçlarına baktığımızda modelimiz 1000 büyüklükteki 2 odalı evin fiyatını negatif tahmin etmiş. Bir evin fiyatının negatif olamayacağını biliyoruz. Buradan Momentum Optimizasyonu'nun bizim problemimiz için uygun olmadığı sonucuna ulaşabiliriz. Batch Gradient Descent ile Mini-Batch Gradient Descent'in tahminlerine baktığımızda modellerin birbirlerine çok yakın değerler tahmin ettiğini görüyoruz. Bu iki yöntemin modelimiz için iyi birer hipotez oluşturduğunu söyleyebiliriz. Öğrenme Oranı ve Devir Sayısını değiştirerek modellerin daha iyi çalışıp çalışmayacağını veya hangi dereceli azalma algoritmasının bu veri setine daha uygun olduğunu deneyip görebiliriz.

3.4.5 Normal Denklemler

Normal Denklemi en küçük kareler maliyet fonksiyonuyla doğrusal regresyon problemine analitik bir çözüm olarak sunulabilir. Bazı durumlarda (küçük özellik kümeleri için olduğu gibi) onu kullanırken dereceli azalma uygulamaktan daha etkilidir.

Normal equation

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

Lineer regresyon için Normal Denklem Türevi

Hipotez işlevi göz önüne alındığında:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n$$

En küçük kareler maliyetini en aza indirmek istiyoruz:

$$J(\theta_{0\ldots n}) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

$x^{(i)}$ 'inci örnek (m örneklerinden) ve $y^{(i)}$ inci beklenen sonuçtur.

Aradığımız regresyon katsayıları θ vektörel: $\begin{pmatrix} \theta_0 \\ \theta_1 \\ \dots \\ \theta_n \end{pmatrix} \in \mathbb{R}^{n+1}$

M girdi örneklerinin her biri, benzer şekilde, n + 1 satırlı bir sütun vektörü olup, x_0, kolaylık sağlamak için 1'dir. Şimdi hipotez fonksiyonunu şöyle yazabiliriz:

$$h_\theta(x) = \theta^T x$$

Bu, tüm örnekler üzerinde toplandığında, matris gösterimine bakabiliriz. "Tasarım matrisi" X'i (büyük harf X), her satırın i inci örneğinde (vektör $x^{(i)}$) olduğu m sıra matrisi olarak tanımlayacağız. Bununla, toplamı matris çarpımıyla değiştirerek asgari kareler maliyetini aşağıdaki gibi yeniden yazabiliriz:

$$J(\theta) = \frac{1}{2m}(X\theta - y)^T(X\theta - y)$$

Şimdi, bazı matris geçiş kimlikleri kullanarak, bunu biraz basitleştirebiliriz. Bir türevi yine de sıfır ile karşılaştıracağımız için $\frac{1}{2m}$ parçasını atacağım:

$$J(\theta) = ((X\theta)^T - y^T)(X\theta - y)$$

$$J(\theta) = (X\theta)^T X\theta - (X\theta)^T y - y^T(X\theta) + y^T y$$

$X\theta$ 'nın bir vektör olduğunu ve bu nedenle de y'dir. Dolayısıyla birbiriyle çarpıştığımızda, sıranın ne olduğu önemli değildir (boyutlar işlediği sürece). Böylece daha basitleştirebiliriz:

$$J(\theta) = \theta^T X^T X\theta - 2(X\theta)^T y + y^T y$$

Unutmayın ki θ bizim bilinmeyenimizdir. Yukarıdaki işlevin minimumu nerede bulursak, θ ile türetilir ve 0 ile karşılaştırırız. Bir vektörle türetmek rahatsız olabilir, ancak endişelenilecek bir şey yoktur. Hatırlayın ki, lineer formüllerin bir sistemini rahatça temsil etmek için yalnızca matris gösterimini kullanıyoruz. Böylece, vektörün her bir bileşeni tarafından türettikten sonra ortaya çıkan türevleri tekrar bir vektöre birleştiriyoruz. Sonuç:

$$\frac{\partial J}{\partial \theta} = 2X^T X\theta - 2X^T y = 0$$

Veya:

$$X^T X\theta = X^T y$$

Şimdi, $X^T X$ matrisinin tersine çevrilebilir olduğunu varsayarsak, her iki tarafı da $(X^T X)^{-1}$ ile çarpabiliriz:

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

İşte Normal Denklem.

- Normal Denklem: En küçük kareler maliyet fonksiyonuyla doğrusal regresyon problemine analitik bir çözüm olarak sunulmasıdır.

Dereceli Azalma, maliyeti en aza indirmenin bir yolunu sunar. Bunu yapmanın ikinci bir yolu da bu sefer minimizasyonu açıkça ve yinelemeli bir algoritmaya başvurmadan gerçekleştirmektir. Bu yöntemde, maliyeti teta'larına göre türevlerini açıkça alıp sıfıra eşitleyerek en aza indiririz. Normal denklemin bulunmasının ispatlarına ve matris gösterimlerine gelmeden önce normal denklemine bakalım.

Normal equation

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

Dereceli Azalma ile Normal Denklemi karşılaştıralım.

Dereceli Azalma:

- Öğrenme Oranı alfa değerini seçmeliyiz.
- Birçok iterasyon (devir,yineleme) yapmalıyız.
- Eğer özellik sayımız çok fazla ise dereceli azalma iyi çalışır.

Normal Denklem:

- Öğrenme Oranı alfa değerini seçmemize gerek yok.
- İterasyona gerek yok.
- Sadece $(X^T X)^{-1}$ hesaplamalıyız.
- Eğer özellik sayısı çok fazla ise (özellik sayısı = n > 10000) yavaş çalışır.

Bir önceki bölümde kullandığımız regresyon_veri_2.txt veri setine şimdi normal denklem uygulayarak tetaları hesaplayıp, ev fiyatı tahminlerini gerçekleştirelim. Bir önceki bölümde yazdığımız ozellik_normalizasyonu.py'yi aynen kullanacağız.

calistir.py

```
# coding=utf-8
""" Normal Equation
 Normal Denklemler """

import pandas as pd
import numpy as np
from ozellik_normalizasyonu import ozellik_normalizasyonu

u""" 1- Veri Setinin Yuklenmesi """
print '1- Veri Seti Yukleniyor ...', 'blue'
veri_seti = pd.read_csv('regresyon_veri_2.txt', delimiter=',',
header=None, names=['Buyukluk', 'Oda', 'Fiyat'])
y = veri_seti['Fiyat'].as_matrix()
del veri_seti['Fiyat']
X = veri_seti.as_matrix()
m = len(y)
```

```
u""" 2- Ozellik Normalizasyonu """
print '2- Ozellik Normalizasyonu Yapiliyor ...'
x, ort, std = ozellik_normalizasyonu(X)

u""" 3- Kesme Degerinin Eklenmesi """
print '3- Kesme Degeri Ekleniyor ...'
X = np.ones(shape=(m, (len(X[0]) + 1)))
X[:, 1:(len(X[0]) + 1)] = x

u""" 4- Normal Denklem ile Tetalarin Hesaplanmasi """
print '4- Normal Denklem ile Tetalar Hesaplanıyor ...'
teta = ((np.linalg.inv(X.T.dot(X))).dot(X.T)).dot(y)  # (X.T * X)
^-1 * X.T * Y
print 'Hesaplanan Teta : ', teta

print '5- Hipotez'
print "J(theta) = " + str(teta[0]) + " + " + str(teta[1]) + " x0 +
" + str(
    teta[2]) + " x1"

print '6- Ev Fiyati Tahmini Yapalim ...'
print ('Ev Buyuklugu 1650, 3 yatak odali ise Fiyati : ',
               np.dot([1, ((1650 - ort[0]) / std[0]), ((3 -
ort[1]) / std[1])], teta))
print ('Ev Buyuklugu 1604, 3 yatak odali ise Fiyati : ',
               np.dot([1, ((1604 - ort[0]) / std[0]), ((3 -
ort[1]) / std[1])], teta))
print ('Ev Buyuklugu 1000, 2 yatak odali ise Fiyati : ',
               np.dot([1, ((1000 - ort[0]) / std[0]), ((2 -
ort[1]) / std[1])], teta))
```

Son olarak, Normal Denklem ile Dereceli Azalma'nin ev fiyatları tahminlerini karşılaştıralım.

Toplu Dereceli Azalma:

('Ev Buyuklugu 1650, 3 yatak odali ise Fiyati : ', 266244.76416209328)
('Ev Buyuklugu 1604, 3 yatak odali ise Fiyati : ', 259246.93573893074)
('Ev Buyuklugu 1000, 2 yatak odali ise Fiyati : ', 140624.41778667527)

Normal Denklem:

('Ev Buyuklugu 1650, 3 yatak odali ise Fiyati : ', 265638.57462691294)
('Ev Buyuklugu 1604, 3 yatak odali ise Fiyati : ', 258401.40584131121)
('Ev Buyuklugu 1000, 2 yatak odali ise Fiyati : ', 141780.19614667015)

Her iki yöntemde de birbirlerine ne kadar yakın tahminlerde bulunduğumuzu görebilirsiniz. Modellerimiz doğru çalışıyor diyebiliriz.