Makine Öğrenmesi Algoritmaları

Regresyon Algoritmaları:

- Doğrusal Regresyon
- Polinom Regresyon
- Üstel Regresyon
- Lojistik Regresyon
- Logaritmik Regresyon

Doğrusal Regresyon

Doğrusal regresyon, ilgili ve bilinen başka bir veri değeri kullanarak bilinmeyen verilerin değerini tahmin eden bir veri analizi tekniğidir. Bilinmeyen veya bağımlı değişkeni ve bilinen veya bağımsız değişkeni doğrusal bir denklem olarak matematiksel olarak modeller. İki veya daha fazla değişken arasındaki doğrusal ilişkiyi modelleyen bir regresyon türüdür. Temel amacı, bir bağımlı değişkeni (y) bağımsız değişkenler (x) cinsinden tahmin etmektir.

İşletmeler, ham verileri güvenilir ve tahmin edilebilir bir şekilde iş zekâsına ve eyleme dönüştürülebilir öngörülere dönüştürmek için kullanır. Bilim insanları birçok alanda, biyoloji ve davranışsal, çevresel ve sosyal bilimler dâhil, ön veri analizi yapmak ve gelecekteki trendleri tahmin etmek için doğrusal regresyon kullanır. Makine öğrenimi ve yapay zekâ gibi birçok veri bilimi yöntemi, karmaşık problemleri çözmek için doğrusal regresyon kullanır.

Formül

$$Y = b0+b1x1+b2x2+\ldots+bnxn$$

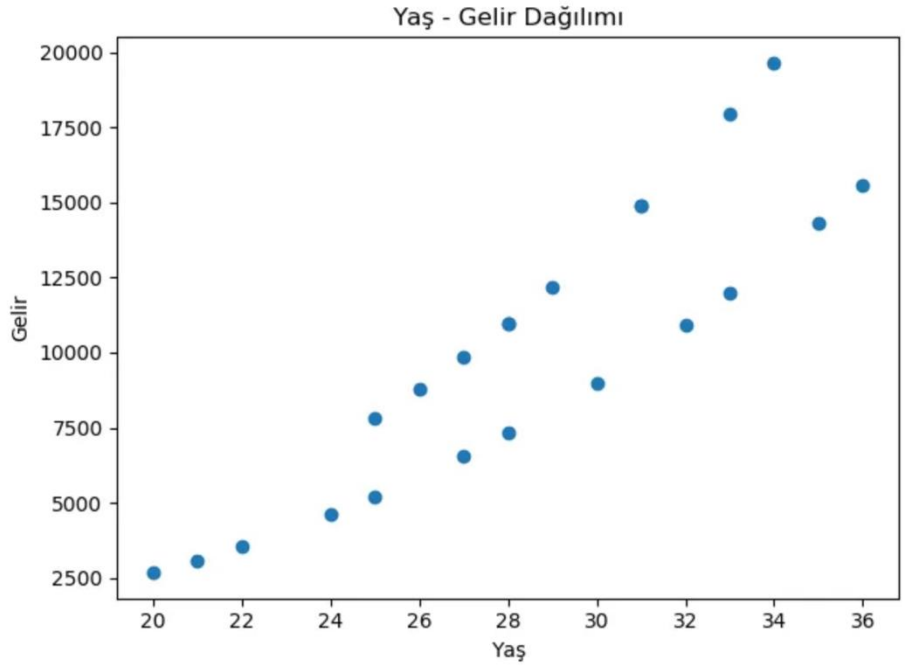
Burada:

- y: Tahmin edilen bağımlı değişken
- $b0$: Sabit terim (intercept)
- bi: Katsayılar (coefficients)
- xi: Bağımsız değişkenler (predictors)

```python
# İlgili kütüphanelerin kurulması

import numpy as np
import pandas as pd
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from sklearn import metrics
import matplotlib.pyplot as plt

# El ile yaş bilgisinin girilmesi

yas = [20,21,22,24,25,25,
       26,27,27,28,28,28,
       29,30,31,31,32,33,
       33,34,35,36]

# list comprehension ile her bir yaş için farklı ücret bilgisinin yaratılması

gelir = [x**3/np.random.randint(2,4) for x in yas]

# İki liste verisinin DataFrame'e aktarılması

df = pd.DataFrame({"yas":yas,
                   "gelir":gelir}))
}
```

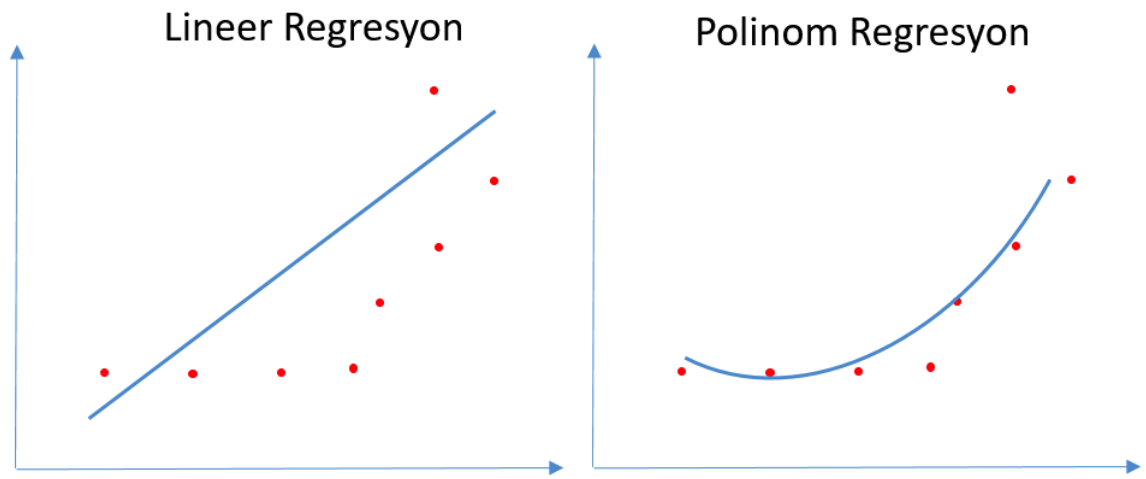
YAŞ	GELİR
20	2666
21	3087
22	3549
24	4608
25	7812
25	5208
26	8788
27	6561
27	9841
28	10976
28	10976
28	7317
29	12194
30	9000
31	14895
31	14895
32	10922
33	11979
33	17968
34	19652
35	14291
36	15552

```python
import matplotlib.pyplot as plt
plt.scatter(df.yas,df.gelir)
plt.title("Yaş - Gelir Dağılımı")
plt.xlabel("Yaş")
plt.ylabel("Gelir")
plt.show()
```



Polinom Regresyon

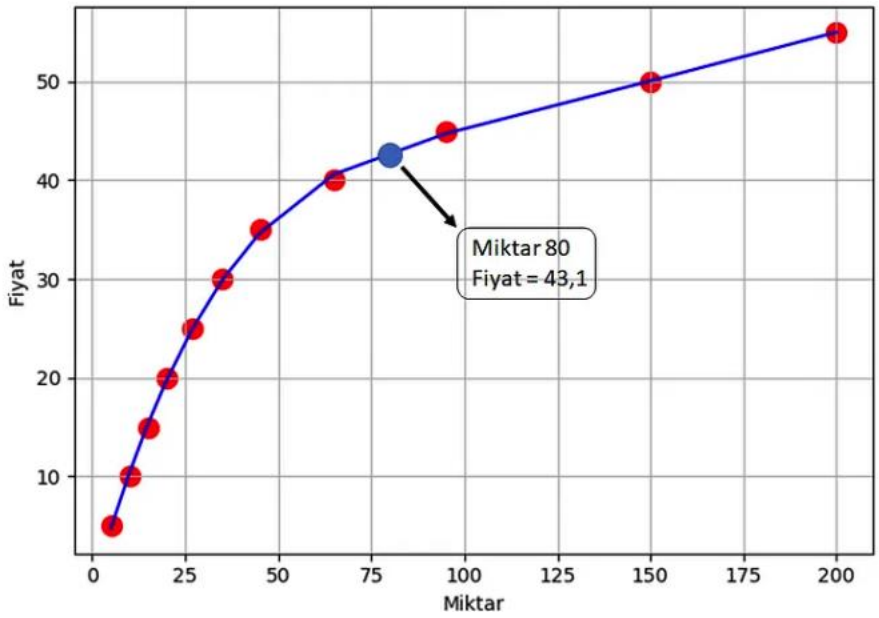
Doğrusal bir regresyonda girdi (x) ve çıktı (y) arasında lineer (doğrusal/line) bir ilişki olduğu varsayılmaktadır. Öte yandan polinomsal regresyonda ise girdi ve çıktı arasındaki ilişki düz bir doğru biçiminde değildir. Burada bir doğru yerine eğriden (curve) bahsedebiliriz. Temelde polinomsal regresyon bize bu eğrinin fonksiyonunu vermektedir. Burada önemli olan kısım bu polinom fonksiyonunun hangi dereceyi alacağıdır.



Formül

$$Y = b0+b1x+b2x2+\ldots+bnxn$$

```python
from sklearn.preprocessing import PolynomialFeatures
# Polinom derecesini belirlenir
poly_reg = PolynomialFeatures(degree = 4)
# Belirlenen dereceye göre bağımsız değişken hazırlanır
X_poly = poly_reg.fit_transform(x)
# Model eğitilir
pol_reg = LinearRegression()
pol_reg.fit(X_poly, y)
def polynomialRegressionVisual():
    plt.scatter(df["quantity"],
                df["price"],
                s=100,
                c="red",
                edgecolors='red'
                )
    plt.plot(x, pol_reg.predict(poly_reg.fit_transform(x)), color='blue')
    plt.title('Polinomsal Regresyon Sonucu')
    plt.xlabel('Miktar')
    plt.ylabel('Fiyat')
    plt.grid(True)
    plt.show()
    return
polynomialRegressionVisual()
print(pol_reg.predict(poly_reg.fit_transform([[80]])))
```



Polinomsal Regresyonun Avantajları

- Geniş fonksiyon yelpazesi altına sığabilir.
- Bu yaklaşım temel olarak geniş bir eğrilik alanına uyabilir.
- Polinomsal regresyon, bağımlı ve bağımsız değişken arasındaki ilişkiye en iyi yaklaşımı verebilir.

Polinomsal Regresyonun Dezavantajları

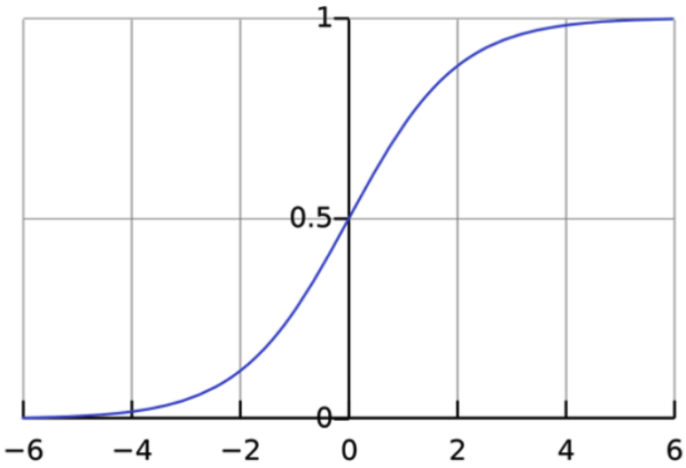
- Aykırı değerlere karşı bir hassasiyet söz konusudur. Bu yüzden veride bir veya iki aykırı değer bulunması, doğrusal olmayan bir analizin sonuçlarını ciddi şekilde etkileyebilir.

Lojistik regresyon

Lojistik regresyon, iki veri faktörü arasındaki ilişkileri bulmak için matematikten yararlanan bir veri analizi tekniğidir. Lojistik regresyon, daha sonra diğerine dayalı bu faktörlerden birinin değerini tahmin etmek için bu ilişkiyi kullanır. Tahminin genellikle evet ya da hayır gibi sınırlı sayıda sonucu vardır. Lojistik regresyon, yapay zeka ve makine öğrenimi (AI/ML) alanında önemli bir tekniktir. ML modelleri, insan müdahalesi olmadan karmaşık veri işleme görevlerini gerçekleştirmek için eğitebileceğiniz yazılım programlarıdır. Lojistik regresyon kullanılarak oluşturulan ML modelleri, kuruluşların iş verilerinden eyleme dönüştürülebilir öngörüler elde etmelerine yardımcı olur. Bu bilgileri operasyonel maliyetleri azaltmak, verimliliği artırmak ve daha hızlı ölçeklendirmek amacıyla tahmine dayalı analiz için kullanabilirler. Örneğin, işletmeler, çalışanların elde tutulmasını artıran veya daha kârlı ürün tasarımına yol açan kalıpları ortaya çıkarabilir.

Formülü

$$P(y=1|x)=\frac{1}{1+e^{-(b0+b1x1+\cdots+bnxn)}}$$



Logit fonksiyonu bağımsız değişkenin değerlerinden bağımsız olarak bağımlı değişken için yalnızca 0 ile 1 arasındaki değerleri döndürür. Lojistik regresyon, bağımlı değişkenin değerini bu şekilde tahmin eder.

Sınıflandırma algoritmaları:

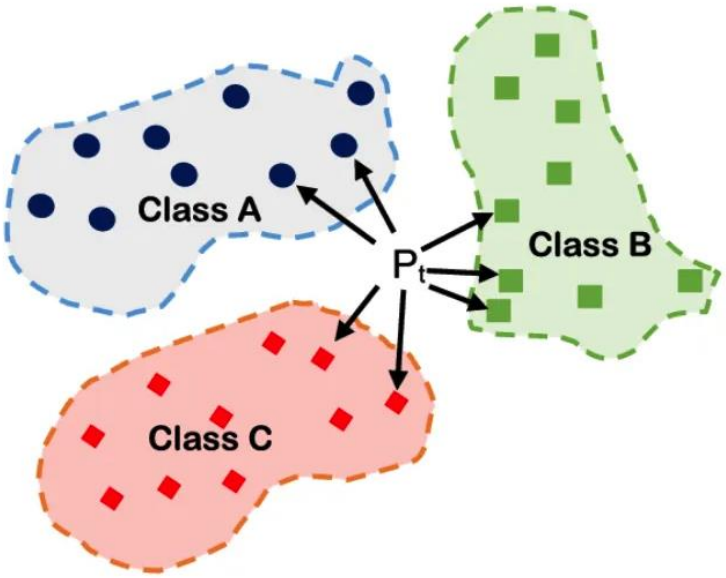
- K-En Yakın Komşular
- Karar Ağaçları
- Rastgele Orman
- Destek Vektör Makinesi
- Naive Bayes

K-En Yakın Komşular

K-En Yakın Komşular (KNN), sınıflandırma ve regresyon problemleri için kullanılan basit ve etkili bir algoritmadır. KNN, bir veri noktasının sınıfını tahmin ederken, eğitim veri setindeki en yakın k*k* komşusuna bakar ve çoğunluğun sınıfını tahmin olarak belirler. Bu algoritma, örüntü tanıma, görüntü sınıflandırma ve öneri sistemlerinde yaygın olarak kullanılır. Gözlemlerin birbirlerine olan benzerlikleri üzerinden tahmin yapılan, Denetimli Öğrenmenin (Supervised Learning) Regresyon ve Sınıflandırma problemlerinde kullanılan algoritmadır.

Formül
KNN, bir veri noktasının sınıfını belirlerken, genellikle Euclidean mesafesi gibi bir mesafe metriğini kullanır:

$$d(x,y)=\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$



K en yakın komşu (KNN) algoritması, adından da anlaşılacağı üzere, komşularına bakarak tahminlemede bulunan bir algoritmadır. KNN algoritmasında, benzer olan şeyler birbirine yakındır varsayımı geçerlidir. Resim incelendiğinde, genel itibari ile, benzer olan sınıfların birbiri ile yakın mesafede oldukları gözlemlenmektedir. Dolayısıyla KNN algoritması da bu gözleme dayanarak, yeni gelecek tahminler bu noktalara yakınlığa göre tahminlenir.

Karar Ağacı

Karar ağacı, belirli bir soruna yönelik tüm potansiyel çözümleri haritalandıran akış şeması benzeri bir diyagramdır. Genellikle kuruluşlar tarafından, bir dizi karar almanın tüm olası sonuçlarını karşılaştırarak en uygun hareket tarzını belirlemeye yardımcı olmak için kullanılır. Karar ağaçları, tahmine dayalı modeller oluşturmak için kullanılabildiğinden makine öğreniminde de popüler bir araçtır. Bu tür karar ağaçları, bir müşterinin önceki satın alma geçmişine dayanarak bir ürünü satın alıp almayacağı gibi tahminler yapmak için kullanılabilir.

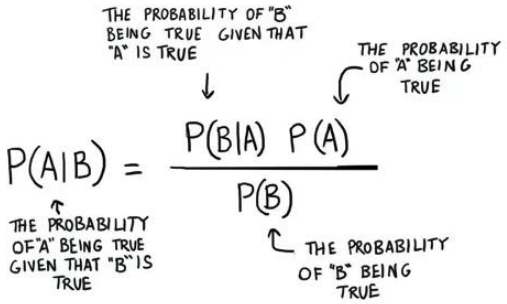
- **Kök Düğüm:** Kök düğüm tüm veri kümesini temsil eder ve ağacı başlatmak için kullanılır. Ağacın başlangıç noktasıdır ve verileri maksimum bilgi kazancı veya minimum Gini Impurity sağlayan özelliğe göre böler.
- **İç Düğüm:** Her bir iç düğüm, verileri iki veya daha fazla alt kümeye ayıran bir özelliği temsil eder. Bölme işlemi özelliğin değerine göre gerçekleştirilir ve her bir gözlemin izleyeceği yolu belirler. İç düğüm daha sonra birden fazla alt düğüme bölünür.
- **Yaprak Düğüm:** Yaprak düğüm, verilerin daha fazla bölünemeyen bir alt kümesini temsil eder. Kendisine ulaşan gözlemler için nihai tahmini içerir. Tahmin, alt kümedeki çoğunluk sınıfına veya hedef değişkenin ortalama değerine dayanır.

Formül

$$Gini = 1 - \sum_{i=1}^{n} (P_i)^2$$

Naive Bayes

Naive Bayes, olasılıksal bir sınıflandırma algoritmasıdır ve her bir özelliğin sınıfa bağımsız olduğunu varsayar. Naive Bayes, metin sınıflandırma, spam filtreleme, duygu analizi ve tıbbi teşhislerde yaygın olarak kullanılır. Naive Bayes sınıflandırmasında sisteme belirli bir oranda öğretilmiş veri sunulur (Örn: 100 adet). Öğretim için sunulan verilerin mutlaka bir sınıfı/kategorisi bulunmalıdır. Öğretilmiş veriler üzerinde yapılan olasılık işlemleri ile, sisteme sunulan yeni test verileri, daha önce elde edilmiş olasılık değerlerine göre işletvilir ve verilen test verisinin hangi kategoride olduğu tespit edilmeye çalışılır. Elbette öğretilmiş veri sayısı ne kadar çok ise, test verisinin gerçek kategorisini tespit etmek o kadar kesin olabilmektedir.



$P(A \mid B)$ = B olayı gerçekleştiğinde A olayının gerçekleşme olasılığı
$P(A)$ = A olayının gerçekleşme olasılığı
$P(B \mid A)$ = A olayı gerçekleştiğinde B olayının gerçekleşme olasılığı
$P(B)$ = B olayının gerçekleşme olasılığı

```python
# csv dosyalarını okumak için
import pandas as pd

# csv dosyamızı okuduk.
data = pd.read_csv('Iris.csv')

# Bağımlı Değişkeni ( species) bir değişkene atadık
species = data.iloc[:,-1:].values

# Veri kümemizi test ve train şekinde bölüyoruz
from sklearn.cross_validation import train_test_split
x_train, x_test, y_train, y_test = train_test_split(data.iloc[:,1:-1],species,test_size=0.33,random_state=0

# GaussianNB sınıfını import ettik
# 3 tane farklı Naive Bayes Sınıfı vardır.
# GaussianNB : Tahmin edeceğiniz veri veya kolon sürekli (real,ondalıklı vs.) ise
# BernoulliNB : Tahmin edeceğiniz veri veya kolon ikili ise ( Evet/Hayır , Sigara içiyor/ İçmiyor vs.)
# MultinomialNB : Tahmin edeceğiniz veri veya kolon nominal ise ( Int sayılar )
# Duruma göre bu üç sınıftan birini seçebilirsiniz. Modelin başarı durumunu etkiler.
from sklearn.naive_bayes import GaussianNB

# GaussianNB sınıfından bir nesne ürettik
gnb = GaussianNB()

# Makineyi eğitiyoruz
gnb.fit(x_train, y_train.ravel())

# Test veri kümemizi verdik ve iris türü tahmin etmesini sağladık
result = gnb.predict(x_test)

# Karmaşıklık matrisi
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,result)
print(cm)

# Başarı Oranı
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, result)
# Sonuç : 0.96
print(accuracy)
```

```
[[16  0  0]
 [ 0 19  0]
 [ 0  2 13]]
```

Karmaşıklık Matrisi Sonucu

16 + 19 + 16 + 2= 50 tane veri içinden 48 tanesini doğru tahmin edilirken 2 tanesi yanlış tahmin edilmiştir. Başarı oranı *48 / 50*= 0,96 ‘ dır.

Kümeleme Algoritmaları:

- K-means
- DBSCAN
- Mean Shift
- Hierarchical

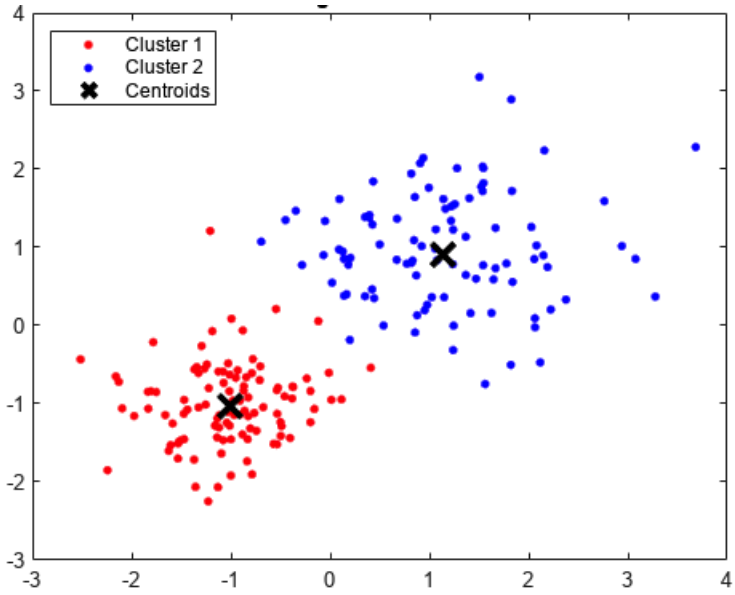
K-means

K-means, veri setini *k* sayıda kümeye ayıran popüler bir kümeleme algoritmasıdır. Bu algoritma, her küme için bir merkez noktası (centroid) belirler ve veri noktalarını en yakın merkez noktalarına atar. K-means, müşteri segmentasyonu, pazar araştırması, biyoinformatik ve görüntü sıkıştırma gibi alanlarda kullanılır.

Algoritmik olarak çalışma adımları aşağdaki gibidir.

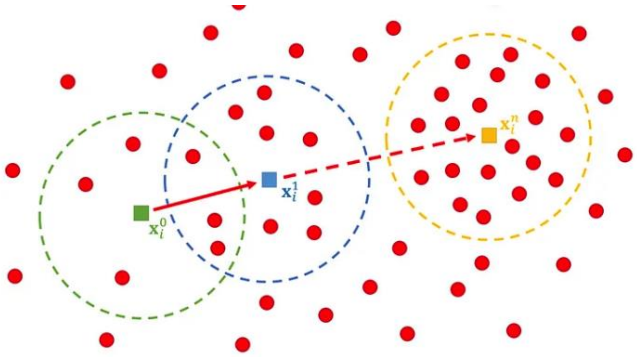
1. Küme merkezlerinin belirlenmesi
2. Merkez dışındaki örneklerin mesafelerine göre sınıflandırılması
3. Yapılan sınıflandırmaya göre yeni merkezlerin belirlenmesi
4. Kararlı hale gelinene kadar 2. ve 3. adımların tekrarlanması

```python
import math

def average(arr):
    return sum(arr) / len(arr)

def euclidian_distance(x1, center):
    return int(math.sqrt((x1 - center) ** 2))

def k_means(k, data):
    clusters = [[] for _ in range(k)]
    centroids = data[:k]
    old_centroids = [0] * k

    while True:
        clusters = [[] for _ in range(k)]

        for point in data:
            distances = [euclidian_distance(point, center) for center in centroids]
            min_distance_index = distances.index(min(distances))
            clusters[min_distance_index].append(point)

        converged = True
        for i in range(k):
            old_centroids[i] = centroids[i]
            centroids[i] = int(average(clusters[i]))
            if old_centroids[i] != centroids[i]:
                converged = False

        if converged:
            break

    return clusters

height_data = [157, 180, 170, 150, 160, 175, 185, 173, 183, 153, 163, 130]
print(k_means(2, height_data))
```



Mean Shift

Mean Shift, veri noktalarının yoğunluk bölgelerine doğru iteratif olarak kaydırılmasıyla kümeleme yapan bir algoritmadır. Bu yöntem, veri kümesinin yoğunluk zirvelerini bulur ve bu zirveleri küme merkezleri olarak tanımlar. Mean Shift, görüntü segmentasyonu, obje izleme ve anomali tespiti gibi alanlarda kullanılır. şekil üzerinde görüldüğü gibi bir daire veri üzerinde rastgele bir noktaya yerleştirilir, ardından dairenin içerisinde kalan verilerin merkez noktaları bulunup, dairenin merkez noktası bulunan merkez noktasına denk gelecek şekilde güncellenir ve bu işlemler merkez noktası değişmeyinceye kadar devam eder.



Hedef bölgenin, uzayın merkezine kaydırıldığı ve normalize edildiği düşünüldüğünde olasılık yoğunluk fonksiyonu aşağıdaki şekilde olacaktır.

$$q_u = C \sum_{i=1}^{n} k(\|\mathbf{x}_i\|^2)\, \delta(b(\mathbf{x}_i) - u),\ C = \left[\sum_{i=1}^{n} k(\|\mathbf{x}_i\|^2) \right]^{-1}$$

$$\delta(a) = \begin{cases} 1 & \text{if } a = 0 \\ 0 & \text{otherwise} \end{cases}$$

Hedef Nesne İçin Olasılık Yoğunluk Fonksiyonu Formülleri

Mevcut frame içinde merkezi y olan aday bölgenin normalize edilmiş olasılık yoğunluk fonksiyonu aşağıdaki şekilde hesaplanır.

$$p_u(\mathbf{y}) = C_h \sum_{i=1}^{n_h} k\left(\left\|\frac{\mathbf{y}-\mathbf{y}_i}{h}\right\|^2\right)\,\delta(b(\mathbf{y}_i)-u)$$

$$C_h = \left[\sum_{i=1}^{n_h} k\left(\left\|\frac{\mathbf{y}-\mathbf{y}_i}{h}\right\|^2\right)\right]^{-1}$$

Aday Nesne İçin Olasılık Yoğunluk Fonksiyonu Formülleri

```python
from sklearn.neighbors import NearestNeighbors
from sklearn.utils import extmath

def mean_shift(X, bandwidth=None, max_iterations=300):

    seeds = X
    n_samples, n_features = X.shape
    stop_thresh = 1e-3 * bandwidth  # when mean has converged
    center_intensity_dict = {}
    nbrs = NearestNeighbors(radius=bandwidth).fit(X)

    # For each seed, climb gradient until convergence or max_iterations
    for my_mean in seeds:
        completed_iterations = 0
        while True:
            # Find mean of points within bandwidth
            i_nbrs = nbrs.radius_neighbors([my_mean], bandwidth,
                                           return_distance=False)[0]
            points_within = X[i_nbrs]
            if len(points_within) == 0:
                break  # Depending on seeding strategy this condition may occur
            my_old_mean = my_mean  # save the old mean
            my_mean = np.mean(points_within, axis=0)
            # If converged or at max_iterations, addS the cluster
            if (extmath.norm(my_mean - my_old_mean) < stop_thresh or
                    completed_iterations == max_iterations):
                center_intensity_dict[tuple(my_mean)] = len(points_within)
                break
            completed_iterations += 1

    # POST PROCESSING: remove near duplicate points
    # If the distance between two kernels is less than the bandwidth,
    # then we have to remove one because it is a duplicate. Remove the
    # one with fewer points.
    sorted_by_intensity = sorted(center_intensity_dict.items(),
                                 key=lambda tup: tup[1], reverse=True)
    sorted_centers = np.array([tup[0] for tup in sorted_by_intensity])
    unique = np.ones(len(sorted_centers), dtype=np.bool)
    nbrs = NearestNeighbors(radius=bandwidth).fit(sorted_centers)
    for i, center in enumerate(sorted_centers):
        if unique[i]:
            neighbor_idxs = nbrs.radius_neighbors([center],
                                                  return_distance=False)[0]
            unique[neighbor_idxs] = 0
            unique[i] = 1  # leave the current point as unique
    cluster_centers = sorted_centers[unique]

    # ASSIGN LABELS: a point belongs to the cluster that it is closest to
    nbrs = NearestNeighbors(n_neighbors=1).fit(cluster_centers)
    labels = np.zeros(n_samples, dtype=np.int)
    distances, idxs = nbrs.kneighbors(X)
    labels = idxs.flatten()

    return cluster_centers, labels

cluster_centers, labels = mean_shift(np.array(data), 4000)

print len(cluster_centers)
17
plt.scatter(data[:,0],data[:,1])
plt.hold(True)
for x in asarray(cluster_centers): plt.plot(x[0],x[1],'rd')
plt.savefig('meanshift_2.png')
```