BLM4510

YAPAY ZEKA

PROJE ÖDEVİ

Ayşe Hilal Doğan - 17011907

Konu: Kiralık ev ilanlarından oluşturulan bir veriseti üzerinde regresyon algoritmalarını çalıştırmak ve sonuçları karşılaştırmak

Youtube linki: https://youtu.be/-r9Rnd2uGm0

A. PROJE KONUSU

https://www.hepsiemlak.com/kiralik sitesinden Python BeautifulSoup kütüphanesi kullanılarak çekilen verilerle, 10 özellik ve 2022 örnekten oluşan bir veri kümesi oluşturuldu. Özellikler sırasıyla; "fiyat", "oda", “salon”, brut", "net", "yas", "kat", "esyali_mi", "banyo", "depozito". Tahmin edilecek hedef değer “fiyat” değeridir. Verikümesi %75 eğitim, %25 test olarak ayrıldı. Buna göre 1515 eğitim, 506 test verisi bulunmaktadır. Bu verikümesi üzerinde 7 hazır, 1 kendi yazdığım regresyon algoritmaları ile regresyon yapılarak sonuçlar karşılaştırılmıştır. Hazır regresyon algoritmaları için sklearn kütüphanesi kullanılmıştır. Aşağıda kullanılan algoritmaların listesi bulunmaktadır.

1. Linear Regression
2. Decision Tree
3. K Nearest Neighbor
4. Naive Bayes Algoritmaları
   4.1. Gaussian Naive Bayes
   4.2. Multinomial Naive Bayes
   4.3. Complement Naive Bayes
   4.4. Bernoulli Naive Bayes
5. **Logistic Regression (kendi yazdığım algoritma)**

Oluşturulan verisetinin görüntüsü: (kiralik.head())

	fiyat	oda	salon	brut	net	yas	kat	esyali_mi	banyo	depozito
0	2.550	3	1	145.0	130.0	15	4	0	1	2.55
1	1.600	3	1	120.0	110.0	38	4	0	1	1.60
2	2.200	3	1	120.0	110.0	30	3	0	1	2.20
3	30.000	6	1	800.0	400.0	10	3	1	3	30.00
4	2.000	2	1	105.0	85.0	0	2	1	1	2.50

Oluşturulan verisetine dair bilgiler: (kiralik.describe())

	fiyat	oda	salon	brut	net	yas	kat	esyali_mi	banyo	depozito
count	2021.000000	2021.000000	2021.000000	2021.000000	2021.000000	2021.000000	2021.000000	2021.000000	2021.000000	2021.000000
mean	8.101930	2.586343	1.111331	133.342067	115.436714	16.280059	6.816428	0.316675	1.454231	965.075210
std	14.768863	1.643798	4.149075	85.677975	71.082590	13.089401	7.736339	0.465295	1.066366	10421.940945
min	1.000000	1.000000	0.000000	1.010000	1.000000	0.000000	1.000000	0.000000	1.000000	1.000000
25%	2.000000	2.000000	1.000000	85.000000	75.000000	5.000000	3.000000	0.000000	1.000000	150.000000
50%	3.000000	3.000000	1.000000	120.000000	100.000000	15.000000	4.000000	0.000000	1.000000	250.000000
75%	7.000000	3.000000	1.000000	150.000000	130.000000	25.000000	6.000000	1.000000	2.000000	500.000000
max	220.000000	50.000000	187.000000	800.000000	650.000000	115.000000	59.000000	1.000000	20.000000	270000.000000

B. UYGULAMA

a) Verisetinin oluşturulması

Yeterli veriyi çekmek için yaklaşık 95 farklı sayfaya ulaşıp, her sayfadaki ev ilanlarına tek tek girilerek bilgiler kaydedilmiştir.

Veriseti oluşturulurken, sayfanın response hatası vermesi üzerine 95 sayfa aynı anda çekilemiş olup, 20 sayfa – 20 sayfa şeklinde çekilerek çekilen veriler birleştirilmiştir.

Burada 95 farklı sayfada bulunan kiralık ilan linkleri çekilerek linkler dizisine kaydedilmiştir.

```
# 95 sayfa icin ilan linklerini al
for i in range(2, 97):
    r = requests.get("https://www.hepsiemlak.com/kiralik?page={}".format(i), headers=headers)
    soup = BeautifulSoup(r.content, "html.parser")
    sayfalar = soup.find_all("div", attrs={"class": "listing-item"})
    for sayfa in sayfalar:
        linkler.append(sayfa.find("a").get("href"))  #linkleri dizide tut
```

Kaydedilen linklere for içinde tek tek gidilerek kiralık evin fiyat – oda sayısı – metrekare (brut-net) – bina yaşı – bulunduğu kat - eşya durumu – banyo sayısı – depozito bilgileri çekilir.

```
# her bir sayfa linkinin icine gir
for link in linkler:
    r = requests.get("https://www.hepsiemlak.com{}".format(link), headers=headers)
    print("https://www.hepsiemlak.com{}".format(link))
    soup = BeautifulSoup(r.content, "html.parser")

    # ev fiyatları
    fiyat = soup.find("p", attrs={"class": "fontRB fz24 price"}).text.strip().replace(" TL", "")

    # alacagimiz ozellikler spec-item classının altında bulunuyor
    ozellikler = soup.find_all("li", attrs={"class": "spec-item"})

    # her bir ilan icin tek tek ozellikleri al
    for ozellik in ozellikler:
        text = ozellik.find_all("span")[0].text
        if text == "Oda + Salon Sayısı":
            oda = ozellik.find_all("span")[1].text.strip()

        if text == "Brüt / Net M2":
            brut = ozellik.find_all("span")[1].text.strip().replace("m2", "")

        if text == "Brüt / Net M2":
            net = ozellik.find_all("span")[2].text.strip().replace("/ ", "").replace("m2", "")

        if text == "Bina Yaşı":
            yas = ozellik.find_all("span")[1].text.strip().replace(" Yaşında", "").replace("Sıfır Bina", "0")

        if text == "Kat Sayısı":
            kat = ozellik.find_all("span")[1].text.strip().replace(" Katlı", "")
```

Çekilen bilgiler dataset dizisine kaydedilir ve excele çevrilir.

```
    dataset.append([fiyat, oda, brut, net, yas, kat, esyali_mi, banyo, depozito])

df = pd.DataFrame(dataset)
df.columns = ["fiyat", "oda", "brut", "net", "yas", "kat", "esyali_mi", "banyo", "depozito"]
df.to_excel("kiralik.xlsx")
```

b) Regresyon algoritmalarının veriseti üzerinde kullanılması

Main fonksiyonunda 9 farklı regresyon algoritması için bir seçim bulunmaktadır ve seçilen regresyon algoritması için regression() fonksiyonu çağırılır.

```
def main():
    selection = 1 #Linear Regression
    #selection = 2 #Decision Tree
    #selection = 3 #KNN
    #selection = 4 #Gaussian Naive Bayes
    #selection = 5 #Multinomial Naive Bayes
    #selection = 6 #Complement Naive Bayes
    #selection = 7 #Bernoulli Naive Bayes
    #selection = 8 #Logistic Regression

    regression(selection)

if __name__ == "__main__":
  main()
```

Regression() fonksiyonu içerisinde sırasıyla veriseti yüklenir, özellikler(X) ve hedef(y) kolonları belirtilir, veriseti %75 - %25 olarak ayrılır ve gelen seçime göre regresyon algoritması seçilir ve o algoritmaya göre yazılan fonksiyon çağırılır.

```
def regression(selection):
   # load the dataset
    kiralik = load_dataset()

    #prepare train data labels
    X=kiralik[['oda','salon','brut','net','yas','kat','esyali_mi','banyo','depozito']]
    # prepare the target
    y=kiralik['fiyat']

    #plot(kiralik)

    # Split this data as 1500 training and 500+ test
    X_train, X_test,y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

    if selection == 1:
      LinearRegressionFunc(X_train, X_test, y_train, y_test)
    elif selection == 2:
      DecisionTreeFunc(X_train, X_test, y_train, y_test)
    elif selection == 3:
      KNNFunc(X_train, X_test, y_train, y_test)
    elif selection == 4:
      GaussianNaiveBayesFunc(X_train, X_test, y_train, y_test)
    elif selection == 5:
      MultinomialNaiveBayesFunc(X_train, X_test, y_train, y_test)
    elif selection == 6:
      ComplementNaiveBayesFunc(X_train, X_test, y_train, y_test)
    elif selection == 7:
      BernoulliNaiveBayesFunc(X_train, X_test, y_train, y_test)
    elif selection == 8:
      LogisticRegressionFunction(X_train, X_test, y_train, y_test)
```

Csv dosya türünde hazırlanılan veriseti pandas kütüphanesi yardımıyla okunur.

```
def load_dataset():
  # read the dataset file (csv)
  kiralik = pd.read_csv('kiralik.csv', delimiter=';')

  print(kiralik.dtypes)
  print(kiralik.head())
  print(kiralik.shape)
  print(kiralik.describe())

  return kiralik
```

Okunan verinin boyutu (2021, 10) dur.

```
kiralik = pd.read_csv('kiralik.csv', delimiter=';')
#kiralik.head()
kiralik.shape
#kiralik.describe()
```

```
(2021, 10)
```

Veriyi daha iyi anlamak için evin fiyat – metrekare(brut) ve fiyat- bina yaşı dağılımı için grafik çizilmiştir.

```
def plot(kiralik):
  kiralik.plot(x='fiyat', y='brut', style='o')
  plt.title('fiyat-metrekare')
  plt.xlabel('fiyat')
  plt.ylabel('brut')
  plt.show()

  kiralik.plot(x='fiyat', y='yas', style='o')
  plt.title('fiyat-bina yasi')
  plt.xlabel('fiyat')
  plt.ylabel('yas')
  plt.show()
```

- Linear Regression

```
def LinearRegressionFunc(X_train, X_test, y_train, y_test):
  regressor = LinearRegression()
  regressor.fit(X_train, y_train)
  y_pred = regressor.predict(X_test)
  df = pd.DataFrame({'Gercek': y_test, 'Tahmin edilen': y_pred})
  print("Gercek - Tahmin Edilen")
  print("")
  print(df)
  print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
```

- Decision Tree

```
def DecisionTreeFunc(X_train, X_test, y_train, y_test):
  depth = 7
  regressor = DecisionTreeRegressor(max_depth=depth)
  regressor.fit(X_train, y_train)
  y_pred = regressor.predict(X_test)
  df = pd.DataFrame({'Gercek': y_test, 'Tahmin edilen': y_pred})
  print("Gercek - Tahmin Edilen")
  print("")
  print(df)
  print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
```

- K Nearest Neighbor

```
def KNNFunc(X_train, X_test, y_train, y_test):
   n = 8
   regressor = neighbors.KNeighborsRegressor(n)
   regressor.fit(X_train, y_train)
   y_pred = regressor.predict(X_test)
   df = pd.DataFrame({'Gercek': y_test, 'Tahmin edilen': y_pred})
   print(df)
   print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
```

- Gaussian Naive Bayes

```
def GaussianNaiveBayesFunc(X_train, X_test, y_train, y_test):
  regressor = GaussianNB()
  regressor.fit(X_train, y_train)
  y_pred = regressor.predict(X_test)
  df = pd.DataFrame({'Gercek': y_test, 'Tahmin edilen': y_pred})
  print("Gercek - Tahmin Edilen")
  print("")
  print(df)
  print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
```

- Multinomial Naive Bayes

```
def MultinomialNaiveBayesFunc(X_train, X_test, y_train, y_test):
  regressor = MultinomialNB()
  regressor.fit(X_train, y_train)
  y_pred = regressor.predict(X_test)
  df = pd.DataFrame({'Gercek': y_test, 'Tahmin edilen': y_pred})
  print(df)
  print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
```

- Complement Naive Bayes

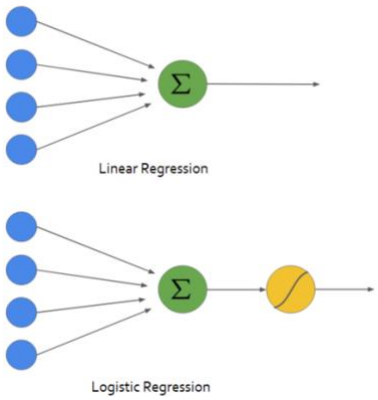
```
def ComplementNaiveBayesFunc(X_train, X_test, y_train, y_test):
  regressor = ComplementNB()
  regressor.fit(X_train, y_train)
  y_pred = regressor.predict(X_test)
  df = pd.DataFrame({'Gercek': y_test, 'Tahmin edilen': y_pred})
  print(df)
  print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
```

- Bernoulli Naive Bayes

```
def BernoulliNaiveBayesFunc(X_train, X_test, y_train, y_test):
  regressor = BernoulliNB()
  regressor.fit(X_train, y_train)
  y_pred = regressor.predict(X_test)
  df = pd.DataFrame({'Gercek': y_test, 'Tahmin edilen': y_pred})
  print(df)
  print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
```

- **Logistic Regression**

Logistic Regresyon, Linear Regresyon'dan farklı olarak bir sigmoid fonksiyonu kullanır. Algoritma yazılırken linear regresyon mantığından çıkarımlar yapılmıştır.




```
def LogisticRegressionFunction(X_train, X_test, y_train, y_test):
  n = 10  #verisetinde 9 ozellik var + 1 ise theta0 icin gerekiyor
  m = X_train.shape[0]  #verisetindeki ornek sayısı
  iteration = True

  katsayilar = np.zeros(n) # baslangicta butun katsayıları 0 ayarla
  # carpim icin trainin basina 1 lerden olusan bir satır eklenir
  train_data = np.concatenate((np.ones((X_train.shape[0],1)),X_train),axis=1)

  min = 10000

  # daha fazla kuculmeyene kadar iterasyonu devam ettir
  while(iteration):
    # katsayilarla train matrisini carp
    carpim = np.dot(katsayilar, train_data.T)

    # belirlenen carpim dizisini fiyat'i tahmin etmek icin kullan
    # sigmoid fonksiyonu
    predicted = 1/(1 + np.exp( -carpim ))

    # error u bul
    error = cost_function(predicted, y_train, m)

    # katsayıları guncelle (10 luk bir dizi)
    katsayilar = katsayilari_guncelle(predicted, m, train_data, y_train)

    if error < min:
      min = error
    # error bir onceki iterasyonla aynı degerde ise iterasyonu durdur
    if error == min:
      iteration = False
  print("error",error)
```

Regresyon yapılırken, ilk adımda özellik sayısı kadar rastgele özellik katsayısı belirlenir. Daha sonra bu katsayılarla verisetindeki değerler çarpılarak bir sonuç elde edilir. Fakat burada formüle bakılırsa, çarpılan ilk değer verisetindeki değerler ile değil, sabit 1 ile çarpılır.

$$y = \beta 0 + \beta 1X1 + \beta 2X2 + \ldots\ldots + \beta nXn$$

$$h_\theta = \begin{bmatrix} \theta_0 & \theta_1 \end{bmatrix} \begin{bmatrix} x_0 & x_0 & x_0 & x_0 \\ x_1 & x_2 & x_3 & x_4 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 6 & 5 & 7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}$$

Çarpım sonucunun üzerine logistic regresyondaki sigmoid fonksiyonu uygulanmalıdır.

Sigmoid fonksiyonu: Pr[1|a1,a2,..,ak] = 1 / (1+exp(-w0-w1a1-w2a2-…-wkak)

Burada exp içerisindeki değerler daha önce çarpımını bulduğumuz değerlerin negatifidir.

```
# katsayilarla train matrisini carp
carpim = np.dot(katsayilar, train_data.T)

# belirlenen carpim dizisini fiyat'i tahmin etmek icin kullan
# sigmoid fonksiyonu
predicted = 1/(1 + np.exp( -carpim ))

# error u bul
error = cost_function(predicted, train_data, m)
```

Cost function:

Sigmoid fonksiyonundan dönen değerler üzerinden cost fonksiyonu (error) bulunur. Bunun için tahmin edilen ve olması gereken değerler arasındaki uzaklık değeri logistic fonksiyona ait aşağıdaki formül ile bulunur.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \times log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \times log(h_\theta(x^{(i)}))\right]$$

```
def cost_function(predicted, y_train, m):
  sum = 0
  for i in range(0, m):
    x = y_train.index[i]
    p1 = log(1 - predicted[i])
    p2 = log(predicted[i])
    sum += (1-x) * p1 + x * p2

  sum = -(1/m) * sum

  return sum
```

Tahmin edilen değerler ile katsayılar güncellenir ve iterasyonun sonuna kadar güncellenmeye devam eder. Olabilecek en doğru katsayılar bulunmaya çalışılır.

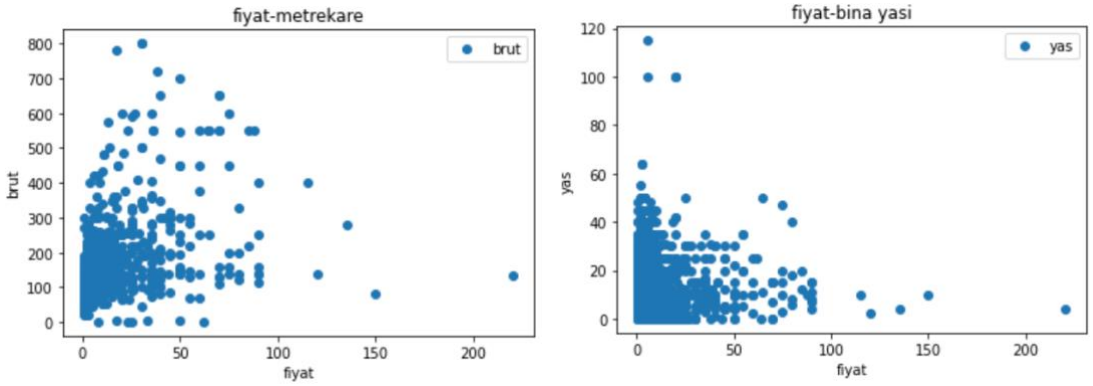
```
def katsayilari_guncelle(predicted, m, train_data, y_train):
  a = 0.001 #learning rate
  sum= 0
  katsayilar = np.zeros(10)

  # her bir ozelligi guncelle (9 ozellik icin)
  for k in range(0,10):
    for i in range(0,m):
      sum += (predicted[i] - y_train.index[i]) * train_data[i][k]

    katsayilar[k] = predicted[k]- (a/m)*sum

  return katsayilar
```

Bulunan error değeri minimize edilmeye çalışılır. Daha fazla küçülmeyene kadar iterasyona devam edilir. Error bir önceki değerle aynıysa, iteration = false yapılarak döngüden çıkılır.

```
if error < min:
  min = error
# error bir onceki iterasyonla aynı degerde ise iterasyonu durdur
if error == min:
  iteration = False
```

C. SONUÇ

Veriyi daha iyi anlamak için evin fiyat – metrekare(brut) ve fiyat- bina yaşı dağılımı aşağıda grafiklerde verilmiştir.



Fiyat- bina yaşı grafiği incelendiğinde, fiyatın yüksek olduğu evlerin yaşlarının küçük yani yeni yapım ev olduğu, Bina yaşı çok fazla olan örneklerin de fiyatın düşük olduğu bölgede olduğu gözlemlenmektedir. Buradan anlaşılabilir ki evin yeni yapım olmasını kira fiyatında etkisi vardır fakat çoğu zaman tek başına belirleyici değildir.

Regresyon

Regresyon algoritmalarının arasında hazırlanılan veriseti ile ölçülen sonuçlara göre, **en iyi başarıyı 5.07 ile Decision Tree Regresyon,** en kötü başarıyı ise 16.69 ile Multinomial Naive Bayes methodu vermiştir. Verisetimize en uygun algoritmanın Decision Tree olduğu

gözlemlenmiştir. **Başarı ölçümü Metrics kütüphanesinden mean absolute error ile yapılmıştır.**

- **Linear Regresyon :**

Linear regresyon için verisetindeki bütün özellikler kullanılarak algoritma uygulanmıştır. Fiyat kolonundaki veriler çok büyük olduğu zaman minimum ve maximum değerler arasında çok fark olduğu için bu veriler 1000 e bölünerek kullanılmıştır. Tahmin edilen değerler bazı evlerin daire değil villa tarzı evler olmasından kaynaklı bazen gereğinden çok fazla olabilmektedir fakat genellikle doğru tahminler yapılmaktadır. **Error sonucu 6.16 olarak belirlenmiştir.**

```
      Gercek  Tahmin edilen
955        2       8.355226
1494       1       2.229779
1858       1       0.676468
215        1       4.509195
506        3       1.765756
...      ...            ...
609        1       8.789222
1116       2       0.424098
1175       2       8.959086
89         1       3.757934
1688       2       0.644199

[506 rows x 2 columns]
Mean Absolute Error: 6.164406436811715
```

- **Decision Tree ile Regresyon :**

Decision Tree ile regresyonda **mean absolute error ile ölçülen başarı sonucu 5.07** dir. Başarı sonucu diğer algoritmaların arasında en iyi olan algoritmadır. İncelendiği zaman tahmin edilen değerlerde linear regresyonda olduğu gibi gerçeğinden çok yüksek alakasız tahminler bulunmadığı gözlemlenmektedir.

```
Gercek - Tahmin Edilen

      Gercek  Tahmin edilen
955        2       3.165049
1494       1       2.187500
1858       1       5.102041
215        1       2.000000
506        3       3.165049
...      ...            ...
609        1       3.165049
1116       2       2.187500
1175       2       2.187500
89         1       2.187500
1688       2       3.165049

[506 rows x 2 columns]
Mean Absolute Error: 5.075504473802515
```

- **K Nearest Neighbor ile Regresyon:**

K Nearest Neighbor ile regresyonda farklı k değerleri için sonuçlar elde edilerek en doğru k değeri bulunmaya çalışılmıştır. Ve en yüksek başarı K=7 yken alınmıştır, **mean absolute error ile ölçülen başarı sonucu 5.29'**dur. K değeri için 3 ten başlanarak 12 ye kadar farklı değerler denenmiş, 3 ten 7 ye kadar k değeri yükseldikçe başarının arttığı, 7 den daha fazla arttırıldığı zaman ise başarının tekrar düştüğü gözlemlenmiştir.

K= 3 için **mean absolute error ile ölçülen başarı sonucu 5.74** tür.

```
      Gercek  Tahmin edilen
955        2       2.666667
1494       1       1.000000
1858       1       1.666667
215        1       1.000000
506        3       3.666667
...      ...            ...
609        1       2.666667
1116       2       2.000000
1175       2       2.000000
89         1       1.000000
1688       2       2.666667

[506 rows x 2 columns]
Mean Absolute Error: 5.737154150197629
```

K= 5 için **mean absolute error ile ölçülen başarı sonucu 5.54** tür.

```
      Gercek  Tahmin edilen
955        2            2.4
1494       1            1.4
1858       1            3.0
215        1            1.0
506        3            3.2
...      ...            ...
609        1            2.4
1116       2            2.0
1175       2            2.2
89         1            1.8
1688       2           10.2

[506 rows x 2 columns]
Mean Absolute Error: 5.544268774703557
```

K= 7 için **mean absolute error ile ölçülen başarı sonucu 5.29** dur. Verilen K değerleri arasında en yüksek başarı 7 değeri için alınmıştır.

```
      Gercek  Tahmin edilen
955        2       2.571429
1494       1       1.285714
1858       1       4.000000
215        1       1.000000
506        3       3.285714
...      ...            ...
609        1       2.571429
1116       2       2.000000
1175       2       2.142857
89         1       1.714286
1688       2       8.142857

[506 rows x 2 columns]
Mean Absolute Error: 5.295878035008469
```

<u>Not: K=7 den sonra başarı yavaş yavaş tekrar düşmüştür.</u>

K= 8 için **mean absolute error ile ölçülen başarı sonucu 5.46** dır.

```
      Gercek  Tahmin edilen
955        2          2.625
1494       1          1.250
1858       1          4.375
215        1          1.375
506        3          3.125
...      ...            ...
609        1          2.625
1116       2          1.875
1175       2          2.500
89         1          1.625
1688       2          7.500

[506 rows x 2 columns]
Mean Absolute Error: 5.468132411067193
```

K= 12 için **mean absolute error ile ölçülen başarı sonucu 5.58** dir.

```
      Gercek  Tahmin edilen
955        2       5.833333
1494       1       1.500000
1858       1       5.583333
215        1       1.250000
506        3       3.500000
...      ...            ...
609        1       2.500000
1116       2       1.833333
1175       2       2.333333
89         1       1.500000
1688       2       5.916667

[506 rows x 2 columns]
Mean Absolute Error: 5.578886693017128
```

- ## Naive Bayes Algoritmaları

  - **Gaussian Naive Bayes ile Regresyon:**

  Gaussian Naive Bayes algoritmasında başarı yukarıda listelenen algoritmalara göre başarısızdır. Gaussian NB için **mean absolute error ile ölçülen başarı sonucu 6.39** dur.

```
      Gercek  Tahmin edilen
955        2              1
1494       1              1
1858       1              1
215        1              3
506        3              3
...      ...            ...
609        1              2
1116       2              1
1175       2              1
89         1              3
1688       2              3

[506 rows x 2 columns]
Mean Absolute Error: 6.391304347826087
```

  - **Multinomial Naive Bayes ile Regresyon:**

  Multinomial Naive Bayes algoritmasında başarı sonucu diğer algoritmalara göre gözle görülür farkla kötüdür. Multinomial NB için **mean absolute error ile ölçülen başarı sonucu 16.69** dur. Bu algoritma verisetimizin yapısına uygun bir algoritma değildir.

```
      Gercek  Tahmin edilen
955        2             15
1494       1             19
1858       1             80
215        1             19
506        3              1
...      ...            ...
609        1              3
1116       2              2
1175       2             30
89         1             19
1688       2              5

[506 rows x 2 columns]
Mean Absolute Error: 16.695652173913043
```

- **Complement Naive Bayes ile Regresyon:**

Complement Naive Bayes algoritmasında başarı sonucu Multinomial Naive Bayes ten sonra en kötü 2. Algoritmadır. Complement NB için **mean absolute error ile ölçülen başarı sonucu 7.24** tür.

```
      Gercek  Tahmin edilen
955        2              2
1494       1              2
1858       1             23
215        1              2
506        3              2
...      ...            ...
609        1              2
1116       2              2
1175       2              2
89         1              2
1688       2              2

[506 rows x 2 columns]
Mean Absolute Error: 7.24505928853755
```

- **Bernoulli Naive Bayes ile Regresyon:**

Bernoulli Naive Bayes algoritmasında başarı sonucu gözle görülür farkla kötüdür. Bernoulli NB için **mean absolute error ile ölçülen başarı sonucu 5.86** dır.

```
      Gercek  Tahmin edilen
955        2              2
1494       1              2
1858       1              2
215        1              2
506        3              2
...      ...            ...
609        1              2
1116       2              2
1175       2              2
89         1              2
1688       2              2

[506 rows x 2 columns]
Mean Absolute Error: 5.865612648221344
```

- **Logistic Regression**

Lojistik regresyon için lojistik regresyona ait sigmoid fonksiyonu, cost fonksiyonu uygulanarak algoritma yazılmış olup, cost fonksiyonu sonucu bulunan başarı değeri 0.69 dur.

```
Basari 0.6931471805599403
```

Kaynakça

https://scikit-learn.org/stable/modules/naive_bayes.html

https://www.sinanerdinc.com/python-beautifulsoup-modulu

https://analyticsindiamag.com/a-beginners-guide-to-regression-techniques/

https://www.geeksforgeeks.org/gradient-descent-in-linear-regression/

https://www.geeksforgeeks.org/ml-linear-regression/

https://www.geeksforgeeks.org/mathematical-explanation-for-linear-regression-working/

https://towardsdatascience.com/building-a-logistic-regression-in-python-301d27367c24

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html

https://www.baeldung.com/cs/gradient-descent-logistic-regression

https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python

https://www.youtube.com/watch?v=T8QQ_NfE76I

https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148