T.C.

KARABÜK ÜNİVERSİTESİ

LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

BİYOMEDİKAL MÜHENDİSLİĞİ YÜKSEK LİSANS PROĞRAMI

BİYOMEDİKAL MÜHENDİSLİĞİNDE YAPAY SİNİR AĞI UYGULAMALARI

PYTHON İLE REGRESYON VE SINIFLANDIRMA UYGULAMASI RAPORU

TALEH BİNNATOV
2028142023

DR. ÖĞR. ÜYESİ HAKAN YILMAZ

Regression

Veri seti seçimi

Life expectancy – Tahmini ömür

Dünya Sağlık Örgütü (WHO) bünyesindeki Küresel Sağlık Gözlemevi (GHO) veri deposu, tüm ülkeler için sağlık durumunun yanı sıra diğer birçok ilgili faktörü de takip etmektedir. 193 ülke için ortalama yaşam süresi, sağlık faktörleri ile ilgili veri seti aynı DSÖ veri havuzu web sitesinden ve bunlara karşılık gelen ekonomik veriler ise Birleşmiş Milletler web sitesinden toplanmıştır. Sağlıkla ilgili faktörlerin tüm kategorileri arasında, yalnızca daha iyi temsil eden kritik faktörler seçilmiştir [1].

Tahmini ömür veri setinde çeşitli faktörler baz alınarak yıl cinsinden ömür tahmini yapılmaktadır. Veri setinde 193 ülke için için 2000-2015 dönemine ait verilerde aşılama faktörlerine, ölüm faktörlerine, ekonomik faktörlere, sosyal faktörlere ve diğer sağlıkla ilgili faktörlere yer verilmiştir.

Veri seti 22 Sütun ve 2938 satırdan oluşmaktadır.

Veri seti aşağıdaki sütunları içermektedir.

Country, Year, Status, Life_expectancy, Adult_Mortality, infant_deaths, Alcohol, percentage_expenditure, Hepatitis_B,Measles, BMI, under_five_deaths, Polio, Total_expenditure, Diphtheria, HIV_AIDS, GDP, Population, thinness_1_19 years, thinness_5_9_years, icor, Schooling

Regresyonda kullanım için aşağıdaki sütunlar belirlenmiştir.

Y	X				
Life expectancy (tahmini ömür)	Adult Mortality (Yetişkin Ölümlülük)	İnfant deaths (Bebek ölümleri)	Measles (Kızamık)	Diphtheria (Difteri)	HIV_AIDS

Veri hazırlık süreci

Genel bakış

Pandas ve Numpy kütüphaneleri import edildikten sonra df=pd.read_csv("led.csv",sep=",") ile Excel'den veri çekilerek incelenmek üzere Dataframe'ye atıldı.

print(df.head()), print(df.info()), df.dtypes, print(df.describe()) komutlarıyla verisetine genel bakış yapıldı.

Sayısal değerler üzerinde inceleme yapmamızı sağlayan print(df.describe()) ile her sütun için sütundakı veri sayısı, her sütun ortalama ve sapma değeri, min ve max değeri incelendi.

```
print(df.describe())
```

```
              Year  Life_expectancy  Adult_Mortality  infant_deaths  \
count  2938.000000      2928.000000      2928.000000    2938.000000
mean   2007.518720        69.224932       164.796448      30.303948
std       4.613841         9.523867       124.292079     117.926501
min    2000.000000        36.300000         1.000000       0.000000
25%    2004.000000        63.100000        74.000000       0.000000
50%    2008.000000        72.100000       144.000000       3.000000
75%    2012.000000        75.700000       228.000000      22.000000
max    2015.000000        89.000000       723.000000    1800.000000

           Alcohol  percentage_expenditure  Hepatitis_B        Measles  \
count  2744.000000             2938.000000  2385.000000    2938.000000
mean      4.602861              738.251295    80.940461    2419.592240
std       4.052413             1987.914858    25.070016   11467.272489
min       0.010000                0.000000     1.000000       0.000000
25%       0.877500                4.685343    77.000000       0.000000
50%       3.755000               64.912906    92.000000      17.000000
75%       7.702500              441.534144    97.000000     360.250000
max      17.870000            19479.911610    99.000000  212183.000000
```

print(df.head())

```
print(df.head())
```

```
       Country  Year      Status  Life_expectancy  Adult_Mortality  \
0  Afghanistan  2015  Developing             65.0            263.0
1  Afghanistan  2014  Developing             59.9            271.0
2  Afghanistan  2013  Developing             59.9            268.0
3  Afghanistan  2012  Developing             59.5            272.0
4  Afghanistan  2011  Developing             59.2            275.0

   infant_deaths  Alcohol  percentage_expenditure  Hepatitis_B  Measles  ...
0             62     0.01               71.279624         65.0     1154  ...
1             64     0.01               73.523582         62.0      492  ...
2             66     0.01               73.219243         64.0      430  ...
3             69     0.01               78.184215         67.0     2787  ...
4             71     0.01                7.097109         68.0     3013  ...
```

print(df.info()) komutuyla incelenme zamanı bazı sütunlarda boş değerler olduğu görüldü.

```
: print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2938 entries, 0 to 2937
Data columns (total 22 columns):
Country                          2938 non-null object
Year                             2938 non-null int64
Status                           2938 non-null object
Life_expectancy                  2928 non-null float64
Adult_Mortality                  2928 non-null float64
infant_deaths                    2938 non-null int64
Alcohol                          2744 non-null float64
percentage_expenditure           2938 non-null float64
Hepatitis_B                      2385 non-null float64
Measles                          2938 non-null int64
BMI                              2904 non-null float64
under_five_deaths                2938 non-null int64
Polio                            2919 non-null float64
Total_expenditure                2712 non-null float64
Diphtheria                       2919 non-null float64
HIV_AIDS                         2938 non-null float64
GDP                              2490 non-null float64
Population                       2286 non-null float64
thinness_1_19 years              2904 non-null float64
thinness_5_9_years               2904 non-null float64
icor                             2771 non-null float64
Schooling                        2775 non-null float64
dtypes: float64(16), int64(4), object(2)
memory usage: 505.0+ KB
None
```

Özellik seçimi

Veri setinden eğitimini yapmak için Life_expectancy, Adult_Mortality, infant_deaths, Measles, Diphtheria, HIV_AIDS sütunlarına karar verildi. Boş veri sayısı az olan ve sağlık alanıyla ilgili sütunların seçilmesine özen gösterildi. Bulunan sütun sayısının fazlalığından tek tek kullanılmayacak sütunları silmek yerine sadece gerekli sütunlar `loc` komutuyla seçilerek dataframe'nin son haline aktarıldı.

```
df=df.loc[:,["Life_expectancy","Adult_Mortality","infant_deaths","Measles","Diphtheria","HIV_AIDS"]]
print(df.head())
```

```
   Life_expectancy  Adult_Mortality  infant_deaths  Measles  Diphtheria  \
0             65.0            263.0             62     1154        65.0
1             59.9            271.0             64      492        62.0
2             59.9            268.0             66      430        64.0
3             59.5            272.0             69     2787        67.0
4             59.2            275.0             71     3013        68.0

   HIV_AIDS
0       0.1
1       0.1
2       0.1
3       0.1
4       0.1
```

HIV_AIDS sütunu için aynı değerler gözüktüğünden veri çeşitliliği kontrol edildi. Kontrol sonrası veri çeşitliliğinin fazla olduğu anlaşıldı.

```
print(df.HIV_AIDS.unique())
```

```
[ 0.1  1.9  2.   2.3  2.6  2.5  2.4  2.1
  0.3  0.6  1.5  1.   1.1  1.2  1.3  1.4
  6.2  9.  12.7 13.4 14.4 20.6 28.4 31.9
  3.6  4.   3.4  3.8  4.3  4.8  5.1  5.2
  6.5  6.7  6.9  7.   7.1  3.5  4.6  4.9
  7.9  4.5  6.6  7.3  8.3 10.  11.2 12.
  5.   3.   5.9  6.8  7.6  2.7  4.2  5.7
 11.1 11.  10.1  9.5  5.6  5.4  6.4  9.1
 18.1  9.3  9.4  9.6 10.5 18.2 27.3 30.
 13.7 14.9 19.3 21.1 22.4 23.4 24.2 24.7
 16.3 16.2 15.9 15.3 12.2  8.7 11.7 15.2
  8.1  8.5 19.  23.5 26.4 28.1 29.5 29.7
```

Veri temizliği ve dönüşümü

Kayıp verilerin tespiti yapıldı. Life_expectancy, Adult_Mortality ve Diphtheria sütunlarında kayıp verilerin olduğu görüldü.

```
#Kayıp verilerin tespiti
print(df.isnull().sum())
print(df.isnull().sum().sum())
```

```
Life_expectancy    10
Adult_Mortality    10
infant_deaths       0
Measles             0
Diphtheria         19
HIV_AIDS            0
dtype: int64
39
```

Eksik verilerin doldurulması

Farklı yöntemlerle eksik verilerin doldurulması yapılmaktadır. Genellikle boş sütunlar Mean – ortalama değerle veya Interpolasyon yöntemiyle doldurulmaktadır.

Eksik verilerin doldurulması Interpolasyon yöntemiyle yapıldı. Elde var olan değerlerden yola çıkarak bilinmeyen değeri tahmin etmeye yarayan bu yöntem sütunlarda bulunan boş değerler yerine tahmini değer atayacaktır. Öğrenmede Forward Interpolasyon (ileri interpolasyon) yöntemi kullanılmıştır.

```
#replacing
df["Life_expectancy"]=df["Life_expectancy"].interpolate()
df["Adult_Mortality"]=df["Adult_Mortality"].interpolate()
df["Diphtheria"]=df["Diphtheria"].interpolate()
```

Interpolasyon sonrası boş değerlerin tamamen doldurulduğu görüldü.

```
print(df.isnull().sum().sum())
```

```
0
```

Veri tiplerinin dönüşümü

Öncelikle sütunların veri tipleri incelendi.

```
print(df.dtypes)
```

```
Life_expectancy    float64
Adult_Mortality    float64
infant_deaths        int64
Measles              int64
Diphtheria         float64
HIV_AIDS           float64
dtype: object
```

Adult_Mortality, Diptheria ve HIV_AIDS sütunlarının veri tipi integer olarak değiştirildi. Live_expectancy sütunu y sütunu olduğundan ve içerisinde ondalık sayılar bulunduğundan float tipinde kalmasının uygun olduğu düşünüldü.

```
donusum = {"Adult_Mortality": int,
           "Diphtheria":int,
           "HIV_AIDS":int,
          }
df=df.astype(donusum)
print(df.dtypes)
```

```
Life_expectancy    float64
Adult_Mortality      int32
infant_deaths        int64
Measles              int64
Diphtheria           int32
HIV_AIDS             int32
dtype: object
```

Tekrarlayan verilerin kaldırılması

Tekrarlayan veriler print(df[df.duplicated()]) komutuyla tespit edildi.

```
print(df[df.duplicated()])
```

```
      Life_expectancy  Adult_Mortality  infant_deaths  Measles  Diphtheria  \
455              72.4              124              0        0          99
607              59.5              272              1        0           7
660              78.7               96              1        0          99
1167             72.5              184              1        0          99
1462             75.0               93              1        9          81
1864             73.9              157              2        0          98
2009             75.3              125              8        0          88
2056             89.0               78              0        0          98
2198             79.0              186              0        0          99
2294             72.1              186              0        0          99
```

Tekrarlayan veriler temizlendi ve indeksler resetlendi.

```
df.drop_duplicates(inplace=True)
```

```
df.reset_index(inplace=True,drop=True)
```

Aşırı verilerin kontrolü

Her sütun için ortalama değer, sapma değeri ve ortanca değeri kontrol edildi. Ortalama ve ortanca değerlerinin yakın olması gerektiğinden farklılık olup olmadığı tüm sütunlar için kontrol edildi.

```
#Aşırı verilerin kontrolü
print(df["Life_expectancy"].mean())
print(df["Life_expectancy"].std())
print(np.median(df["Life_expectancy"]))
```

```
69.19600409836072
9.512799502374593
72.0
```

```
print(df["Adult_Mortality"].mean())
print(df["Adult_Mortality"].std())
print(np.median(df["Adult_Mortality"]))
```

```
164.7933743169399
124.39278411202002
144.0
```

```
print(df["infant_deaths"].mean())
print(df["infant_deaths"].std())
print(np.median(df["infant_deaths"]))
```

```
30.40266393442623
118.1155767176714
3.0
```

```
print(df["Measles"].mean())
print(df["Measles"].std())
print(np.median(df["Measles"]))
```

```
2427.852800546448
11485.971585336632
17.0
```

```
print(df["Diphtheria"].mean())
print(df["Diphtheria"].std())
print(np.median(df["Diphtheria"]))
```

```
82.315627141878
23.609077028462345
93.0
```

```
print(df["HIV_AIDS"].mean())
print(df["HIV_AIDS"].std())
print(np.median(df["HIV_AIDS"]))
```

```
1.5150788211103496
5.018062558896799
0.0
```

Measles sütunu için ortalama ve ortanca değerinde yüksek farklılıklar ortaya çıktı. Aşırı verilerin kontrolü quantile (çeyreklik) uç veri kontrolü ile yapıldı. Önce baştaki 2,5 % lik kısıma bakılmaktadır. Baştaki kısımın mediandan küçük olması beklenir. Baştaki kısımda sorun gözükmedi.

```
q_low  = df["Measles"].quantile(0.025)
print(q_low)

print(df[df["Measles"]<q_low])
```

```
0.0
Empty DataFrame
Columns: [Life_expectancy, Adult_Mortality, infant_deaths,
Index: []
```

Daha sonra sondaki 2,5 % lik kısıma bakılır. Sondaki kısımın büyük olması beklenir. Fakat ölçülen kuantil değeri üzerindeki verilerin çıkarılması gerekmektedir.

```
q_hi  = df["Measles"].quantile(0.975)
print(q_hi)

print(df[df["Measles"]>q_hi])
```

```
22971.099999999908
     Life_expectancy  Adult_Mortality  infant_deaths  Measles  Diphtheria  \
202              67.8              155            174    25934          93
406              56.9              283             44    54118          92
493              51.5               41             61    23934          63
541              48.0                4             42    24908          26
559              76.1               85            157    42361          99
560              75.8               86            171    52628          99
561              75.6               88            185    26883          99
564              75.0               92            231    38159          99
565              74.9               93            248    52461          99
566              74.5               97            266   131441          97
567              74.4               96            285   109023          93
568              74.2               98            307    99602          93
569              73.9               99            332   124219          87
```

Kuantil yöntemiyle yüksek değerli veri sayısı çok olduğundan ve bu verilerin silinmesinin skorları düşüreceği düşünüldüğünden ikinci bir yöntem olarak Measles sütunun Z koru hesaplandı. Veri setindeki veri sayısı çok olduğundan Z skoru 7 nin üzerinde olanlar gösterildi.

```
from scipy import stats
z=np.abs(stats.zscore(df["Measles"]))
#print(z)
print(np.where(z>7))
```

```
(array([ 566,  567,  568,  569,  573,  720,  722,  728, 1182, 1570, 1897,
       1899, 1901, 1902], dtype=int64),)
```

Z skoru 7 üzerinde olan satırlar ekranda incelenmek üzere yazdırıldı.

```
df.iloc[[566,  567,  568,  569,  573,  720,  722,  728, 1182, 1570, 1897, 1899, 1901, 1902]]
```

	Life_expectancy	Adult_Mortality	infant_deaths	Measles	Diphtheria	HIV_AIDS
566	74.5	97	266	131441	97	0
567	74.4	96	285	109023	93	0
568	74.2	98	307	99602	93	0
569	73.9	99	332	124219	87	0
573	72.2	11	457	88962	86	0
720	58.8	272	238	88381	74	1
722	57.9	278	239	133802	74	1
728	54.3	314	233	182485	6	2
1182	68.3	181	910	90387	87	0
1570	52.9	462	35	118712	93	13
1897	49.2	4	556	110927	36	5
1899	48.1	41	567	141258	29	5
1901	47.4	48	574	168107	27	5
1902	47.1	45	576	212183	29	4

Seçili satırlar veri setinden kaldırıldı ve indeks resetlendi.

```
df.drop([566,  567,  569, 722,  728, 1570, 1897, 1899, 1901, 1902], axis=0, inplace=True)
df.reset_index(inplace=True,drop=True)
```

Tekrar kontrol edildiğinde ortalama ve sapma değerinin kısmen düştüğü görüldü.

```
print(df["Measles"].mean())
print(df["Measles"].std())
print(np.median(df["Measles"]))
```

```
1945.3721727210418
7788.298975760563
17.0
```

Normalizasyon

X değerlerinin Y ye etkisini aynı aralığa çekmek için kullanılmaktadır. Normalizasyonu yapılan sütunun min değeri 0, max değeri ise 1 olarak değiştirilir. Diğer veriler formule göre bu aralıkta hesaplanır.

```
x_data=df.drop(["Life_expectancy"],axis=1)
a=(x_data-np.min(x_data))/(np.max(x_data)-np.min(x_data))
```

```
df1=pd.concat([df.loc[:,["Life_expectancy"]],a],axis=1)
print(df1.head())
```

```
   Life_expectancy  Adult_Mortality  infant_deaths   Measles  Diphtheria  \
0             65.0         0.362881       0.034444  0.011586    0.649485
1             59.9         0.373961       0.035556  0.004940    0.618557
2             59.9         0.369806       0.036667  0.004317    0.639175
3             59.5         0.375346       0.038333  0.027981    0.670103
4             59.2         0.379501       0.039444  0.030250    0.680412

   HIV_AIDS
0       0.0
1       0.0
2       0.0
3       0.0
4       0.0
```

Modelin eğitilmesi

Eğitim için y değeri olarak Life_expectancy, x değeri olarak kalan tüm sütunlar belirlendi.

```
y=df.Life_expectancy .values
x=df.drop(["Life_expectancy"],axis=1)
```

İlk önce veri setini ayırmadan eğitim yapıldı.

Öncelikle 4 çeşit regresyon için gerekli Sklearn kütüphanesi ve 2 çeşit değerlendirme için Sklearn.metrics kütüphanesi import edildi. 4 çeşit regresyon için gerekli kodlar yazıldı ve çalıştırıldı.

```
#Verisetini bölmeden yapılan eiğitim
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score,mean_absolute_error

plr=PolynomialFeatures(degree=3)
x_pol=plr.fit_transform(x)
lr_pol=LinearRegression()
lr_pol.fit(x_pol,y)
pol_pred=lr_pol.predict(x_pol)

mlr=LinearRegression()
mlr.fit(x,y)
mlr_pred=mlr.predict(x)

dt=DecisionTreeRegressor(random_state=42)
dt.fit(x,y)
dt_pred=dt.predict(x)

rf=RandomForestRegressor(n_estimators=100,random_state=42)
rf.fit(x,y.ravel())
rf_pred=rf.predict(x)
```

4 çeşit regresyon için R2 skor ve MAE (Mean Absolute Error) olmak üzere 2 çeşit eğitim sonucu hesaplandı. Eğitim skorları aşağıda verilmiştir.

```
#eğitim sonucu
print("pol r2:",r2_score(y,pol_pred),"mae:",mean_absolute_error(y,pol_pred))
print("mlr","r2",r2_score(y,mlr_pred),"mae:",mean_absolute_error(y,mlr_pred))
print("dt","r2",r2_score(y,dt_pred),"mae:",mean_absolute_error(y,dt_pred))
print("rf","r2",r2_score(y,rf_pred),"mae:",mean_absolute_error(y,rf_pred))
```

```
pol r2: 0.8362797159006125 mae: 2.8158439538955395
mlr r2 0.626119344665004 mae: 4.298873742153392
dt r2 0.9984143207285097 mae: 0.05065113091158328
rf r2 0.9896478940939388 mae: 0.6296981823522612
```

Bu sonuçlar eğitilen değerler ile test edilen değerlerin aynı değer kümesine ait öğrenmeye aittir. Dolayısıyla Overfit olarak nitelendirilen ezberleme durumu olasıdır. Özellikle Decision Tree ve Random Forest eğitimlerinde Overfit durumları kaçınılmazdır. Bu gibi durumları ortadan kaldırmak ve sonuçların doğruluğunu tespit etmek amacıyla veri setini train – eğitim ve test kısımlarına bölerek denemek gerekmektedir. Bu durumda modelin eğitim için ayrılan değerlerde eğitimi yapılacak, hiç görmediği test için ayrılan değerlerde ise testi yapılacaktır.

Train ve test olarak bölmek için Sklearn.model_selection den Train_test_split import edildi. Ardından test boyutu olarak 0.2 seçildi. Bu tüm verilerin %20'nin test için ayrıldığı demektir.

```
#Train ve test şeklinde bölünerek yapılan eğitim  Test boyutu=%20
from sklearn.model_selection import train_test_split
x_train, x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)

mlr=LinearRegression()
mlr.fit(x_train,y_train)
mlr_pred=mlr.predict(x_test)

plr=PolynomialFeatures(degree=2)
x_train_pol=plr.fit_transform(x_train)
x_test_pol=plr.fit_transform(x_test)
lr_pol=LinearRegression()
lr_pol.fit(x_train_pol,y_train)
pol_pred=lr_pol.predict(x_test_pol)


dt=DecisionTreeRegressor(random_state=42)
dt.fit(x_train,y_train)
dt_pred=dt.predict(x_test)

rf=RandomForestRegressor(n_estimators=150,random_state=42)
rf.fit(x_train,y_train.ravel())
rf_pred=rf.predict(x_test)

#eğitim sonucu
print("pol r2:",r2_score(y_test,pol_pred),"mae:",mean_absolute_error(y_test,pol_pred))
print("mlr","r2",r2_score(y_test,mlr_pred),"mae:",mean_absolute_error(y_test,mlr_pred))
print("dt","r2",r2_score(y_test,dt_pred),"mae:",mean_absolute_error(y_test,dt_pred))
print("rf","r2",r2_score(y_test,rf_pred),"mae:",mean_absolute_error(y_test,rf_pred))
```

```
pol r2: 0.7845020245251717 mae: 3.3948320216108456
mlr r2 0.6472038084414751 mae: 4.206150173739172
dt r2 0.8892754056033833 mae: 2.1032534246575345
rf r2 0.9394104112739246 mae: 1.6200597534790186
```

Verisetini bölmeden yapılan eğitim sonucuyla kıyaslayacak olursak Decision Tree ve Random Forest eğitimlerinde Overfit, Multiple linear regression için ise Underfit gözlemlendiği söylenebilir .

Random Forest için tree sayısı 100 yapılınca aşağıdaki gibi bir sonuç gözlemlendi.

```
#Random forest için tree(tahminci) sayısını 100 yaptığımızda
from sklearn.model_selection import train_test_split
x_train, x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)

rf=RandomForestRegressor(n_estimators=100,random_state=42)
rf.fit(x_train,y_train.ravel())
rf_pred=rf.predict(x_test)

#eğitim sonucu

print("rf","r2",r2_score(y_test,rf_pred),"mae:",mean_absolute_error(y_test,rf_pred))
```

```
rf r2 0.9397135357933223 mae: 1.6206515839041091
```

Random Forest için tree sayısı 200 yapılınca aşağıdaki gibi bir sonuç gözlemlendi.

```
#Random forest için tree(tahminci) sayısını 200 yaptığımızda
from sklearn.model_selection import train_test_split
x_train, x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)

rf=RandomForestRegressor(n_estimators=200,random_state=42)
rf.fit(x_train,y_train.ravel())
rf_pred=rf.predict(x_test)

#eğitim sonucu

print("rf","r2",r2_score(y_test,rf_pred),"mae:",mean_absolute_error(y_test,rf_pred))
```

```
rf r2 0.9390817767916156 mae: 1.618338351068171
```

Test boyutu olarak 0.1 seçildi. Bu tüm verilerin %10'nin test için ayrıldığı demektir. Test verilerinin az olması modelin daha fazla veri gördüğü için daha iyi bir tahmin edebileceği anlamına gelmektedir.

```
#Train ve test şeklinde bölünerek yapılan eiğitim  Test boyutu=%10
from sklearn.model_selection import train_test_split
x_train, x_test,y_train,y_test=train_test_split(x,y,test_size=0.1,random_state=42)

mlr=LinearRegression()
mlr.fit(x_train,y_train)
mlr_pred=mlr.predict(x_test)

plr=PolynomialFeatures(degree=2)
x_train_pol=plr.fit_transform(x_train)
x_test_pol=plr.fit_transform(x_test)
lr_pol=LinearRegression()
lr_pol.fit(x_train_pol,y_train)
pol_pred=lr_pol.predict(x_test_pol)


dt=DecisionTreeRegressor(random_state=42)
dt.fit(x_train,y_train)
dt_pred=dt.predict(x_test)

rf=RandomForestRegressor(n_estimators=150,random_state=42)
rf.fit(x_train,y_train.ravel())
rf_pred=rf.predict(x_test)

#eğitim sonucu
print("pol r2:",r2_score(y_test,pol_pred),"mae:",mean_absolute_error(y_test,pol_pred))
print("mlr","r2",r2_score(y_test,mlr_pred),"mae:",mean_absolute_error(y_test,mlr_pred))
print("dt","r2",r2_score(y_test,dt_pred),"mae:",mean_absolute_error(y_test,dt_pred))
print("rf","r2",r2_score(y_test,rf_pred),"mae:",mean_absolute_error(y_test,rf_pred))
```

```
pol r2: 0.7707804278758311 mae: 3.4666521827935437
mlr r2 0.6153256696774732 mae: 4.373541633259354
dt r2 0.8969146834513297 mae: 2.012328767123287
rf r2 0.9370019532263588 mae: 1.6032686335797686
```

Polinomal regresyon derecesi 3 olunca sonuçlar

```
#polinom derecesini 3 yaptığımızda
plr=PolynomialFeatures(degree=3)
x_train_pol=plr.fit_transform(x_train)
x_test_pol=plr.fit_transform(x_test)
lr_pol=LinearRegression()
lr_pol.fit(x_train_pol,y_train)
pol_pred=lr_pol.predict(x_test_pol)

#eğitim sonucu
print("pol r2:",r2_score(y_test,pol_pred),"mae:",mean_absolute_error(y_test,pol_pred))
```

```
pol r2: 0.8307622944780141 mae: 2.9062397908125623
```

Polinomal regresyon derecesi 4 olunca sonuçlar

```
#polinom derecesini 4 yaptığımızda
plr=PolynomialFeatures(degree=4)
x_train_pol=plr.fit_transform(x_train)
x_test_pol=plr.fit_transform(x_test)
lr_pol=LinearRegression()
lr_pol.fit(x_train_pol,y_train)
pol_pred=lr_pol.predict(x_test_pol)

#eğitim sonucu
print("pol r2:",r2_score(y_test,pol_pred),"mae:",mean_absolute_error(y_test,pol_pred))
```

```
pol r2: 0.744291225729262 mae: 3.1825217090747264
```

Polinomal regresyon derecesi 5 olunca sonuçlar

```
#polinom derecesini 5 yaptığımızda
plr=PolynomialFeatures(degree=5)
x_train_pol=plr.fit_transform(x_train)
x_test_pol=plr.fit_transform(x_test)
lr_pol=LinearRegression()
lr_pol.fit(x_train_pol,y_train)
pol_pred=lr_pol.predict(x_test_pol)

#eğitim sonucu
print("pol r2:",r2_score(y_test,pol_pred),"mae:",mean_absolute_error(y_test,pol_pred))
```

```
pol r2: -15.21433704900976 mae: 8.665970641372596
```

Normalizasyon yapılmış verilerle elde edilmiş sonuçlar

```
#Normalizasyonu yapılan verilerle eğitim
from sklearn.model_selection import train_test_split
x_train, x_test,y_train,y_test=train_test_split(a,y,test_size=0.1,random_state=42)

mlr=LinearRegression()
mlr.fit(x_train,y_train)
mlr_pred=mlr.predict(x_test)

plr=PolynomialFeatures(degree=3)
x_train_pol=plr.fit_transform(x_train)
x_test_pol=plr.fit_transform(x_test)
lr_pol=LinearRegression()
lr_pol.fit(x_train_pol,y_train)
pol_pred=lr_pol.predict(x_test_pol)


dt=DecisionTreeRegressor(random_state=42)
dt.fit(x_train,y_train)
dt_pred=dt.predict(x_test)

rf=RandomForestRegressor(n_estimators=100,random_state=42)
rf.fit(x_train,y_train.ravel())
rf_pred=rf.predict(x_test)

#eğitim sonucu
print("pol r2:",r2_score(y_test,pol_pred),"mae:",mean_absolute_error(y_test,pol_pred))
print("mlr","r2",r2_score(y_test,mlr_pred),"mae:",mean_absolute_error(y_test,mlr_pred))
print("dt","r2",r2_score(y_test,dt_pred),"mae:",mean_absolute_error(y_test,dt_pred))
print("rf","r2",r2_score(y_test,rf_pred),"mae:",mean_absolute_error(y_test,rf_pred))
```

```
pol r2: 0.8307339753733365 mae: 2.9049727021985103
mlr r2 0.6153256696774715 mae: 4.373541633259371
dt r2 0.8991025478620245 mae: 1.9808219178082183
rf r2 0.9374061216099066 mae: 1.6051142816373096
```

En yüksek skor Random Forestte (tahminci sayı=100,test boyutu=0.2) gözlemlendi.

```
rf r2 0.9397135357933223 mae: 1.6206515839041091
```

Multiple Linear regresyon ile x=[263,62,1154,65,0] değerlerine uygun yapılan tahmin sonucu y=64 olarak görüldü. Verisetindeki tahmini yapılan x değerleri ise 65.0'a eşittir.

```
from sklearn.model_selection import train_test_split
x_train, x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
mlr=LinearRegression()
mlr.fit(x_train,y_train)

print(mlr.intercept_,mlr.coef_)
y_pred_mlr=mlr.predict([[263,62,1154,65,0]])
print(int(y_pred_mlr))
```

```
67.86616836749793 [-3.72491501e-02 -5.46629050e-03 -6.54023562e-05  1.04619413e-01
 -5.06823375e-01]
64
```

Multiple Linear regresyon ile x=[263,62,1154,65,0] değerlerine uygun yapılan tahmin sonucu y=60 olarak görüldü. Verisetindeki tahmini yapılan x değerleri ise 65.0'a eşittir.

```
from sklearn.model_selection import train_test_split
x_train, x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
from sklearn.preprocessing import PolynomialFeatures
plr=PolynomialFeatures(degree=3)

x_train_pol=plr.fit_transform(x_train)
x_test_pol=plr.fit_transform(x_test)
lr_pol=LinearRegression()
lr_pol.fit(x_train_pol,y_train)

print(lr_pol.intercept_,lr_pol.coef_)

xnew=plr.fit_transform([[263,62,1154,65,0]])
y_pred_pol=lr_pol.predict(xnew)

print(int(y_pred_pol))
```

```
71.45813575234922 [ 1.85680974e-05  1.07364656e-01 -1.51056108e-01 -6.65995231e-04
 -4.28777157e-01 -3.11063855e+00 -6.27012871e-04  4.87520098e-04
  1.73023276e-06  2.82003196e-04  1.04594953e-02  1.41537619e-04
  1.63296099e-07  1.27529428e-03  1.41522951e-02  7.40500444e-09
  9.02138561e-06 -9.49661672e-06  6.58618143e-03  5.35558671e-03
  9.13267377e-02  6.34071253e-07 -8.29915847e-07  1.01338826e-09
  3.11308707e-07 -9.65509442e-06 -1.92348386e-07 -6.72745248e-10
  1.73005120e-07  1.73964384e-06 -9.05839292e-12 -2.37721391e-08
 -3.12002586e-08 -6.69119783e-06  1.73315003e-05 -1.03640713e-04
 -1.53648624e-08 -5.06788833e-10 -6.56477447e-07 -9.58229357e-06
  6.50724613e-12 -1.70703555e-09  5.33219185e-08 -6.66748939e-06
 -1.86416044e-05 -5.84051723e-04 -5.44425616e-14 -1.47835078e-11
 -1.80965687e-10 -1.78931861e-08 -3.22181527e-07  2.09034242e-06
 -1.75709837e-05 -4.90683798e-05 -3.60878179e-04 -2.95606516e-04]
60
```

Classification

Veri seti seçimi

Kronik böbrek hastalığı

Veriler Hindistan'da 25 özellikte (örneğin kırmızı kan hücresi sayımı, beyaz kan hücresi sayımı, vb.) ve 2 aylık bir süre boyunca alınmıştır. Bu veri seti, kronik böbrek hastalığının tahmin edilmesi için kullanılabilir. Şöyle ki, farklı özelliklere göre 'ckd' – kronik böbrek hastalığı veya 'notckd' – kronik böbrek hastalığı olmayan şeklinde sınıflandırma yapılabilir.

Kısaca hastanın kronik böbrek hastalığından muzdarip olup olmadığı makine öğrenimi sınıflandırma ile tahmin edilmektedir.

Veri seti 26 Sütun ve 280 satırdan oluşmaktadır. Veri seti aşağıdaki sütunları içermektedir [2].

bp - blood pressure	hemo - hemoglobin	sc - serum creatinine	appet - appetite
sg - specific gravity	pcv - packed cell volume	sod - sodium	pe - pedal edema
al - albumin	ba - bacteria	pot - potassium	ane - anemia
su - sugar	bgr - blood glucose random	wc - white blood cell count	class - class
rbc - red blood cells	bu - blood urea	rc - red blood cell count	age - age
pc - pus cell	dm - diabetes mellitus	htn - hypertension	
pcc - pus cell clumps	cad - coronary artery disease	id	

Sınıflandırmada kullanım için aşağıdaki sütunlar belirlenmiştir.

Y	X						
classification	bgr – rastgele kan şekeri	bu – kan üre	sc – serum kreatinin	sod – sodyum	pot – potasyum	hemo – hemoglobin	pcv – paketlenmiş hücre hacmi

Veri hazırlık süreci

Genel bakış

Pandas ve Numpy kütüphaneleri import edildikten sonra df=pd.read_csv("kdtrain.csv",sep=",") ile Excel'den veri çekilerek incelenmek üzere Dataframe'ye atıldı.

print(df.head()), print(df.info()), df.dtypes, print(df.describe()) komutlarıyla verisetine genel bakış yapıldı.

print(df.head())

```
import pandas as pd
import numpy as np

df=pd.read_csv("kdtrain.csv",sep=',')
print(df.head())
```

```
    id   age     bp     sg   al   su
0  157  62.0   70.0  1.025  3.0  0.0
1  109  54.0   70.0    NaN  NaN  NaN
2   17  47.0   80.0    NaN  NaN  NaN
3  347  43.0   60.0  1.025  0.0  0.0
4   24  42.0  100.0  1.015  4.0  0.0

           ba  ...   pcv    wc   rc
0  notpresent  ...  39.0  7900  3.9
1  notpresent  ...   NaN   NaN  NaN
2  notpresent  ...   NaN   NaN  NaN
3  notpresent  ...  43.0  7200  5.5
4     present  ...  39.0  8300  4.6
```

```
print(df.describe())
```

```
                id         age          bp
count  280.000000  275.000000  271.000000
mean   202.928571   51.454545   76.051661
std    111.988168   17.476176   14.256289
min      1.000000    2.000000   50.000000
25%    110.500000   42.000000   70.000000
50%    202.000000   55.000000   70.000000
75%    302.250000   65.000000   80.000000
max    399.000000   90.000000  180.000000
```

Sayısal değerler üzerinde inceleme yapmamızı sağlayan print(df.describe()) ile her sütun için sütundakı veri sayısı, her sütun ortalama ve sapma değeri, min ve max değeri incelendi.

print(df.info()) komutuyla incelenme zamanı bazı sütunlarda boş değerler olduğu görüldü. Fazla eksik değeri bulunan sütunların eğitimde kullanılmaması düşünüldü.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 280 entries, 0 to 279
Data columns (total 26 columns):
id                280 non-null int64
age               275 non-null float64
bp                271 non-null float64
sg                244 non-null float64
al                245 non-null float64
su                242 non-null float64
rbc               173 non-null object
pc                230 non-null object
pcc               276 non-null object
ba                276 non-null object
bgr               247 non-null float64
bu                266 non-null float64
sc                268 non-null float64
sod               213 non-null float64
pot               212 non-null float64
hemo              241 non-null float64
pcv               229 non-null float64
wc                203 non-null object
rc                187 non-null object
htn               279 non-null object
dm                279 non-null object
cad               279 non-null object
appet             280 non-null object
pe                280 non-null object
ane               280 non-null object
classification    280 non-null object
dtypes: float64(12), int64(1), object(13)
memory usage: 57.0+ KB
```

Özellik seçimi

Veri setinden eğitimini yapmak için bgr, bu, sc, sod, pot, hemo, pcv, classification sütunlarına karar verildi. Boş veri fazla olan, sonucu direkt etkiliyeceği düşünülen, kategorik ve sayısal olmayan verilerin kullanılmamasına karar verildi. Bulunan sütun sayısının fazlalığından tek tek kullanılmayacak sütunları silmek yerine sadece gerekli sütunlar loc komutuyla seçilerek dataframe'nin son haline aktarıldı.

```
df=df.loc[:,["bgr","bu","sc","sod","pot","hemo","pcv","classification"]]
print(df.head())
```

```
    bgr    bu   sc    sod  pot  hemo   pcv classification
0  122.0  42.0  1.7  136.0  4.7  12.6  39.0            ckd
1  233.0  50.1  1.9    NaN  NaN  11.7   NaN            ckd
2  114.0  87.0  5.2  139.0  3.7  12.1   NaN            ckd
3  108.0  25.0  1.0  144.0  5.0  17.8  43.0         notckd
4    NaN  50.0  1.4  129.0  4.0  11.1  39.0            ckd
```

Veri temizliği ve dönüşümü

Kayıp verilerin tespiti yapıldı. Birçok sütunda kayıp verilerin olduğu görüldü.

```
print(df.isnull().sum())
```

```
bgr               33
bu                14
sc                12
sod               67
pot               68
hemo              39
pcv               51
classification     0
dtype: int64
```

Eksik verilerin doldurulması

Regresyondan olduğu gibi sınıflandırma için de verilerin doldurulması Interpolasyon yöntemiyle yapıldı. Öğrenmede Forward Interpolasyon (ileri interpolasyon) yöntemi kullanılmıştır.

```
#replacing
df["bgr"]=df["bgr"].interpolate()
df["bu"]=df["bu"].interpolate()
df["sc"]=df["sc"].interpolate()
df["sod"]=df["sod"].interpolate()
df["pot"]=df["pot"].interpolate()
df["hemo"]=df["hemo"].interpolate()
df["pcv"]=df["pcv"].interpolate()
```

Interpolasyon sonrası sütunlarda boş verilen kalmadığı görüldü.

```
print(df.isnull().sum())
```

```
bgr               0
bu                0
sc                0
sod               0
pot               0
hemo              0
pcv               0
classification    0
dtype: int64
```

Veri tiplerinin dönüşümü

Öncelikle sütunların veri tipleri incelendi.

```
df.dtypes
```

```
bgr                float64
bu                 float64
sc                 float64
sod                float64
pot                float64
hemo               float64
pcv                float64
classification      object
dtype: object
```

Sınıflandırma işlemine kötü etkileri olmaması adına sadece tamsayılar veya içerisinden az sayıda ondalık sayı bulunan sütunlar integer tipine dönüştürüldü.

```
donusum = {"bgr": int,
           "bu": int,
           "sod":int,
           "pcv":int}
df=df.astype(donusum)
print(df.dtypes)
```

```
bgr                int32
bu                 int32
sc               float64
sod                int32
pot              float64
hemo             float64
pcv                int32
classification    object
dtype: object
```

Tekrarlayan verilerin kaldırılması

Tekrarlayan veriler `print(df[df.duplicated()])` komutuyla tespit edildi.

```
print(df[df.duplicated()])
```

```
Empty DataFrame
Columns: [bgr, bu, sc, sod, pot, hemo, pcv, classification]
Index: []
```

Komut sonrası dataframe'de tekrarlayan veri tespit edilmedi.

Aşırı verilerin kontrolü

Her sütun için ortalama değer, sapma değeri ve ortanca değeri kontrol edildi. Ortalama ve ortanca değerlerinin yakın olması gerektiğinden farklılık olup olmadığı tüm sütunlar için kontrol edildi.

```
print(df["bgr"].mean())
print(df["bgr"].std())
print(np.median(df["bgr"]))
```

```
150.475
74.41518288927637
124.5
```

```
print(df["bu"].mean())
print(df["bu"].std())
print(np.median(df["bu"]))
```

```
57.767857142857146
51.85981623757039
41.5
```

```
print(df["sc"].mean())
print(df["sc"].std())
print(np.median(df["sc"]))
```

```
3.065535714285717
5.821054640813501
1.35
```

```
print(df["sod"].mean())
print(df["sod"].std())
print(np.median(df["sod"]))
```

```
137.18214285714285
10.741623512227722
137.5
```

```
print(df["pot"].mean())
print(df["pot"].std())
print(np.median(df["pot"]))
```

```
4.965357142857145
4.08347488007521
4.477500000000001
```

```
print(df["hemo"].mean())
print(df["hemo"].std())
print(np.median(df["hemo"]))
```

```
12.531428571428574
2.7775044924451002
12.75
```

Kontrol sonrası sütunlarda aşırı veri bulunmadığı düşünüldü.

```
print(df["pcv"].mean())
print(df["pcv"].std())
print(np.median(df["pcv"]))
```

```
39.135714285714286
8.345763387075912
40.0
```

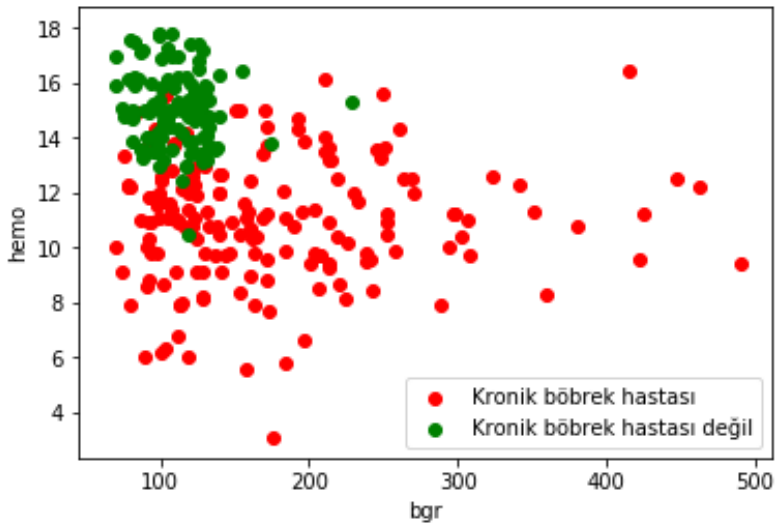
Grafikte görselleştirme

Öncelikle classification sütunundaki ckd(kronik böbrek hastası) ve notckd(kronik böbrek hastası olmayan) değerleri ckd ve notckd değişkenlerine atandı. Bu grafikte 2 sütunun ckd ve notckd durumlarına incelenmesi için önem arz etmektedir.

```
notckd=df[df.classification=="notckd"]
ckd=df[df.classification=="ckd"]
```

```
import matplotlib.pyplot as plt
plt.scatter(ckd.bgr,ckd.hemo,color="red",label="Kronik böbrek hastası")
plt.scatter(notckd.bgr,notckd.hemo,color="green",label="Kronik böbrek hastası değil")
plt.xlabel("bgr")
plt.ylabel("hemo")
plt.legend()
plt.show()
```

Rastgele kan şekeri (bgr) ve hemoglobin (hemo) değerlerinin birlikte hastalığa etkisini incelemek amacıyla yapılan gafikten değerleri bgr için 70-15 ve hemo için 12-18 arasında olan deneklerin kornik böbrek hastası olmadığı kabaca söylenebilir.



Classificaiton sütununda bulunan ckd ve notckd değerleri df.classification=[1 if i =="ckd" else 0 for i in df.classification] ile uygun olarak 1 ve 0 olarak değiştirildi.

```
df.classification=[1 if i =="ckd" else 0 for i in df.classification]
print(df.head())
```

```
   bgr  bu   sc  sod  pot  hemo  pcv  classification
0  122  42  1.7  136  4.7  12.6   39               1
1  233  50  1.9  137  4.2  11.7   40               1
2  114  87  5.2  139  3.7  12.1   41               1
3  108  25  1.0  144  5.0  17.8   43               0
4  184  50  1.4  129  4.0  11.1   39               1
```

Normalizasyon

X değerlerinin Y ye etkisini aynı aralığa çekmek için kullanılmaktadır. Normalizasyonu yapılan sütunun min değeri 0, max değeri ise 1 olarak değiştirilir. Diğer veriler formule göre bu aralıkta hesaplanır. Değer – sütun min değeri / sütun max değeri – sütun min değeri şeklinde hesaplanır. Regresyon için skorlarda çok ta fark yaratmasa da sınıflandırma için normalizasyon önemlidir. Y sütununda herhangi değişiklik olmayacağı için y dışındaki tüm sütunlar için normalizasyon yapıldı.

```
x_data=df.drop(["classification"],axis=1)
```

```
x=(x_data-np.min(x_data))/(np.max(x_data)-np.min(x_data))
```

Normalizasyon sonrası sütunların son hali aşağıdaki gibidir.

```
print(x)
```

```
          bgr        bu        sc       sod       pot      hemo       pcv
0    0.123810  0.083990  0.017196  0.830189  0.045147  0.646259  0.666667
1    0.388095  0.104987  0.019841  0.836478  0.033860  0.585034  0.688889
2    0.104762  0.202100  0.063492  0.849057  0.022573  0.612245  0.711111
3    0.090476  0.039370  0.007937  0.880503  0.051919  1.000000  0.755556
4    0.271429  0.104987  0.013228  0.786164  0.029345  0.544218  0.666667
5    0.454762  0.125984  0.023810  0.685535  0.006772  0.761905  0.666667
6    0.030952  0.102362  0.006614  0.849057  0.013544  0.979592  0.688889
7    0.097619  0.060367  0.007937  0.918239  0.051919  0.829932  0.777778
8    0.164286  0.023622  0.006614  0.867925  0.066591  0.653061  0.733333
9    0.078571  0.181102  0.064815  0.823899  0.081264  0.217687  0.222222
10   0.147619  0.020997  0.009259  0.899371  0.045147  0.721088  0.800000
```

Modeli eğitmek

X değişkenine değerler normalizasyon sırasında atıldığı için sadece y değişkenine **classification** sütunun değerleri atıldı.

```
y=df.classification.values
```

K-NEAREST NEIGHBORS – K-En Yakın Komşu Algoritması

En yakın komşu algoritmasında öncelikle eğitim ve değerlendirme için gerekli sklearn kütüphaneleri eklendi.

Veri setini eğitim ve test kısımlarına ayırmak için ise Sklearn.model_selection den Train_test_split import edildi. Ardından test boyutu olarak 0.2 seçildi. Bu tüm verilerin %20'nin test için ayrıldığı demektir. En yakın komşu sayısı olarak (K) 2 seçildi.

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
x_train, x_test,y_train, y_test=train_test_split(x,y,test_size=0.20,random_state=1)

knn=KNeighborsClassifier(n_neighbors=2)
knn.fit(x_train,y_train)

knn_pred=knn.predict(x_test)

print(confusion_matrix(y_test, knn_pred))
print(classification_report(y_test, knn_pred))
knn_score=knn.score(x_test,y_test)
print(knn_score)
```

```
[[20  1]
 [ 2 33]]
              precision    recall  f1-score   support

           0       0.91      0.95      0.93        21
           1       0.97      0.94      0.96        35

   micro avg       0.95      0.95      0.95        56
   macro avg       0.94      0.95      0.94        56
weighted avg       0.95      0.95      0.95        56

0.9464285714285714
```

En yakın komşu sayısı olarak (K) 2 seçildiğinde Knn için R2 skoru = 0.946 F1 skoru ise hasta olmayanlar için 0.93, hastalar için ise 0.96, ortalama 0.95 olarak ölçüldü.

Modelin değerlendirilmesinde R2 skorun yanı sıra F1 skor da kullanılmıştır. F1 skoru karışıklık Matrisinde (Confusion Matrix) bulunan Precision ve Recall değerlerine göre hesaplanmaktadır.

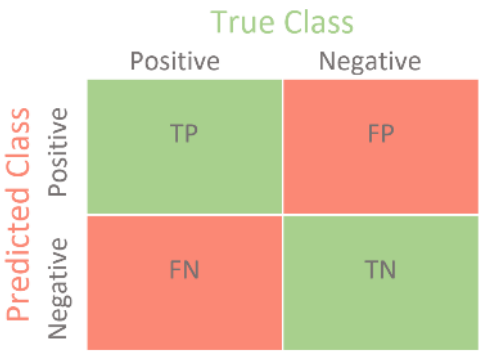
Karışıklık Matrisinde tahminlerin doğru mu yanlış mı olduğunu dört değere bakarak kontrol edilmektedir.

TN / True Negative: vaka negatifti ve negatif tahmin edildi.

TP / Gerçek Pozitif: vaka pozitifti ve pozitif tahmin edildi.

FN / Yanlış Negatif: vaka olumluydu ancak olumsuz tahmin edildi.

FP / Yanlış Pozitif: vaka olumsuzdu ancak olumlu tahmin edildi.



Tahminlerinizin yüzde kaçı doğruydu?

Precision – olumlu tahminlerin doğruluğunu göstermektedir. Bir sınıflandırıcının aslında negatif olan bir örneği pozitif olarak etiketlememe yeteneğidir. Her sınıf için, gerçek pozitiflerin gerçek pozitif ve yanlış pozitiflerin toplamına oranı olarak tanımlanır.

$$\text{Precision} = TP / (TP + FP)$$

Olumlu vakaların yüzde kaçı yakalandı?

Recall – bir sınıflandırıcının tüm pozitif örnekleri bulma yeteneğidir. Her sınıf için gerçek pozitiflerin gerçek pozitiflerin ve yanlış negatiflerin toplamına oranı olarak tanımlanır.

$$\text{Recall} = TP / (TP + FN)$$

Olumlu tahminlerin yüzde kaçı doğruydu?

F1 skoru, en iyi skor 1.0 ve en kötüsü 0.0 olacak şekilde, Precision ve Recall 'ın ağırlıklı harmonik ortalamasıdır. F1 puanları, hesaplamalarına hassasiyet ve geri çağırma ekledikleri için doğruluk (Accuracy) ölçümlerinden daha düşüktür. Genel bir kural olarak, genel doğruluğu değil sınıflandırıcı modellerini karşılaştırmak için F1'in ağırlıklı ortalaması kullanılmalıdır.

$$\text{F1 Puanı} = 2 * (\text{Recall} * \text{Precision}) / (\text{Recall} + \text{Precision})$$

$$\text{Accuracy} = TP + TN / (TP + TN + FP + FN)$$

Support, sınıfın belirtilen veri kümesindeki gerçek oluşumlarının sayısıdır. Eğitim verilerindeki dengesiz support, sınıflandırıcının rapor edilen puanlarındaki yapısal zayıflıkları göstermektedir.

Micro avg: farklı kümeler için sistemin bireysel gerçek pozitiflerini (TP), yanlış pozitiflerini (FP) ve yanlış negatiflerini (FN) toplar ve bunları istatistik almak için uygun işlemleri yapar.

$$\text{Micro-average of precision} = (TP1+TP2) / (TP1+TP2+FP1+FP2)$$

$$\text{Micro-average of recall} = (TP1+TP2) / (TP1+TP2+FN1+FN2)$$

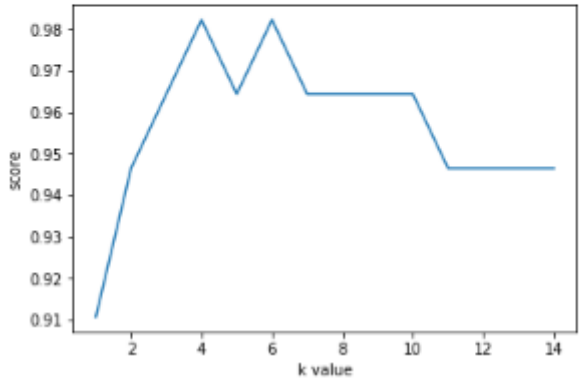
Micro avg: farklı setlerde sistemin Precision ve Recall değerlerinin ortalaması.

Weighted avg: bir veri kümesindeki bazı faktörlerin göreceli önemini veya sıklığını hesaba katan ortalamadır.

KNN algoritmasının K nın hangi değerinde maksimum skora ulaştığı tespit edildi. Grafikten K'nın 4 ve 6 değerlerinde max skora (R2=0.98) ulaşacağı görüldü.

```
score_list=[]
for i in range(1,15):
    knn2=KNeighborsClassifier(n_neighbors=i)
    knn2.fit(x_train,y_train)
    score_list.append(knn2.score(x_test,y_test))

plt.plot(range(1,15),score_list)
plt.xlabel("k value")
plt.ylabel("score")
plt.show()
```



En yakın komşu sayısı olarak (K) 4 seçildiğinde KNN için

R2 skoru = 0.9821

F1 skoru ise hasta olmayanlar için 0.98, hastalar için ise 0.99 , ortalama 0.98 olarak ölçüldü.

```
knn=KNeighborsClassifier(n_neighbors=4)
knn.fit(x_train,y_train)

knn_pred=knn.predict(x_test)

print(confusion_matrix(y_test, knn_pred))
print(classification_report(y_test, knn_pred))
knn_score=knn.score(x_test,y_test)
print(knn_score)
```

```
[[21  0]
 [ 1 34]]
              precision    recall  f1-score   support

           0       0.95      1.00      0.98        21
           1       1.00      0.97      0.99        35

    micro avg       0.98      0.98      0.98        56
    macro avg       0.98      0.99      0.98        56
 weighted avg       0.98      0.98      0.98        56

0.9821428571428571
```

SVM

SVM için

R2 skoru = 0.9821

F1 skoru ise hasta olmayanlar için 0.93, hastalar için ise 0.96, ortalama 0.95 olarak ölçüldü.

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.svm import SVC
svm = SVC(kernel='linear')
svm.fit(x_train, y_train)

y_pred = svm.predict(x_test)

print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print("svm score:",svm.score(x_test,y_test))
```

```
[[20  1]
 [ 2 33]]
              precision    recall  f1-score   support

           0       0.91      0.95      0.93        21
           1       0.97      0.94      0.96        35

    micro avg       0.95      0.95      0.95        56
    macro avg       0.94      0.95      0.94        56
 weighted avg       0.95      0.95      0.95        56

svm score: 0.9464285714285714
```

Decision Tree

Decision Tree için

Test size=0.2

R2 skoru = 0.875

F1 skoru ise hasta olmayanlar için 0.84, hastalar için ise 0.90, ortalama 0.88 olarak ölçüldü.

```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier(random_state=1)
dt.fit(x_train,y_train)

y_pred = dt.predict(x_test)

from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
print("dt score:",dt.score(x_test,y_test))
```

```
[[19  2]
 [ 5 30]]
              precision    recall  f1-score   support

           0       0.79      0.90      0.84        21
           1       0.94      0.86      0.90        35

   micro avg       0.88      0.88      0.88        56
   macro avg       0.86      0.88      0.87        56
weighted avg       0.88      0.88      0.88        56

dt score: 0.875
```

Decision Tree için

Test size=0.1

R2 skoru = 0.964

F1 skoru ise hasta olmayanlar için 0.96, hastalar için ise 0.97, ortalama 0.96 olarak ölçüldü.

```
x_train, x_test,y_train, y_test=train_test_split(x,y,test_size=0.10,random_state=1)
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier(random_state=1)
dt.fit(x_train,y_train)

y_pred = dt.predict(x_test)

from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
print("dt score:",dt.score(x_test,y_test))
```

```
[[11  0]
 [ 1 16]]
              precision    recall  f1-score   support

           0       0.92      1.00      0.96        11
           1       1.00      0.94      0.97        17

   micro avg       0.96      0.96      0.96        28
   macro avg       0.96      0.97      0.96        28
weighted avg       0.97      0.96      0.96        28

dt score: 0.9642857142857143
```

Random Forest

Random Forest için

Test size=0.2

Tree sayısı=100

R2 skoru = 0.964

F1 skoru ise hasta olmayanlar için 0.95, hastalar için ise 0.97, ortalama 0.96 olarak ölçüldü.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

x_train, x_test,y_train, y_test=train_test_split(x,y,test_size=0.20,random_state=1)
rf=RandomForestClassifier(n_estimators=100, random_state=1)
rf.fit(x_train,y_train)

y_pred = rf.predict(x_test)

from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
print("dt score:",rf.score(x_test,y_test))
```

```
[[20  1]
 [ 1 34]]
              precision    recall  f1-score   support

           0       0.95      0.95      0.95        21
           1       0.97      0.97      0.97        35

   micro avg       0.96      0.96      0.96        56
   macro avg       0.96      0.96      0.96        56
weighted avg       0.96      0.96      0.96        56

dt score: 0.9642857142857143
```

Sınıflandırma için en iyi skor KNN algoritmasında görülmüştür. F1=0.98, R2=0.982

KAYNAKÇA

[1] https://www.kaggle.com/kumarajarshi/life-expectancy-who

[2] https://www.kaggle.com/colearninglounge/chronic-kidney-disease