KNN & Lojistik Regresyon ile Diabet Veri Seti İncelemesi

Ali Kerem Şimşek

Veri Setinin Hikayesi

21 yaş ve üzeri olan Pima Indian kadınları üzerinde yapılmış bazı test sonuçlarını içerir.
768 gözlem ve 8 sayısal bağımsız değişkenden oluşmaktadır.
Hedef değişkenin «outcome» olduğu belirlenmiş olup;
-1 diyabet test sonucunun pozitif olduğunu,
-0 ise diyabet test sonucunun negatif olduğunu belirtmektedir.

AMACIMIZ :
Özellikleri belirtildiğinde kişilerin diyabet hastası olup olmadıklarını tahmin edebilecek bir model geliştirmek.
Biz burada seçtiğimiz iki algoritmayı kıyaslayacağız ve tahmin oranı en yüksek olan modeli bulup, test edeceğiz.

DEĞİŞKENLER :
1. Pragnancies : Hamilelik Sayısı
2. Glucose : Glikoz
3. BloodPressure : Kan Basıncı
4. SkinThickness : Cilt Klaınlığı
5. Insulin : İnsülin
6. BMI : Vücut Kitle Endeksi
7. DiabbetesPedigreeFunction : Soyumuzdaki kişilere göre diyabet ihtimalimizi hesaplayan bir fonksiyon
8. Age : Yaş
9. Outcome : Kişinin diyabet olup olmadığının bilgisi

Lojistik Regresyon

Lojistik Regresyon Analizi bağımlı değişkenin tahmini değerlerini olasılık olarak hesaplayarak,
olasılık kurallarına uygun sınıflama yapma imkanı veren bir yöntemdir.
İki veri faktörü arasındaki ilişkileri bulmak için matematikten yararlanır.
Tahminin genellikle **evet** ya da **hayır** gibi sınırlı sayıda sonucu vardır.

Lojistik Regresyon ‘un diğer modellere göre daha çok tercih edilmesinin sebeplerini şöyle sıralayabiliriz;

1. **Basitlik** : Uygulaması ve anlaması kolay.
2. **Hız** : Büyük hacimli verileri işlerken daha az karmaşık olduğu için, daha hızlı yanıt verirler.
3. **Esneklik** : İki veya daha fazla sınırlı sonucu olan soruların yanıtlarını bulmak için kullanılabilir.
4. **Görünürlük** : Hesaplamalar daha az karmaşık olduğundan sorun giderme ve hata düzeltme de daha kolaydır.

Üretim, Sağlık hizmetleri (Tıbbi Araştırmalar), Finans, Pazarlama gibi alanlarda sıklıkla tercih edilir. Benim tercih etme sebebim de tıbbi araştırmalarda, hastalarda hastalık olasılığını tahmin etmede sıklıkla kullanılıyor olmasıdır.

K-Nearest Neighbors (KNN)

KNN en basit anlamı ile içerisinde tahmin edilecek değerin bağımsız değişkenlerinin oluşturduğu vektörün en yakın komşularının hangi sınıfta yoğun olduğu bilgisi üzerinden sınıfını tahmin etmeye dayanır.

KNN (K-Nearest Neighbors) Algoritması iki temel değer üzerinden tahmin yapar;

1. **Distance (Uzaklık):** Tahmin edilecek noktanın diğer noktalara uzaklığı hesaplanır.

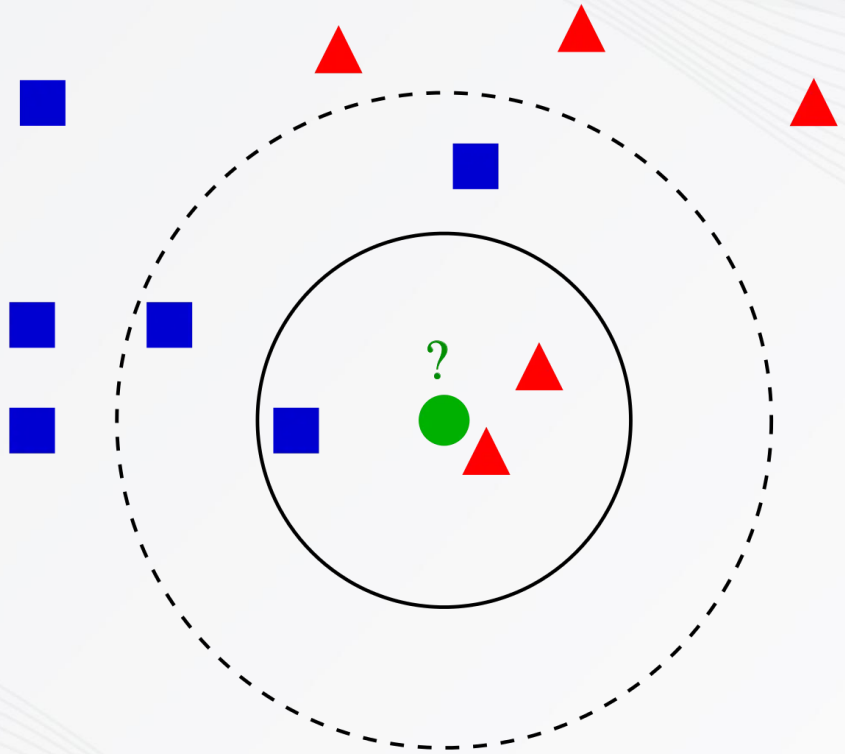
Bunun için Minkowski uzaklık hesaplama fonksiyonu kullanılır.

2. **K (komuşuluk sayısı):** En yakın kaç komşu üzerinden hesaplama yapılacağını söyleriz.

K değeri sonucu direkt etkileceyecektir. K 1 olursa overfit etme olasılığı çok yüksek olacaktır.
Çok büyük olursa da çok genel sonuçlar verecektir. Bu sebeple,
optimum K değerini tahmin etmek problemin asıl konusu olarak karşımızda durmaktadır.

K değerinin önemini aşağıdaki grafik çok güzel bir şekilde göstermektedir.

K-Nearest Neighbors (KNN)



Eğer K=3 (düz çizginin olduğu yer) seçersek sınıflandırma algoritması
? işareti ile gösterilen noktayı, kırmızı üçgen sınıfı olarak tanımlayacaktır.
Fakat K=5 (kesikli çizginin olduğu alan) seçersek sınıflandırma algoritması, aynı noktayı mavi kare sınıfı olarak tanımlayacaktır.

Lojistik Regresyon

```
import pandas as pd
import seaborn as sns

import matplotlib
matplotlib.use('Qt5Agg')
import matplotlib.pyplot as plt

from sklearn.preprocessing import RobustScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix, classification_report, plot_roc_curve
from sklearn.model_selection import train_test_split, cross_validate

# KEŞİFÇİ VERİ ANALİZİ
df = pd.read_csv("datasets/diabetes.csv")
df.head()
df.shape
```

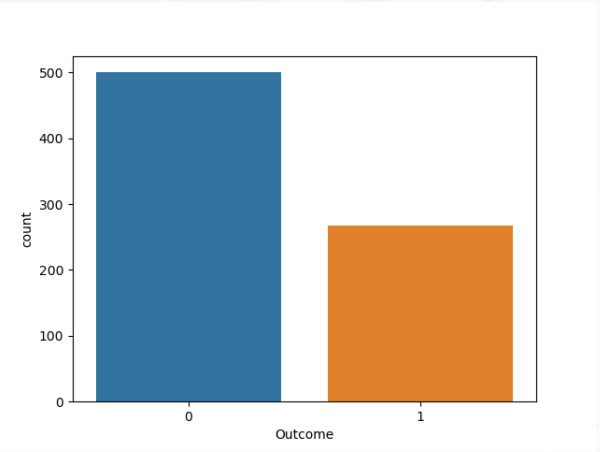
```
Out[36]: (768, 9)
```

Lojistik Regresyon

```python
# 1.Target Analizi
df["Outcome"].value_counts() # Hangi sınıftan kaç tane var?

sns.countplot(x="Outcome", data=df)
plt.show()
```

```
In [37]: df["Outcome"].value_counts() # Hangi sınıftan kaç tane var?
Out[37]:
0    500
1    268
```




```python
# 2.Feature Analizi
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.000	3.845	3.370	0.000	1.000	3.000	6.000	17.000
Glucose	768.000	120.895	31.973	0.000	99.000	117.000	140.250	199.000
BloodPressure	768.000	69.105	19.356	0.000	62.000	72.000	80.000	122.000
SkinThickness	768.000	20.536	15.952	0.000	0.000	23.000	32.000	99.000
Insulin	768.000	79.799	115.244	0.000	0.000	30.500	127.250	846.000
BMI	768.000	31.993	7.884	0.000	27.300	32.000	36.600	67.100
DiabetesPedigreeFunction	768.000	0.472	0.331	0.078	0.244	0.372	0.626	2.420
Age	768.000	33.241	11.760	21.000	24.000	29.000	41.000	81.000
Outcome	768.000	0.349	0.477	0.000	0.000	0.000	1.000	1.000

Lojistik Regresyon

```python
cols = [col for col in df.columns if "Outcome" not in col] # Bağımlı değişkeni çıkartıp geri kalan sütunları listeledik.
# 3.Target ve Features Bir Arada Değerlendirme
# Fonksiyonlaştırdığımızda:
def target_summary_with_num(dataframe, target, numerical_col):
    print(dataframe.groupby(target).agg({numerical_col: "mean"}), end="\n\n\n")

for col in cols:
    target_summary_with_num(df, "Outcome", col)
```

```
         Pregnancies
Outcome
0              3.298
1              4.866


         Glucose
Outcome
0        109.980
1        141.257


         BloodPressure
Outcome
0               68.184
1               70.825


         SkinThickness
Outcome
0               19.664
1               22.164


         Insulin
Outcome
0         68.792
1        100.336


           BMI
Outcome
0       30.304
1       35.143


         DiabetesPedigreeFunction
Outcome
0                           0.430
1                           0.550


           Age
Outcome
0       31.190
1       37.067
```

Lojistik Regresyon

```python
# MODELLEME & TAHMİN (MODEL & PREDICTION)
y = df["Outcome"] #Bağımlı değişkenimiz.
X = df.drop(["Outcome"], axis=1) #Bağımsız değişkenlerimiz.

log_model = LogisticRegression().fit(X, y)

y_pred = log_model.predict(X) #Bağımsız değişkenleri kullanarak, bağımlı değişkenimizi tahmin ettirdik ve y_pred 'e atadık.
y_pred[0:10] #İlk 10 tanesine bakmak istersek.
y[0:10] #Gerçek değerlerin ilk 10 tanesine de bakalım ve karşılaştıralım.
```

```
In [61]: y_pred[0:10] #İlk 10 tanesine bakmak istersek.
Out[61]: array([1, 0, 1, 0, 1, 0, 0, 1, 1, 0], dtype=int64)
In [62]: y[0:10] #Gerçek değerlerin ilk 10 tanesine de bakalım ve karşılaştıralım.
Out[62]:
0    1
1    0
2    1
3    0
4    1
5    0
6    1
7    0
8    1
9    1
Name: Outcome, dtype: int64
```

Lojistik Regresyon

```
print(classification_report(y, y_pred)) #Precision, Recall, F1 Score, Support(Sınıfların Frekansları) getirir.
#Precision : 0.74 = 1 olarak yaptığımız tahminlerin %74 'ü başarılıymış.
#Recall : 0.58 = 1 olarak yaptığımız tahminlerin %58 'i başarılıymış.
#F1 Score : 0.65 = 1 olarak yaptığımız tahminlerin %65 'i başarılıymış.
#Accuracy : 0.78
```

	precision	recall	f1-score	support
0	0.80	0.89	0.84	500
1	0.74	0.57	0.65	268
accuracy			0.78	768
macro avg	0.77	0.73	0.75	768
weighted avg	0.78	0.78	0.77	768

Lojistik Regresyon

```python
# MODEL DOĞRULAMA
# Model Validation : Holdout
# Holdout : Verisetini iki parçaya böl. Biriyle modeli eğit, diğeriyle test et.

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=17) #Veri setini 80 'e 20 olacak şekilde böldük.
# Random_state=17 rastgele oluşturduğumuz train,test veriseti için randomluğumuzun kodu 17 gibi düşünebiliriz. Aynı değerleri alabilmemiz için.

lod_model = LogisticRegression().fit(X_train, y_train)

y_pred = log_model.predict(X_test) #Eğitim yaparken kullanmadığımız bağımsız değişkenler üzerinden tahmin yaptırdık. Daha sonra yine eğitimde kullanmadığımız y (gerçek değerler) ile kıyaslayacağız.

y_prob = log_model.predict_proba(X_test)[:, 1]

print(classification_report(y_test, y_pred))
```

```
In [68]: print(classification_report(y_test, y_pred))
              precision    recall  f1-score   support

           0       0.77      0.92      0.84        97
           1       0.79      0.54      0.65        57

    accuracy                           0.78       154
   macro avg       0.78      0.73      0.74       154
weighted avg       0.78      0.78      0.77       154
```

Lojistik Regresyon

```python
# 10 KATLI ÇAPRAZ DOĞRULAMA (10 FOLD CROSS VALIDATION)
# Hangi 80 'e 20 almalıyız. Modelin doğrulama sürecini en doğru şekilde ele almak için yaparız.

y = df["Outcome"]
X = df.drop(["Outcome"], axis=1)

log_model = LogisticRegression().fit(X, y)

cv_results = cross_validate(log_model, X, y, cv=5, scoring=["accuracy", "precision", "recall", "f1", "roc_auc"]) # 5 katlı cross validate yapacağımızı belirledik.

cv_results['test_accuracy'] #5 katlı yaptığımız için verisetini 5 farklı şekilde iki parçaya bölüp değerlendirdikten sonra accuracy sonuçlarını getirdi.
cv_results['test_accuracy'].mean() #0.77
cv_results['test_precision'].mean() #0.71
cv_results['test_recall'].mean() #0.57
cv_results['test_f1'].mean() #0.63
cv_results['test_roc_auc'].mean() #0.83
```

```
In [73]: cv_results['test_accuracy'] #5 katlı yaptığımız için verisetini 5 farklı şekilde iki parçaya bölüp değerlendirdikten sonra accuracy sonuçlarını getirdi.
Out[73]: array([0.77272727, 0.74675325, 0.75974026, 0.81699346, 0.75163399])
In [74]: cv_results['test_accuracy'].mean() #0.77
Out[74]: 0.7695696460402341
In [75]: cv_results['test_precision'].mean() #0.71
Out[75]: 0.7182238325877177
In [76]: cv_results['test_recall'].mean() #0.57
Out[76]: 0.5634521313766596
In [77]: cv_results['test_f1'].mean() #0.63
Out[77]: 0.6294528781642184
In [78]: cv_results['test_roc_auc'].mean() #0.83
Out[78]: 0.8304807826694619
```

Lojistik Regresyon

```
# SONUÇ

#Train Verisetindeki Değerler    #Test Verisetiyle Karşılaştırdığımız Değerler    #CV Sonrası Değerler (En Doğru Değerler)
#Accuracy : 0.78                 #Accuracy : 0.77                                 #Accuracy : 0.77
#Precision : 0.74                #Precision : 0.79                                #Precision : 0.71
#Recall : 0.58                   #Recall : 0.53                                   #Recall : 0.57
#F1-Score : 0.65                 #F1-Score : 0.63                                 #F1-Score : 0.63
```

K-Nearest Neighbors (KNN)

```
import pandas as pd
import matplotlib
from sklearn.preprocessing import StandardScaler
matplotlib.use('Qt5Agg')
import warnings
from sklearn.metrics import roc_auc_score, classification_report
from sklearn.model_selection import cross_validate
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier


warnings.simplefilter(action="ignore")

pd.set_option('display.max_columns', None) # Bütün sütunları göster.
pd.set_option('display.float_format', lambda x: '%.3f' % x) # Virgülden sonra üç basamak göster.
pd.set_option('display.width', 500)

df = pd.read_csv("datasets/diabetes.csv")
df.head()
```

```
Out[6]:
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin    BMI  DiabetesPedigreeFunction  Age  Outcome
0            6      148             72             35        0 33.600                     0.627   50        1
1            1       85             66             29        0 26.600                     0.351   31        0
2            8      183             64              0        0 23.300                     0.672   32        1
3            1       89             66             23       94 28.100                     0.167   21        0
4            0      137             40             35      168 43.100                     2.288   33        1
```

K-Nearest Neighbors (KNN)

```
##################################### EXPLORATORY DATA ANALYSIS ##################################################

def check_df(dataframe, head=5):
    print("##################### Shape #####################")
    print(dataframe.shape) # 768 değişken, 9 gözlem birimimiz var.
    print("##################### Types #####################")
    print(dataframe.dtypes) # 2 float ve 7 int.
    print("##################### Head #####################")
    print(dataframe.head(head)) # Bağımlı değişkenimiz Outcome, int tipinde.
    print("##################### Tail #####################")
    print(dataframe.tail(head))
    print("##################### NA #####################")
    print(dataframe.isnull().sum()) # Hiç boş değerimiz yok.
    print("################### Quantiles ###################")
    print(dataframe.quantile([0, 0.05, 0.50, 0.95, 0.99, 1]).T)
    print("################### Y Classes ###################")
    print(df["Outcome"].value_counts())

check_df(df)
```

K-Nearest Neighbors (KNN)

```
In [8]: check_df(df)
##################### Shape #####################
(768, 9)
##################### Types #####################
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
##################### Head #####################
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin    BMI  DiabetesPedigreeFunction  Age  Outcome
0            6      148             72             35        0 33.600                     0.627   50        1
1            1       85             66             29        0 26.600                     0.351   31        0
2            8      183             64              0        0 23.300                     0.672   32        1
3            1       89             66             23       94 28.100                     0.167   21        0
4            0      137             40             35      168 43.100                     2.288   33        1
##################### Tail #####################
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin    BMI  DiabetesPedigreeFunction  Age  Outcome
763           10      101             76             48      180 32.900                     0.171   63        0
764            2      122             70             27        0 36.800                     0.340   27        0
765            5      121             72             23      112 26.200                     0.245   30        0
766            1      126             60              0        0 30.100                     0.349   47        1
767            1       93             70             31        0 30.400                     0.315   23        0
##################### NA #####################
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
################### Quantiles ###################
                          0.000  0.050   0.500   0.950   0.990   1.000
Pregnancies               0.000  0.000   3.000  10.000  13.000  17.000
Glucose                   0.000 79.000 117.000 181.000 196.000 199.000
BloodPressure             0.000 38.700  72.000  90.000 106.000 122.000
SkinThickness             0.000  0.000  23.000  44.000  51.330  99.000
```

K-Nearest Neighbors (KNN)

```
######################################## DATA PREPROCESSING & FEATURE ENGINEERING ########################################

y = df["Outcome"]
X = df.drop(["Outcome"], axis=1)

# Daha doğru ve daha hızlı sonuçlar elde edebilmek için, elimizdeki bağımsız değişkenleri standartlaştırıyoruz.
X_scaled = StandardScaler().fit_transform(X) #Bu şekilde pek okunabilir değil.

X = pd.DataFrame(X_scaled, columns=X.columns) # Kolon isimleri X 'in kolon isimleri yaptık ve X değişkenlerimizi
# standartlaştırdık.

############################################### MODELING & PREDICTION ###############################################

knn_model = KNeighborsClassifier().fit(X, y) # KNN 'e göre bağımlı ve bağımsız değişkenler arasındaki ilişkiyi öğrendik.

random_user = X.sample(1, random_state=45) # Verisetinden rastgele bir kişi seçtik.

knn_model.predict(random_user) # Seçtiğimiz bu kişiyi tahminm ettirdik.
```

K-Nearest Neighbors (KNN)

```
################################################ MODELING & PREDICTION ################################################

knn_model = KNeighborsClassifier().fit(X, y) # KNN 'e göre bağımlı ve bağımsız değişkenler arasındaki ilişkiyi öğrendik.

random_user = X.sample(1, random_state=45) # Verisetinden rastgele bir kişi seçtik.

knn_model.predict(random_user) # Seçtiğimiz bu kişiyi tahminm ettirdik.
```

```
In [15]: knn_model.predict(random_user) # Seçtiğimiz bu kişiyi tahminm ettirdik.
Out[15]: array([1], dtype=int64)
```

```
################################################ MODEL EVALUTION ################################################


# Confusion matrix için y_pred:
y_pred = knn_model.predict(X) #Bütün gözlem birimleri için tahmin ettiğimiz değerleri y_pred 'e atadık.

# AUC için y_prob
y_prob = knn_model.predict_proba(X)[:, 1] # 1 sınıfına ait olma olasılıklarını getirdik.

print(classification_report(y, y_pred)) # 1 ve 0 sınıflarına göre hesaplama işlemleri yapıyor. Ana odağımız 1 sınıfı.
# Precision : 1 olarak tahmin ettiklerimizin başarısı.
# Recall : Gerçekte 1 olanları, 1 olarak tahmin etme başarımız.
# F1-Score : Precision ve Recall 'ın harmonik ortalaması.
```

	precision	recall	f1-score	support
0	0.85	0.90	0.87	500
1	0.79	0.70	0.74	268
accuracy			0.83	768
macro avg	0.82	0.80	0.81	768
weighted avg	0.83	0.83	0.83	768

K-Nearest Neighbors (KNN)

```python
# AUC
roc_auc_score(y, y_prob) # %90 çıktı.

cv_results = cross_validate(knn_model, X, y, cv=5, scoring=["accuracy", "f1", "roc_auc"])

cv_results['test_accuracy'].mean()
cv_results['test_f1'].mean()
cv_results['test_roc_auc'].mean()

# CV
# Accuracy : 0.73
# F1 : 0.59
# Roc_Auc : 0.78

# Bu skorları nasıl artırabiliriz?
# 1. Veriseti boyutu artırılabilir.
# 2. Veri ön işleme yapılabilir.
# 3. Özellik mühendisliği yapılabilir. (Yeni değişkenler türetilebilir)
# 4. İlgili algoritma için optimizasyonlar yapılabilir.
```

```
In [19]: roc_auc_score(y, y_prob) # %90 çıktı.
Out[19]: 0.9017686567164179
In [20]: cv_results = cross_validate(knn_model, X, y, cv=5, scoring=["accuracy", "f1", "roc_auc"])
In [21]: cv_results['test_accuracy'].mean()
Out[21]: 0.733112638994992
In [22]: cv_results['test_f1'].mean()
Out[22]: 0.5905780011534191
In [23]: cv_results['test_roc_auc'].mean()
Out[23]: 0.7805279524807827
```

K-Nearest Neighbors (KNN)

```
########################################## HYPERPARAMETER OPTIMIZATION ##################################################

knn_model = KNeighborsClassifier()
knn_model.get_params() # Burada komşuluk sayısını 5 olarak gözlemliyoruz.
# Amacımız komşuluk sayısını değiştirerek, olması gereken en optimum komşuluk sayını bulmak.
```

```
In [25]: knn_model.get_params() # Burada komşuluk sayısını 5 olarak gözlemliyoruz.
Out[25]:
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 5,
 'p': 2,
 'weights': 'uniform'}
```

```
knn_params = {"n_neighbors": range(2, 50)} # 2 'den 50 'ye kadar komuşulukları ara diyoruz.

knn_gs_best = GridSearchCV(knn_model, knn_params, cv=5, n_jobs=-1, verbose=1).fit(X, y) #n_job=-1 ise işlemcileri
# tamamen kullanır. Hızlıca çözüm almak için.
# verbose=1 ise rapor almamızı sağlar.
# Sonuç 48 candidates, yani knn_params 'da denenecek 48 tane aday var. Her birinde 5 katlı doğrulama yapıldığı için
# toplam 240 tane fit etme işlemi (model kurma) varmış.
```

```
In [27]: knn_gs_best = GridSearchCV(knn_model, knn_params, cv=5, n_jobs=-1, verbose=1).fit(X, y) #n_job=-1 ise işlemcileri
Fitting 5 folds for each of 48 candidates, totalling 240 fits
```

K-Nearest Neighbors (KNN)

```
knn_gs_best.best_params_ # 17 komşuluk sayısı ile kurarsak, model daha başarılı olurmuş. Bunu öğrendik.
```

```
In [28]: knn_gs_best.best_params_ # 17 komşuluk sayısı ile kurarsak, model daha başarılı olurmuş. Bunu öğrendik.
Out[28]: {'n_neighbors': 17}
```

```
########################################################## FINAL MODEL ##########################################################

knn_final = knn_model.set_params(**knn_gs_best.best_params_).fit(X, y) # Burada iki yıldız kullarak atama yaptığımızda,
# bu fonsiyon ile gelen dictionary türü çıktıları tek tek elle yazmamıza gerek kalmıyor.
# Burada modelimizi kurduk.


cv_results = cross_validate(knn_final, X, y, cv=5, scoring=["accuracy", "f1", "roc_auc"])

cv_results['test_accuracy'].mean()
cv_results['test_f1'].mean()
cv_results['test_roc_auc'].mean()

# Yeni CV değerleri:
# Accuracy : 0.76
# F1 : 0.61
# Roc_Auc : 0.81
```

```
# CV
# Accuracy : 0.73
# F1 : 0.59
# Roc_Auc : 0.78
```

```
In [31]: cv_results['test_accuracy'].mean()
Out[31]: 0.7669892199303965
In [32]: cv_results['test_f1'].mean()
Out[32]: 0.6170909049720137
In [33]: cv_results['test_roc_auc'].mean()
Out[33]: 0.8127938504542278
```

Sonuç

```
#KNN Sonuçları:          #Lojistik Regresyon Sonuçları:
#Accuracy : 0.76         #Accuracy : 0.77
#F1-Score : 0.61         #F1-Score : 0.63
#Roc-AUC  : 0.81         #Roc-AUC  : 0.83
```

Lojistik Regresyon ile başarım oranlarımızın daha yüksek olduğunu görebiliriz. Bu yüzden Lojistik regresyon kullanarak tahminleme yapalım.

```
# TAHMİNLEME
random_user = X.sample(1, random_state=45) #Verisetinden rastgele birini seçtik. Değerleri elle de girebilirdik.
log_model.predict(random_user) #1 sonucunu verdi. Yani modele göre bu kişi diyabet hastası.
```

```
In [47]: log_model.predict(random_user)
Out[47]: array([1], dtype=int64)
```

Kaynakça

Lojistik Regresyon hakkında temel bilgiler :
https://aws.amazon.com/tr/what-is/logistic-regression/#:~:text=Lojistik%20regresyon%2C%20iki%20veri%20fakt%C3%B6r%C3%BC,gibi%20s%C4%B1n%C4%B1rl%C4%B1%20say%C4%B1da%20sonucu%20vard%C4%B1r.

KNN hakkında temel bilgiler :
https://arslanev.medium.com/makine-%C3%B6%C4%9Frenmesi-knn-k-nearest-neighbors-algoritmas%C4%B1-bdfb688d7c5f

Teşekkürler

Ali Kerem Şimşek