KARABÜK ÜNİVERSİTESİ

LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

BİYOMEDİKAL MÜHENDİSLİĞİ

BİYOMEDİKAL MÜHENDİSLİĞİNDE YAPAY SİNİR AĞI UYGULAMALARI

FİNAL RAPORU

ADI: Yusuf Ali

SOYADI: AKAR

NO: 1928142025

CTG.csv adlı veri setini Keras ve MLP yapay sinir ağı kütüphanelerini kullanarak performans değerlendirmelerinin yapılması amaçlanmıştır.

KERAS UYGULAMA 1

Dataset Kaynağı: https://archive.ics.uci.edu/ml/datasets/Cardiotocography

CTG.csv isimli Dataset verileri, 2126 fetal kardiyotokogram (CTG) otomatik olarak işlenip ve ilgili tanı özellikleri ölçüldü. CTG'ler ayrıca üç uzman kadın doğum uzmanı ve her birine atanmış bir uzlaşma sınıflandırma etiketi ile sınıflandırılmıştır. Sınıflandırma hem morfolojik örüntü (A, B, C. ...) hem de fetal duruma (N, S, P) göredir.

Veri sınıfı modeli hem 10-sınıf hemde 3-sınıf olarak kullanılabilir. 3-sınıflı olarak modellerde kullanıldı.

```
@author: Yusuf Ali
"""

#%%
from warnings import filterwarnings
filterwarnings('ignore')
import pandas as pd
import numpy as np

df = pd.read_csv("CTG.csv")


df.head()

# Drop unnecessaries
df=df.drop(["FileName","Date","SegFile","b","e","A", "B","C", "D" ,"E", "AD", "DE" ,"LD", "FS", "SUSP"],axis=1)

df.head()
```

Liste 1

```
df.head()

# Coloumns names
df.columns

df.shape

df.isnull().sum()

# tüm  nan verileri silme işlemi
df = df.dropna()

df.isnull().sum()

df.dtypes
```

Liste 2

```
## 3 sınıflı model için kullanılacak dataların seçilmesi
X=df.drop(["NSP","CLASS"],axis=1)

y=df["NSP"]

X.head()

nsp_classes = y.unique()
nsp_classes

from keras import utils as np_utils
from sklearn.preprocessing import LabelEncoder
# Encode class values as integers and perform one-hot-encoding
encoder = LabelEncoder()
encoder.fit(y)
y = encoder.transform(y)
y = np_utils.to_categorical(y)
print(y)

y.shape
```

Liste 3

```
#%% Veri Standartlaştırma
from sklearn.preprocessing import StandardScaler
Scaler=StandardScaler()
X=Scaler.fit_transform(X)

X[0:1]

X.shape
```

```
#%% KERAS YSA


from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier
from keras.models import Sequential
from keras.layers import Dense




# Train-Test
from sklearn.model_selection import train_test_split
# shuffle and split training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=0)
```

Liste 4

Multi Layer Perceptron için Keras Python kütüphanesi, modellerin bir katman dizisi olarak oluşturulmasına odaklanır. Keras ile model için her bir katman tek tek oluşturulabilir ve her bir katman özelinde ayarlamalar yapılmasına olanak sağladığı için daha esnek bir yapı ortaya koymaktadır. Model katmanları tek tek oluşturulduğu için her bir katmandaki nöron sayısı Dense fonksiyonu ile belirtilmelidir. Giriş katmanı için veri setindeki bağımsız değişken sayısı(girdi boyutu) verilmelidir. Data setimizde 23 tane bağımsız değişkenimiz olduğu için input_dim=23 (Liste5) olarak belirtildi. Daha sonra her bir katman için aktivasyon fonksiyonu da her bir katman özelinde

ayarlanabildiği için bunun da belirtilmesi gerekmektedir. Keras için çıktı katmanın nöron sayısının da(sınıf sayısının) belirtilmesi gerekmektedir. Ayrıca Keras modelinde sınıf sayısı 2'den fazla ise çıktı katmanının aktivasyon fonksiyonu "softmax" olarak belirtilmelidir. MLPClassifier'da ise bunları belirtmemize gerek yok. Keras modelini compile(derleme) ederken loss(kayıp) fonksiyonunu belirtmemiz gerekmektedir. MLPClassifier için loss fonksiyonunu belirtmeye gerek yoktur kendisi otomatik olarak seçer.

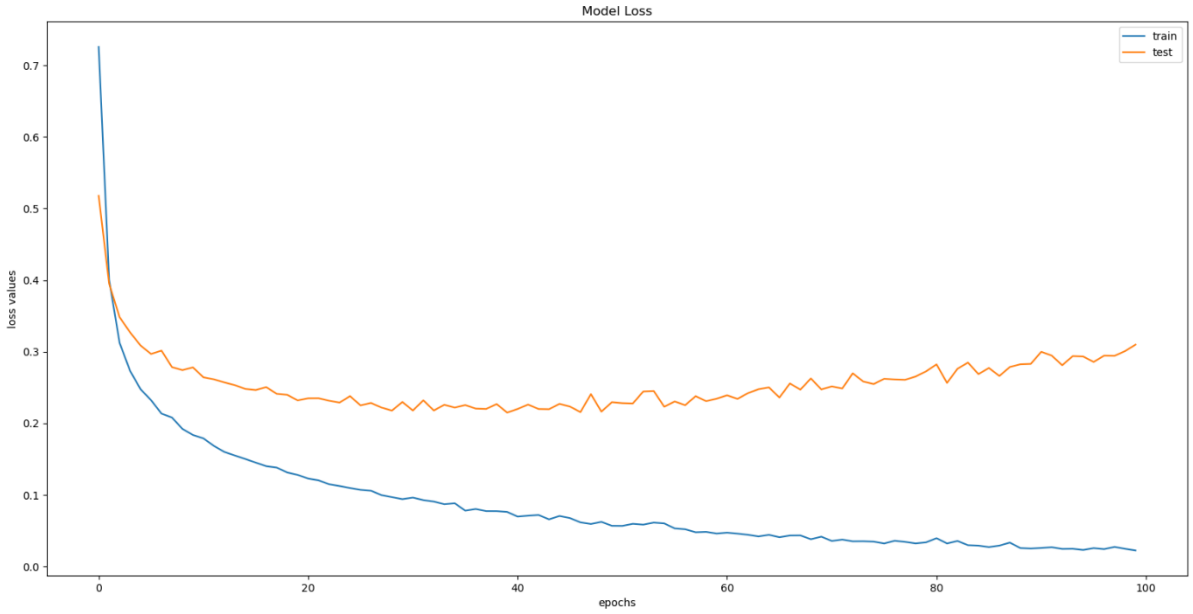
```
#İlk olarak modeli kötü bir şekilde kuralım ve train kaybı ve test kaybını gözlemleyelim.
class_weight = {0: 1, 1: 5.74, 2: 9.4}



def create_model(optimizer="adam"):
    # create model
    model = Sequential()
    model.add(Dense(64, input_dim=23, activation='relu'))

    model.add(Dense(32, activation='sigmoid'))
    model.add(Dense(3, activation='relu'))


    model.add(Dense(3, activation='softmax')) # 3 output olduğu için output layer 3 olmalı
    # multi-class olduğu için activation fonksiyonu 'softmax' seçilmelidir
    # multi class olduğu için loss fonksiyonu "categorical_crossentropy"
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer=optimizer,metrics=["accuracy"])
    return model
model = create_model() # tune edilmemiş model nesnesini oluşturma

egitim=model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=1,validation_data=(X_test,y_test))
```

Liste 5

Liste 5'te: Bir sinir ağı modelindeki parametre değerlerini optimize etmek için bir kayıp fonksiyonu kullanılır. Kayıp fonksiyonları, ağ için bir dizi parametre değerini, bu parametrenin ağın yapmayı amaçladığı görevi ne kadar iyi başardığını gösteren bir skaler değere eşler. Tipik olarak sinir ağları ile hatayı en aza indirmeye çalışırız. Bu nedenle, objektif fonksiyon genellikle bir maliyet fonksiyonu veya bir kayıp fonksiyonu olarak adlandırılır ve kayıp fonksiyonu tarafından hesaplanan değer basitçe " kayıp " olarak adlandırılır. Kayıp fonksiyonu seçimi doğrudan sinir ağınızın çıkış katmanında kullanılan aktivasyon fonksiyonu ile ilgilidir bu yüzden önemlidir.

Kerasta, 2 sınıflı sınıflandırma problemleri için hedef değişkeni {-1,1} şeklinde ise 'hinge' veya 'squared_hinge' kayıp fonksiyonları, hedef değişkeni {0,1} şeklinde binary formda ise 'binary_crossentropy' kayıp fonksiyonu seçilir. Hedef değişkeni {0,1,2..,n} şeklinde her sınıfa benzersiz bir tamsayı değerine sahip olduğu çok sınıflı sınıflandırma problemlerinde 'categorical_crossentropy' kayıp fonksiyonu kullanılır. Çok fazla sayıda etiketle sınıflandırma sorunları olan 'categorical_crossentropy' kullanıldığında olası bir hayal kırıklığı yapabilir bunun nedeni, one hot encoding işlemidir. Örneğin, bir kelime haznesindeki kelimeleri tahmin etmek, her bir etiket için bir tane olmak üzere on veya yüz binlerce kategoriye sahip olabilir. Bu, her eğitim örneğinin hedef elemanının, önemli bellek gerektiren on veya yüz binlerce sıfır değere sahip one hot encoding gerektirebileceği anlamına gelebilir.

loss: 0.0226 - accuracy: 0.9946 - val_loss: 0.3099 - val_accuracy:



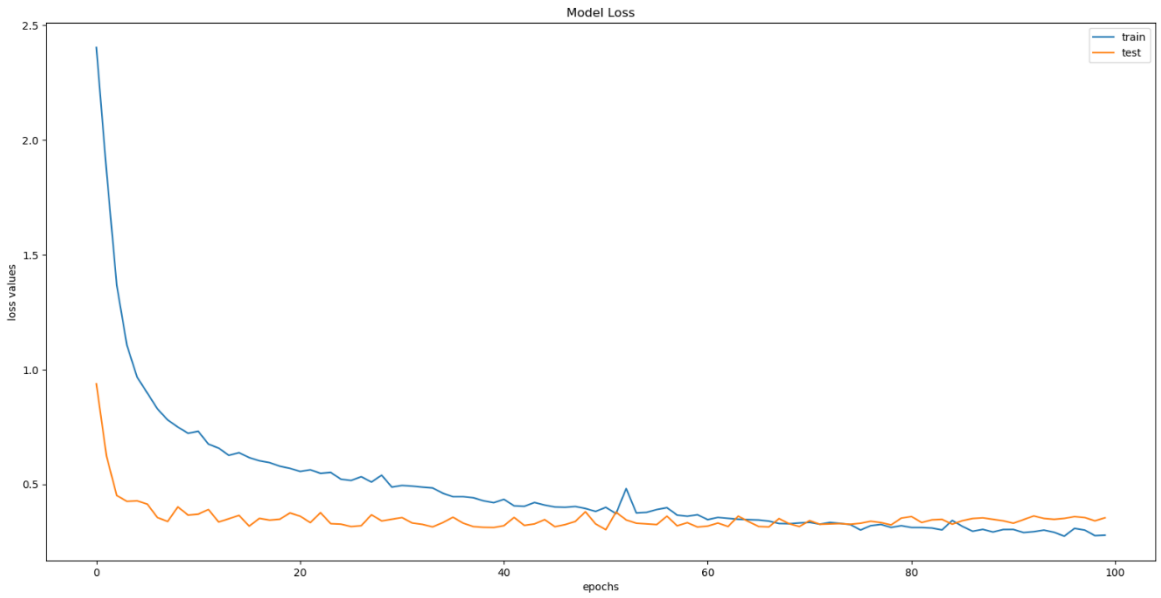
Liste 6

Liste 6'daki grafikten de anlaşılacağı üzere oldukça kötü bir model. Modelin test hatası artarken train hatası düşmektedir. Bu da modelin overfittinge doğru gittiğini yani modelin train seti üzerinde yüksek doğruluğa sahip olup test seti üzerinde daha düşük doğruluğa sahip olduğunu göstermektedir. Model train setini çok başarılı bir şekilde tahmin ederken, test setini o başarıda tahmin edememiştir. Şimdi modelimizi biraz geliştirelim overfittingin önüne geçmek için modelde düzenleme yapmamız gerekiyor. Modele fazladan bir katman eklendi ve sınıf sayılarındaki dengesizlik için class_weight kullanıldı ve katmanların nöron sayılarıda düzenlendi.

```
class_weight = {0: 1, 1: 5.74, 2: 9.4}


def create_model(optimizer="adam"):
    # create model
    model = Sequential()
    model.add(Dense(20, input_dim=23, activation='relu'))

    model.add(Dense(40, activation='sigmoid'))
    model.add(Dense(60, activation='relu'))


    model.add(Dense(3, activation='softmax')) # 3 output olduğu için output layer 3 olmalı
    # multi-class olduğu için activation fonksiyonu 'softmax' seçilmelidir
    # multi class olduğu için loss fonksiyonu "categorical_crossentropy"
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer=optimizer,metrics=["accuracy"])
    return model
model = create_model() # tune edilmemiş model nesnesini oluşturma

egitim=model.fit(X_train, y_train, epochs=100, batch_size=32,class_weight=class_weight, verbose=1,validation_data=(X_test,y_test))
```

Liste 7

Liste 7' deki kodlarl sınıf dağılımdaki dengesiz dağılım giderilmiş ve aşağıdaki tablo (Liste 8) elde edilmiştir.



Liste 8

Liste 8'de görüldüğü gibi modelin test ve train kayıpları birbirine yaklaşmıştır. Modelin train setindeki tahmin başarısı ve test setindeki tahmin başarısı da birbirine yakın değerler göstermektedir.

```
# %% plot loss during training
import matplotlib.pyplot as plt
plt.plot(egitim.history['loss'], label='train')
plt.plot(egitim.history['val_loss'], label='test')
plt.title('Model Loss')
plt.xlabel('epochs')
plt.ylabel('loss values')
plt.legend(loc='upper right')
plt.show()
```

```
# %% Modelin Tune Edilmemiş Skorları
import sklearn.metrics as metrics
y_pred=model.predict_classes(X_test)
```

```
# %%Accuracy

print("Accuracy:",metrics.accuracy_score(np.argmax(y_test, axis=1),y_pred))
```

```
# %%f1 score

print("f1_weighted:",metrics.f1_score(np.argmax(y_test, axis=1), y_pred,average='weighted'))
```

```
#%% Grid Search Cross Validation
# GridSearch Cross Validation Parametreleri
param_grid = {

    'epochs': [50,100,150],
    'batch_size':[32,50,100],
    'optimizer':['RMSprop', 'Adam','SGD'],

}
```

Liste 9

Liste 9'da model için hiçbir parametre girmeden yani model tune(ayar) edilmeden "accuracy" ve "f1_weighted" skorları hesaplandı.

Aşağıda denenmesi istenilen parametreler sözlük yapısı şeklinde gösterilmiştir. Ne kadar çok parametre gönderilirse o kadar işlem hacmi büyümektedir. Çünkü gönderilen her bir parametre bir birleri ile tek tek denenerek modeller oluşturulmaktadır. (Liste10)

```
param_grid = {

    'epochs': [50,100,150],
    'batch_size':[32,50,100],
    'optimizer':['RMSprop', 'Adam','SGD'],
```

Liste 10

epochs: Bir Epoch, tüm veri kümesinin sinir ağı üzerinden sadece bir kez ileri ve geri aktarılmasıdır. Başlangıçta, tüm veri kümesinin bir sinir ağı üzerinden geçirilmesi yeterli değildir. Ve tam veri kümesini aynı sinir ağına birçok kez geçirmemiz gerekiyor. Ancak, sınırlı bir veri kümesi kullandığımızı ve yinelemeli bir süreç olan Gradient Descent kullandığımız öğrenmeyi ve grafiği optimize etmek için kullandığımızı unutulmamalıdır. Bu nedenle, ağırlıkların tek geçiş veya bir dönemle güncellenmesi yeterli değildir.

batch_size: Küme büyüklüğü (batch_size) bir seferde yapay sinir ağını eğitmek için kullanılacak örnek sayısını belirtir.

optimizer : Optimize edici, { 'RMSprop', 'Adam','SGD', Adadelta, Adagrad, Adamax, Nadam, Ftrl}, Optimizerler ağırlık optimizasyonu içindir.

activation : { sigmoid, relu, softmax, softplus,tanh, softsign,selu,elu, exponential ve keras.layers.advanced_activations ile PreLU, LeakyReLU}, Gizli katman için aktivasyon fonksiyonunu ifade eder.

Yukarıda Grid Search yapısına gönderilen, Keras modelinin bazı parametreleri verilmiştir. Aktivasyon fonksiyonlarından en önemlileri relu genellikle gizli katmanlarda yani çıkış değil ara katmanlarda kullanılır. Sigmoid ise genelde sınıflandırıcılarda iyi çalışmakta ve iki çıkışlı sınıflandırma problemi için genelde sigmoid tercih edilir. Softmax ise genelde çok sınıflı problemlerde çıkış katmanı olarak kullanılır.

En iyi parametreler: 'batch_size': 50, 'epochs': 150, 'optimizer': 'Adam'

K-fold Cross Validation Accuracy Sonuçları: [0.88372093 0.90588235 0.87058824 0.89411765 0.81176471]

K-fold Cross Validation Accuracy Sonuçlarının Ortalaması: 0.8732147742818057

K-fold Cross Validation f1_weighted Sonuçları: [0.88235709 0.94141427 0.8982699 0.87373411 0.82106618]

K-fold Cross Validation f1_weighted Sonuçlarının Ortalaması: 0.8833683094688147

	f1_weighted	accuracy
Tune Edilmemiş(İlkel Model)	0.8898234437932919	0.8849765258215962
K-fold Cross Validation Ortalaması	0.8833683094688147	0.8732147742818057
Tune Edilmiş(En İyi Modelle)	0.9225352112676056	0.9230346636719856

Skorlama için kullanılan metrik olarak f1_weighted ve accuracy kullanılmıştır. Test seti üzerinden 5 farklı f1_weighted ve accuracy skoru elde edilmiştir. Daha sonra bu skorların ortalaması alınarak modelin valide edilmiş(doğrulanmış) skorlar elde edilmiş oldu.

	precision	**recall**	**f1-score**	**support**
0	0.96	0.96	0.96	326
1	0.72	0.81	0.76	58
2	0.94	0.76	0.84	42
accuracy			0.92	426
macro avg	0.87	0.84	0.85	426
weighted avg	0.93	0.92	0.92	426

	Class 0 Predicted	**Class 1 Predicted**	**Class 2 Predicted**	**Toplam Support**
Class 0 Actual	314	10	2	326
Class 1 Actual	11	47	0	58
Class 2 Actual	2	8	32	42

Class 0: Normal **Class 1**: Suspect **Class 3**:Pathological

314 hasta test datasında gerçekte Normal, tahmin sonucunda da Normal (Doğru Sınıflandırma)

47 hasta test datasında gerçekte Suspect, tahmin sonucunda da Suspect (Doğru Sınıflandırma)

32 hasta test datasında gerçekte Pathological, tahmin sonucunda da Pathological (Doğru Sınıflandırma)

10 hasta test datasında gerçekte Normal, tahmin sonucunda da Suspect (Yanlış Sınıflandırma)

2 hasta test datasında gerçekte Normal, tahmin sonucunda da Pathological (Yanlış Sınıflandırma)

11 hasta test datasında gerçekte Suspect, tahmin sonucunda da Normal (Yanlış Sınıflandırma)

2 hasta test datasında gerçekte Pathological, tahmin sonucunda da Normal (Yanlış Sınıflandırma)

8 hasta test datasında gerçekte Pathological, tahmin sonucunda da Suspect (Yanlış Sınıflandırma)

Accuracy=393/426=0.9225≅ 0.92 yani modelimiz %82 accuracy değerine sahip.

Sınıf 0 için:

Precision= 314/(314+11+2)=0.9602≅0.96

Recall=314/(314+10+2)=0.9631 ≅0.96

f1=2*(0.96*0.96)/(0.96+0.96)=0.96

Sınıf 1 için:

Precision= 47/(47+10+8)=0.7230≅0.72

Recall=47/(47+11+0)=0.8103 ≅0.81

f1=2*(0.72*0.81)/(0.72+0.81)=0.7623≅0.76

Sınıf 2 için:

Precision= 32/(32+2+0)=0.9411≅0.94

Recall=32/(32+2+8)=0.7619 ≅0.76

f1=2*(0.76*0.94)/(0.76+0.94)=0.8404≅0.84

f1_weighted	=(0.96*326+0.76*58+0.84*42)/426=0.9209≅0.92
f1_macro	=(0.96+0.76+0.84)/3=0.8533≅0.85
precision_weighted	=(0.96*326+0.72*58+0.94*42)/ 426=0.9253≅0.93
precision_macro	=(0.96+0.72+0.94)/3=0.8733≅0.87
recall_weighted	=(0.96*326+0.81*58+0.76*42)/ 426=0.9198≅0.92
recall_macro	=(0.96+0.81+0.76)/3=0.8433≅0.84

Liste 11

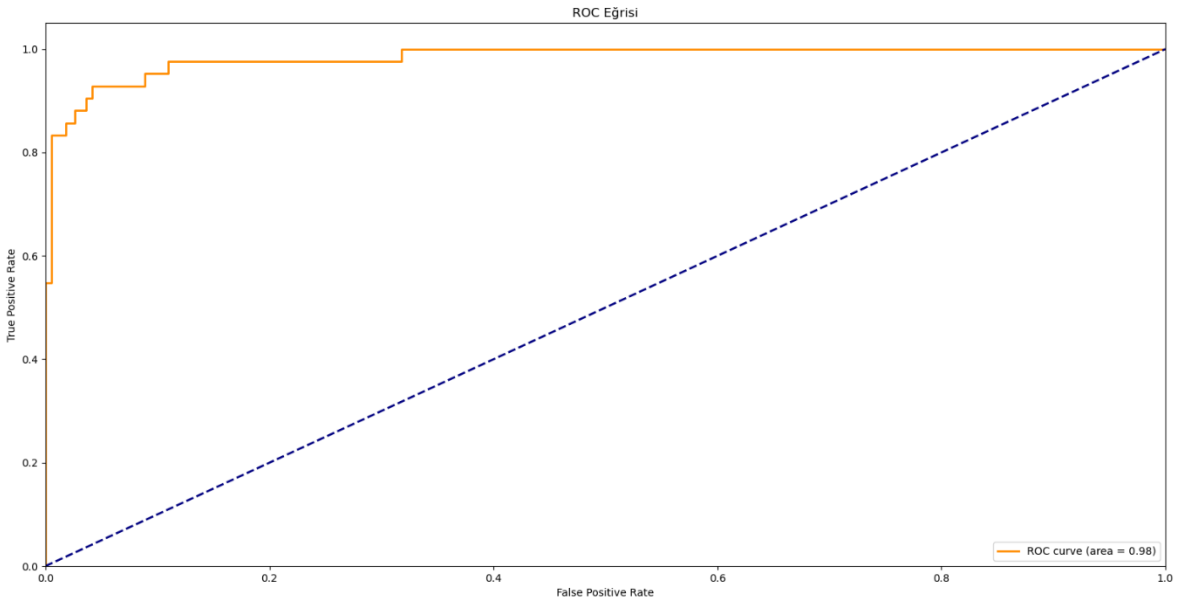
Liste 11'de tabloda görüldüğü gibi precision,recall,f1 skorları için macro ve weighted şeklinde hesaplamaları yapıldı. Bu skorların macro değerleri hesaplanırken tüm sınıflara göre değerleri toplanarak toplam sınıf sayısına bölünerek bulunur. Weighted değerleri ise her sınıfın skor değeri her bir sınıfın gerçek toplam sayıları ile çarpılarak yani support değerleri ile çarpılarak toplanır ve toplam popülasyon sayısına(toplam nüfus) bölünerek bulunur.

```
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["micro"]),
         color='deeppink', linestyle=':', linewidth=4)

plt.plot(fpr["macro"], tpr["macro"],
         label='macro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["macro"]),
         color='navy', linestyle=':', linewidth=4)

colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Eğrisinin Çok Sınıfa Genişletilmesi')
plt.legend(loc="lower right")
plt.show()
```

Liste 12

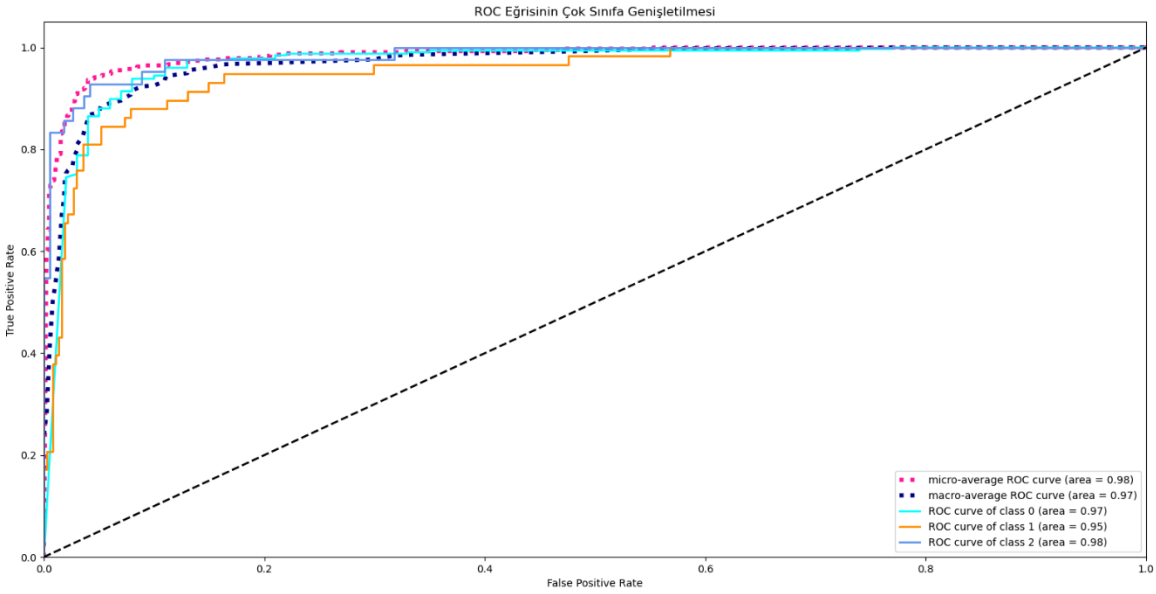
Liste 12'de ReceiverOperating Characteristic Curve (Alıcı çalışma karakteristik eğrisi) ve Area Under the Curve(Eğri Altındaki Alan) uygulaması yapılmıştır.

Modelimizde 0, 1 ve 2 adında üç sınıfınız var , 1 ve 2'ye karşı sınıflandırılmış 0 için bir ROC, 0 ve 3'e karşı sınıflandırılmış 2 için başka bir ROC ve 2 ve 0'a göre sınıflandırılmış 3 için üçüncü bir ROC eğrisine sahip olmaktayız. Model için ROC eğrileri ve AUC değerleri hesaplaması öncelikle gözlemlemek adına kendi seçtiğimiz herhangi bir sınıf özelinde yapıldı.



Liste 13

Liste 13'de görülen grafik modeli sınıf 2 için AUC değeri 0.98 gibi yüksek bir orana sahip. Bunun anlamı modelin sınıf ayrımı yaparken 2 sınıf için başarısı %98 oranında olduğu anlamına gelmektedir. Daha sonra tüm sınıfların ROC eğrileri ve her bir sınıfın AUC değerleri ve macro-micro ortalama ROC eğrileri ve AUC değerleri hesaplanarak çizdirilmiştir.



SONUÇ:

Model için elde edilen AUC değerleri; micro-average ROC eğrisi için 0.98, macro-average ROC eğrisi için 0.97, class 0 ROC eğrisi için(Normal) 0.97, class 1 ROC eğrisi için(Suspect) 0.95 ve class 2 ROC eğrisi için (Pathological) 0.98 şeklinde gözlemlenmiştir. Sonuçlardan da görüleceği gibi modelimiz sınıf ayrımını yapma doğruluğu konusunda class 1 için diğerlerine göre biraz daha düşük sonuç ortaya koymuştur. Bunu zaten confusion matrisinde class 1 için olan değerlerde tespit etmiştik. Genel olarak modelimiz sınıf ayrımını yüksek bir başarı ile yapabilmektedir. Bu oranlar ne kadar yüksek olursa modelin tahminleri yaparken hata yapma olasılığı o kadar düşük olmaktadır. ROC eğrileri bu konuda bize bu önemli bilgileri vermektedir.

MLP UYGULAMA 1

Multi Layer Perceptron (MLP) Classifier sklearn kütüphanesinde bulunan bir çok katmanlı algılayıcıdır. İlk olarak MLP ile model nesnesi oluşturuldu. Daha sonra model nesnesi train seti üzerinden fit edildi. Öncelikle valide edilmemiş(doğrulanmamış,ilkel) test skorlarını belirlemek istedim. Bunun için model tahmin işlemi yapılarak, “accuracy” ve “f1_weighted” skorlarını hesapladım. Burada f1 yerine f1_weigted kullanmamızın sebebi hedef değişkeninin(bağımlı değişkenin) multiclass olmasından dolayı f1 score hesaplarken average argümanına [None, 'micro', 'macro', 'weighted'] bunlardan bir tanesini kullanmak gerekmektedir aksi takdirde hata vermektedir. Bu bölümde model için hiçbir parametre girmeden yani model tune(ayar) edilmeden “accuracy” ve “f1_weighted” skorları hesaplandı. Bu skorlar nispeten tune edilmiş modele göre düşük olacaktır. Ayrıca model hedef(bağımlı) değişkeni multi-class(çok sınıflı) olduğu için model tahmin ve skorlama işlemlerinde bağımlı değişkene ait kısımlarda numpy methodu olan np.argmax ile eski haline getirilerek işlemler yapılmıştır. Model için herhangi bir katman bilgisi ya da diğer parametrelerden hiçbirisi girilmeden model oluşturuldu. Her defasında farklı sonuçlar vermemesi açısından tune edilmemiş model için de tune edilmiş model içinde random_state aynı ayarlandı. Bu bölümden sonra Grid Search yapısı ile MLP modelimizin parametre optimizasyonu için çeşitli parametreler modele denenecek ve en uygun parametreler ile model oluşturulacaktır. [(Liste 14), (Liste 15), (Liste 16), (Liste17)]

```
@author: Yusuf Ali
"""

#%%

from warnings import filterwarnings
filterwarnings('ignore')
import pandas as pd
import numpy as np

df = pd.read_csv("CTG.csv")


df.head()

# Drop unnecessaries
df=df.drop(["FileName","Date","SegFile","b","e","A", "B","C", "D" ,"E", "AD", "DE" ,"LD", "FS", "SUSP"],axis=1)

df.head()

# Coloumns names
df.columns

df.shape

df.isnull().sum()

# tüm  nan verileri silme işlemi
df = df.dropna()

df.isnull().sum()

df.dtypes
```

Liste 14

```
## 3 sınıflı model için kullanılacak dataların seçilmesi
X=df.drop(["NSP","CLASS"],axis=1)

y=df["NSP"]

X.head()

nsp_classes = y.unique()
nsp_classes

from keras import utils as np_utils
from sklearn.preprocessing import LabelEncoder
# Encode class values as integers and perform one-hot-encoding
encoder = LabelEncoder()
encoder.fit(y)
y = encoder.transform(y)
y = np_utils.to_categorical(y)
print(y)

y.shape
```

Liste 15

```
#%% Veri Standartlaştırma
from sklearn.preprocessing import StandardScaler
Scaler=StandardScaler()
X=Scaler.fit_transform(X)

X[0:3]

X.shape
```

```
#%% MLP YSA

from sklearn.model_selection import train_test_split
# shuffle and split training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=0)

from sklearn.neural_network import MLPClassifier # YSA kütüphanesi importu
# İlkel(Validation yapılmamış) Modeli Kurma
mlpc = MLPClassifier(random_state = 0) # YSA model nesnesi oluşturuldu

mlpc.fit(X_train, y_train) # YSA model nesnesi fit edildi
```

```
#%% Valide Edilmemiş Model Üzerinden Tahmin İşlemi
y_pred = mlpc.predict(X_test) # model tahmin işlemi test seti üzerinden

import sklearn.metrics as metrics
```

Liste 16

```
# %%Accuracy

print("Accuracy:",metrics.accuracy_score(y_test,y_pred))

# %%f1 score

print("f1_weighted:",metrics.f1_score(y_test, y_pred,average='weighted'))

#%% Grid Search Cross Validation

# Cross Validation İşlemi
# CV için parametreler sözlük yapısı şeklinde oluşturuldu
# GİRİLEN PARAMETRELER HAKKINDA BİLGİLER
# alpha : float, default=0.0001 L2 penalty (regularization term) parameter. (ceza parametresi)


mlpc_params = {"alpha": [0.1, 0.01, 0.0001],
               "hidden_layer_sizes": [(10,10,10),
                                      (100,100,100),
                                      (100,100)],
               "solver" : ["lbfgs","adam","sgd"],
               "activation": ["relu","logistic"]}

from sklearn.model_selection import GridSearchCV



mlpc = MLPClassifier(random_state = 0) # Model nesnesi oluşturuldu
# Model CV etme
mlpc_cv_model = GridSearchCV(mlpc, mlpc_params,
                             cv = 5, # 5 Katlı CV yapılması için
                             n_jobs = -1, # işlemciyi tam performansta kullanıma olanak sağlar
                             verbose = 2) # işlemciyi tam performansta kullanıma olanak sağlar

mlpc_cv_model.fit(X_train, y_train) # Model fit etme işlemi
# fit ederken scale edilmiş train seti üzerinden fit ettik!!
```

Liste 17

Grid search ile bir sözlük yapısı vasıtası ile modele optimizasyon için gönderilen parametreler:

alpha : float, default=0.0001 L2 penalty (regularization term) parameter. (ceza parametresi)

Alpha, ağırlıkların boyutunu kısıtlayarak overfitting(aşırı öğrenme) ile mücadele eden normalleştirme terimi, yani ceza terimi için bir parametredir. Artan alfa, daha küçük ağırlıkları teşvik ederek yüksek varyans (overfitting işareti) düzeltebilir ve bu da daha az eğrilikle ortaya çıkan bir karar sınır grafiğine yol açabilir. Benzer şekilde, azalan alfa, daha büyük ağırlıkları teşvik ederek yüksek bias(yanlılık) (underfitting belirtisi) düzeltebilir ve bu da daha karmaşık bir karar sınırına yol açabilir.

hidden_layer_sizes : tuple, length = n_layers - 2, default=(100,) gizli katmandaki nöron sayısını temsil eder. Burada gizli katman ekleme için içerisine nöron sayısı girerek ifade ediyoruz. Örneğin; "hidden_layer_sizes": (100,50,10) 3 katmanlı 1. katman 100, 2. katman 50, 3. katman 10 hücreden. Kullanılacak gizli katman sayısı, katmanlardaki nöron sayısı probleme göre yani datasete göre değişiklik göstermektedir.

solver : Çözücü, {'lbfgs', 'sgd', 'adam'}, default='adam'. Çözücüler ağırlık optimizasyonu içindir.

'lbfgs': yarı Newton metotları ailesinde bir optimize edicidir. Küçük veri setleri için kullanılır.

'sgd' :stokastik gradyan iniş anlamına gelir.

sdg: Bazı özel parametreleri doğru şekilde ayarlarsanız, çoğu problemin üstesinden gelir. LBFGS, küçük veri setleri için, büyük olanlar için Adam çalışır ve parametrelerini doğru ayarlarsanız SDG çoğu problemi çözebilir. SDG'nin parametreleri öğrenme hızını yansıtabilen learning_rate ve sinir ağının daha az yararlı çözümlerden kaçınmasına yardımcı olan bir değer olan momentum (veya Nesterov'un momentumu). Öğrenme oranını belirlerken, başlangıç değerini (genellikle 0.001 civarında, ancak daha az olabilir) ve eğitim sırasında hızın nasıl değiştiğini (learning_rate_init: 'constant','invscaling', veya 'adaptive' olarak tanımlamanız gerekir.)

'adam': Kingma, Diederik ve Jimmy Ba tarafından önerilen stokastik gradyan tabanlı bir optimizer anlamına gelir. ‘adam' , çok komplex ve büyük verisetleri için iyi bir optimize edicidir.

activation : {'identity', 'logistic', 'tanh', 'relu'}, default='relu'. Gizli katman için aktivasyon fonksiyonunu ifade eder.

- 'identity', f (x) = x değerini döndürür

- 'logistic', 'logistic sigmoid fonksiyonudur f (x) = 1 / (1 + exp (-x)) değerini döndürür.

- 'tanh', hiperbolik tan fonksiyonu, f (x) = tanh (x) değerini döndürür.

- 'relu', düzeltilmiş doğrusal birim fonksiyonu, f(x) = max(0, x) değerini döndürür.

```
#%% Model Tuning
# Final Modelinin en iyi parametre ile kurulması

mlpc_tuned = mlpc_cv_model.best_estimator_
# Final Modelinin fit edilmesi
mlpc_tuned.fit(X_train, y_train)
```

En iyi parametreler: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (100, 100), 'solver': 'adam'}

```
#%% K-fold f1_weighted

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

# K fold
kf = KFold(shuffle=True, n_splits=5) # 5 katlı cv

cv_results_kfold = cross_val_score(mlpc_tuned, X_test, np.argmax(y_test, axis=1), cv=kf, scoring= 'f1_weighted')

print("K-fold Cross Validation f1_weigted Sonuçları: ",cv_results_kfold)
print("K-fold Cross Validation f1_weigted Sonuçlarının Ortalaması: ",cv_results_kfold.mean())
```

```
#%% K-fold accuracy

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

# K fold
kf = KFold(shuffle=True, n_splits=5) # 5 katlı cv

cv_results_kfold = cross_val_score(mlpc_tuned, X_test, np.argmax(y_test, axis=1), cv=kf, scoring= 'accuracy')

print("K-fold Cross Validation accuracy Sonuçları: ",cv_results_kfold)
print("K-fold Cross Validation accuracy Sonuçlarının Ortalaması: ",cv_results_kfold.mean())
```

Liste 18

Liste 18 sonuçları:

K-fold Cross Validation f1_weigted Sonuçları: [0.86830451 0.88097292 0.90014789 0.89164953 0.9158371]

K-fold Cross Validation f1_weigted Sonuçlarının Ortalaması: 0.8913823894736403

K-fold Cross Validation accuracy Sonuçları: [0.88372093 0.85882353 0.88235294 0.83529412 0.90588235]

K-fold Cross Validation accuracy Sonuçlarının Ortalaması: 0.8732147742818057

	f1_weighted	accuracy
Tune Edilmemiş(İlkel Model)	0.915629126221463	0.9084507042253521
K-fold Cross Validation Ortalaması	0.8913823894736403	0.8732147742818057
Tune Edilmiş(En İyi Modelle)	0.9118176847496193	0.9131455399061033

Modele K-fold Cross Validation uygulanmış hali ile elde edilen skorlarda bize gösteriyor ki model veri seti farklı alt kümeye bölünmesi ile farklı alt kümelerde farklı sonuçlar vermektedir. K-fold'un bize daha güvenilir sonuçlar verdiği açık bir şekilde gözlemlenmiş oldu..

Sonraki aşamada tune edilmiş model için classification report ve confusion matrix hesaplamaları yapıldı. Classification report çeşitli skorlama türlerinin sınıflara göre değerlerini bir arada veren bir raporlama şeklidir. Aşağıda classification report ile alınan değer gösterilmiştir.

	precision	**recall**	**f1-score**	**support**
0	0.94	0.96	0.95	326
1	0.75	0.71	0.73	58
2	0.92	0.81	0.86	42
accuracy			0.91	426
macro avg	0.87	0.83	0.85	426
weighted avg	0.91	0.91	0.91	426

Confusion matrix ise sınıflandırma problemine ilişkin tahmin sonuçlarının bir özetidir. Doğru ve yanlış tahminlerin sayısı sayım değerleri ile özetlenir ve her sınıf tarafından ayrılır. Confusion matrixi sınıflandırma modelinizin tahminlerde ne zaman karıştırıldığını gösterir. Bize sadece bir sınıflandırıcı tarafından yapılan hatalar hakkında değil, daha da önemlisi yapılan hata türleri hakkında bilgi verir.

	Class 1 Predicted	*Class 2 Predicted*
Class 1 Actual	**TP**	**FN**
Class 2 Actual	**FP**	**TN**

Tablo 1

Tablo 1 içim:

Sınıf 1: Pozitif

Sınıf 2: Negatif

Pozitif (P): Gözlem pozitiftir (örneğin: sağlıklı).

Negatif (N): Gözlem olumlu değildir (örneğin: sağlıklı değil).

Gerçek Pozitif (TP): Gerçekte pozitif olan gözlem pozitif olarak tahmin edilmiştir.

Yanlış Negatif (FN): Gerçekte pozitif olan gözlem negatif olarak tahmin edilmiştir.

Gerçek Negatif (TN): Gerçekte negatif olan gözlem negatif olarak tahmin edilmiştir.

Yanlış Pozitif (FP): Gerçekte negatif olan gözlem pozitif olarak tahmin edilmiştir.

Accuracy = TP+TN/TP+FP+FN+TN **Precision** = TP/TP+FP **Recall** = TP/TP+FN

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

Bu şekilde hesaplama sonucunu aşağıda inceleyelim.

	Class 0 Predicted	**Class 1 Predicted**	**Class 2 Predicted**	**Toplam Support**
Class 0 Actual	314	10	2	326
Class 1 Actual	16	41	1	58
Class 2 Actual	4	4	34	42

Class 0: Normal **Class 1**: Suspect **Class 3**:Pathological

314 hasta test datasında gerçekte Normal, tahmin sonucunda da Normal (Doğru Sınıflandırma)

41 hasta test datasında gerçekte Suspect, tahmin sonucunda da Suspect (Doğru Sınıflandırma)

34 hasta test datasında gerçekte Pathological, tahmin sonucunda da Pathological (Doğru Sınıflandırma)

10 hasta test datasında gerçekte Normal, tahmin sonucunda da Suspect (Yanlış Sınıflandırma)

2 hasta test datasında gerçekte Normal, tahmin sonucunda da Pathological (Yanlış Sınıflandırma)

16 hasta test datasında gerçekte Suspect, tahmin sonucunda da Normal (Yanlış Sınıflandırma)

1 hasta test datasında gerçekte Suspect, tahmin sonucunda da Pathological (Yanlış Sınıflandırma)

4 hasta test datasında gerçekte Pathological, tahmin sonucunda da Normal (Yanlış Sınıflandırma)

4 hasta test datasında gerçekte Pathological, tahmin sonucunda da Suspect (Yanlış Sınıflandırma)

Modelin confusion matrixine göre classification reporttaki değerleri hesaplayalım:

Toplam Doğru Sınıflandırma Sayısı:(314+41+34)=389

Toplam Popülasyon Sayısı: (314+10+2+16+41+1+4+4+34)=426

Accuracy=389/426=0.9131≅ 0.91 yani modelimiz %91 accuracy değerine sahip.

Sınıf 0 için:

Precision= 314/(314+16+4)=0.9401≅0.94

Recall=314/(314+10+2)=0.9631 ≅0.96

f1=2*(0.96*0.94)/(0.96+0.94)=0.9498 ≅0.95

Sınıf 1 için:

Precision= 41/(41+10+4)=0.7454≅0.75

Recall=41/(41+16+1)=0.7068 ≅0.71

f1=2*(0.71*0.75)/(0.71+0.75)=0.7294≅0.73

Sınıf 2 için:

Precision= 34/(34+2+1)=0.9189≅0.92

Recall=34/(34+4+4)=0.8095 ≅0.81

f1=2*(0.81*0.92)/(0.81+0.92)=0.8615≅0.86

Son olarak precision,recall,f1 skorları için macro ve weighted şeklinde hesaplamalar yapılacaktır. Bu skorların macro değerleri hesaplanırken tüm sınıflara göre değerleri toplanarak toplam sınıf sayısına bölünerek bulunur. Weighted değerleri ise her sınıfın skor değeri her bir sınıfın gerçek toplam sayıları ile çarpılarak yani support değerleri ile çarpılarak toplanır ve toplam popülasyon sayısına(toplam nüfus) bölünerek bulunur.

f1_weighted=(0.95*326+0.73*58+0.86*42)/426=0.9111≅0.91

f1_macro=(0.95+0.73+0.86)/3=0.8466≅0.85

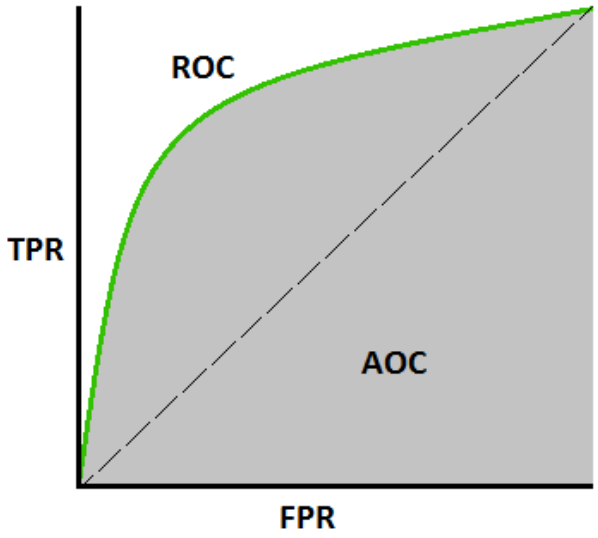
precision_weighted=(0.94*326+0.75*58+0.92*42)/ 426=0.9121≅0.91

precision_macro=(0.94+0.75+0.92)/3=0.87

recall_weighted=(0.96*326+0.71*58+0.81*42)/ 426=0.9111≅0.91

recall_macro=(0.96+0.71+0.81)/3=0.8266≅0.83

Son olarak model için ROC (Alıcı çalışma karakteristik eğrisi) ve AOC(Eğri Altındaki Alan) uygulaması yapılmıştır. Çok sınıflı sınıflandırma probleminin performansını kontrol etmemiz veya görselleştirmemiz gerektiğinde, AUC (Eğrinin Altındaki Alan) ROC (Alıcı Çalışma Özellikleri) eğrisini kullanıyoruz. Herhangi bir sınıflandırma modelinin performansını kontrol etmek için en önemli değerlendirme metriklerinden biridir.



ROC EĞRİSİ

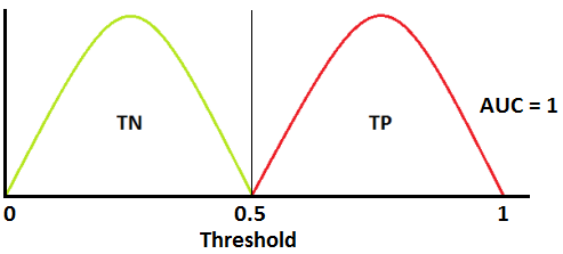
ROC Eğrisinde Kullanılan terimler:

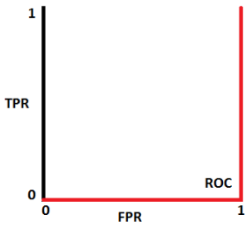
TPR (Gerçek Pozitif Hız) = Recall = Hassasiyet **TPR** =TP/(TP+FN)

Specificity(Özgünlük)=TN/(TN+FP), FPR=1-Specificity, **FPR**=FP/(TN+FP)

Mükemmel bir model, 1'e yakın AUC'ye sahiptir, bu da iyi bir ayrılabilirlik ölçüsüne sahip olduğu anlamına gelir. Zayıf bir model 0'a yakın AUC'ye sahiptir, bu da en kötü ayrılabilirlik ölçüsüne sahip olduğu anlamına gelir.

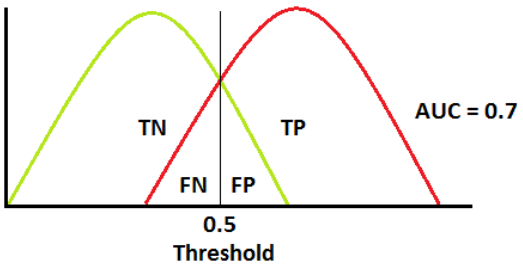
ROC eğrisini ve AUC skorunu 2 sınıflı bir örnek için ele alacak olursak:

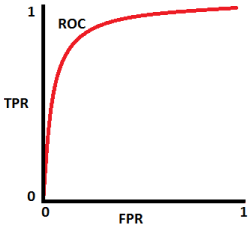




Not: Kırmızı dağılım eğrisi pozitif sınıftadır ve yeşil dağılım eğrisi negatif sınıflıdır.

Bu ideal bir durum. İki eğri hiçbir şekilde örtüşmediğinde, model ideal bir ayrılabilirlik ölçüsüne sahiptir. Pozitif sınıf ile negatif sınıf arasında mükemmel bir ayrım yapabilir.



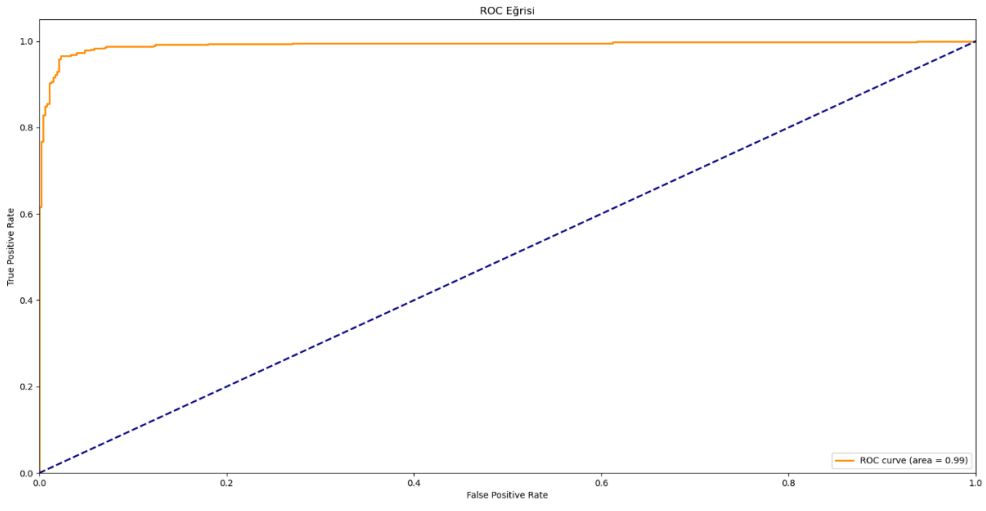


İki dağılım çakıştığında, tip 1 ve tip 2 hatasını ortaya koyarız. Eşiğe bağlı olarak, bunları en aza indirebilir veya en üst düzeye çıkarabiliriz. AUC 0.7 olduğunda, modelin pozitif sınıf ile negatif sınıf arasında ayrım yapabilme şansının% 70 olduğu anlamına gelir.

SONUÇ:

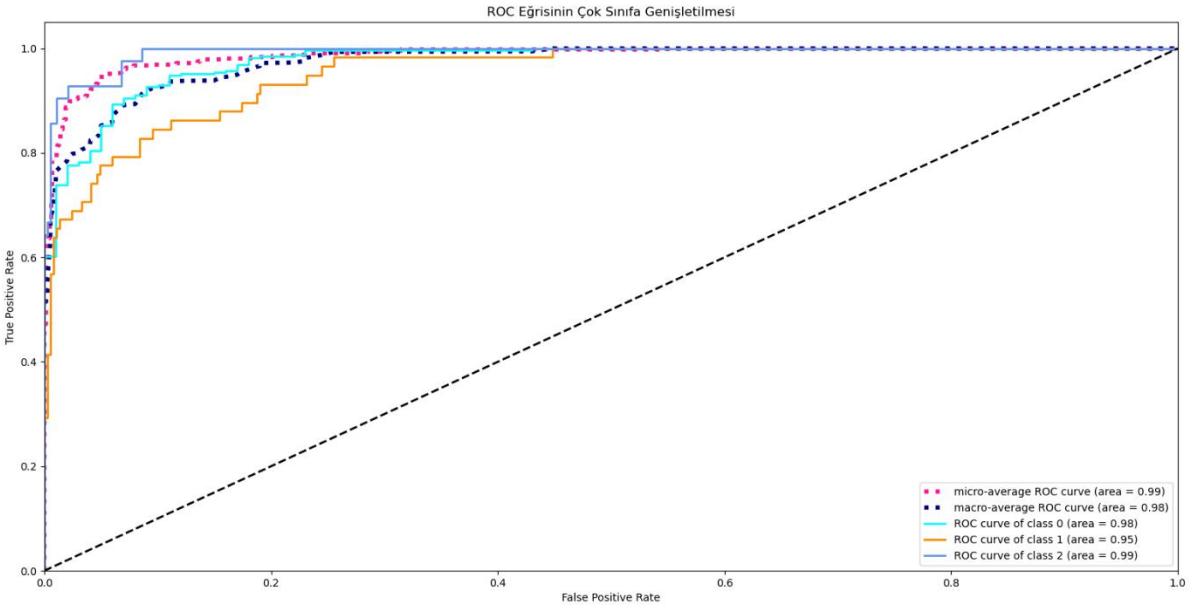
Modelimizde 0, 1 ve 2 adında üç sınıfınız var , 1 ve 2'ye karşı sınıflandırılmış 0 için bir ROC, 0 ve 3'e karşı sınıflandırılmış 2 için başka bir ROC ve 2 ve 0'a göre sınıflandırılmış 3 için üçüncü bir ROC eğrisine sahip olmaktayız. Model için ROC eğrileri ve AUC değerleri hesaplaması öncelikle gözlemlemek adına kendi seçtiğimiz herhangi bir sınıf özelinde yapıldı.

Aşağıdaki grafik sınıf 2'ye ait ROC eğrisini göstermektedir.



Sınıf 2

Model sınıf 2 için AUC değeri 0.99 gibi yüksek bir orana sahip. Bunun anlamı modelin sınıf ayrımı yaparken 2 sınıf için başarısı %99 oranında olduğu anlamına gelmektedir. Daha sonra tüm sınıfların ROC eğrileri ve her bir sınıfın AUC değerleri ve macro-micro ortalama ROC eğrileri ve AUC değerleri hesaplanarak çizdirilmiştir.



Model için elde edilen AUC değerleri;

micro-average ROC eğrisi için 0.99,

macro-average ROC eğrisi için 0.98,

class 0 ROC eğrisi için(Normal) 0.98,

class 1 ROC eğrisi için(Suspect) 0.95 ve

class 2 ROC eğrisi için (Pathological) 0.99 şeklinde değer elde edildi.

Breast_cancer_data.csv adlı veri setini Keras ve MLP yapay sinir ağı kütüphanelerini kullanarak performans değerlendirmelerinin yapılması amaçlanmıştır. [KERAS UYGULAMA 2 ve MLP UYGULAMA 2]

İstenilen 2 outputlu yani 0 ve 1 çıktılarını veren Keras ve MLP kütüphaneleri kullanılarak yapılan uygulamayı içermektedir. [KERAS UYGULAMA 2 ve MLP UYGULAMA 2]

KERAS UYGULAMA 2

Öncelikle **veriseti** isimli DataFrame nesnesi pandas kütüphanesi yardımıyla oluşturulmuştur.

Breast_cancer_data.csv isimli dosya 5 adet girdi özniteliği ve diagnosis isimli bir çıktı özniteliğine sahip 570 satırdan oluşmaktadır. Diagnosis özniteliği 0 ve 1 çıktılarını vermektedir. Bu data setinde meme kanseri teşhisi tahmini yapılacaktır.

```
@author: Yusuf Ali
"""
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from keras.models import Sequential
from keras.layers import Dense,Activation

veriseti=pd.read_csv("Breast_cancer_data.csv")
print(veriseti.info())
#veri setinde kayıp ya da sayısal olmayan verilerin olup olmadığının kontrolü için bu kod bloğu oluş
import re
kayip_veriler=[]
sayisal_olmayan_veriler=[]

for oznitelik in veriseti:
    essiz_deger=veriseti[oznitelik].unique()
    print("'{}' özniteliğine ait (unique) veriler : {}".format(oznitelik, essiz_deger.size))
    if (essiz_deger.size>10):
        print("10 adet essizdeger listele")
        print(essiz_deger[0:10])
        print("\n---------\n")

        if(True in pd.isnull(essiz_deger)):
            s="{} özniteliğe ait kayıp veriler {}".format(oznitelik, pd.isnull(veriseti[oznitelik]).s
            kayip_veriler.append(s)

            for i in range (1,np.prod(essiz_deger.shape)):
                if (re.match('nan',str(essiz_deger[i]))):
                    break
                if not (re.search('(^\d+\.?\d*$)|(^\d*\.?\d+$)', str(essiz_deger[i]))):
                    sayisal_olmayan_veriler.append(oznitelik)
                    break
print("Kayıp veriye sahip öznitelikler:\n{}\n\n".format(kayip_veriler))
print("Sayısal olmayan veriye sahip öznitelikler:\n{}".format(sayisal_olmayan_veriler))
```

Liste 19

Liste 19'de öncelikle veriseti adlı dataframe içerisinden bağımsız değişken (X) ve bağımlı değişken (y) öznitelikleri, Breast_cancer_data.csv dosyasındaki isimleri kullanılarak oluşturulmuştur.

Veri setinde kayıp ya da sayısal olmayan verilerin olup olmadığının kontrolü için Liste 19'deki kod bloğu oluşturulmuştur. Kod bloğu veri setini daha iyi tanımak için yararlı olacaktır. Bu kod bloğunda, her özniteliğin içerdiği eşsiz(unique) değerler görülebilmektedir. Eşsiz değerin çok olması durumu dikkate alınarak eğer 10'dan fazla eşsiz değer var ise ekrana yazdırılmaktadır. Kodda kullanıla **re** (regular expression) kütüphanesi sayısal olmayan verilerin araştırılmasında kullanılmıştır. Ekran çıktısı incelendiğinde kayıp veri olmadığı görülmüştür. Veri ön işleme yapılmıştır.

```
# %%
y=veriseti.iloc[:,5:6].values
x_data=veriseti.iloc[:,0:5].values

# %% öznitelik ölçeklendirme
x=(x_data-np.min(x_data))/(np.max(x_data)-np.min(x_data))

# %% veri setinin eğitim ve  test kümelerine ayrılması

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test=train_test_split(x,y,test_size=0.3,random_state=1)
```

Liste 20

Liste 20' de veri setinin eğitim ve test kümelerine ayrılması ve öznitelik ölçeklendirme için uygulanan kodlar görülmektedir. Veri setinin %70'i eğitim %30'u test için ayrılmıştır. Veri ön işleme adımlarından olan öznitelik ölçeklendirme işlemi veri setindeki tüm değerlerin 0 ile 1 arasında değer alan bir dönüşüme uğratmıştır.

```
# %%keras
siniflandirici=Sequential()

#ilk gizli katman
siniflandirici.add(Dense(output_dim=16, init='uniform',activation='relu', input_dim=5))

#ikinci gizli katman
siniflandirici.add(Dense(output_dim=32, init='uniform',activation='relu'))

#cikti katmanı
siniflandirici.add(Dense(output_dim=1, init='uniform',activation='sigmoid'))

#derleme işlemi
siniflandirici.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])


egitim=siniflandirici.fit(x_train,y_train,epochs=100,validation_data=(x_test,y_test))

import sklearn.metrics as metrics
y_pred=siniflandirici.predict_classes(x_test)
```

Liste 21

Liste 19'da Keras kütüphanesi import edilmiştir.

5 girdi ve 1 çıktıdan oluşan yapay sinir ağı modeli, iki gizli katman kullanılarak oluşturulmuştur. İlk gizli katmanda 16, ikinci 32 adet yapay nöron kullanılmıştır.

Liste 21'de görülen kodlar yapay sinir ağı modelini oluşturmaktadır.

add() fonksiyonu yapay sinir ağı katmaları oluşturulmasında kullanılır. Gizli katman sayısı bu fonksiyon ile arttırılabilir.

output_dim parametresi katmandaki yapay nöron sayısını ifade eder.

input_dim parametresi girdi özniteliklerinin sayısını belirtmeye yarar.

İnit parametresi ise başlangıç ağırlık değerlerinin belirlenmesi için alternatifler sunar.

compile() metodu öğrenme sürecini yapılandırmak için kullanılır.

optimizer parametresi kullanılan en iyileme algoritmasının belirlemesini sağlar.

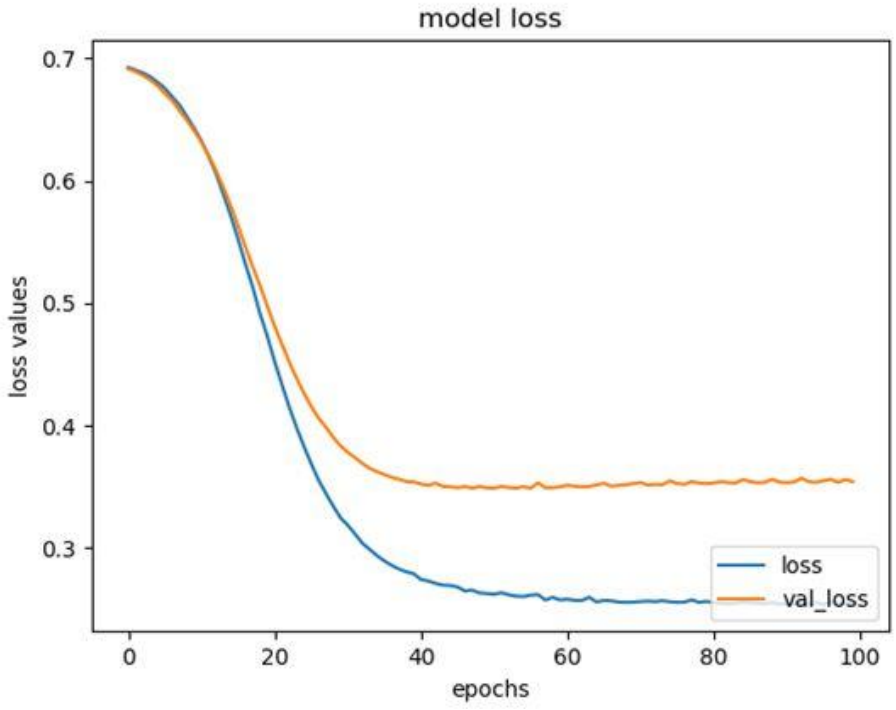
metrics parametresi model performansının değerlendirilmesinde kullanılan değerler alır.

loss parametresi hata fonksiyonlarının tanımlanmasında kullanılır.

```
plt.plot(egitim.history['loss'])
plt.plot(egitim.history['val_loss'])
plt.title('model loss')
plt.xlabel('epochs')
plt.ylabel('loss values')
plt.legend(['loss','val_loss'],loc='lower right')
plt.show()
```

Liste 22

Liste 22'de parametrelerin grafiği çizdirilmiştir.



Şekil a

```
from sklearn.metrics import confusion_matrix, classification_report
hm=confusion_matrix(y_test, y_pred)
print(classification_report(y_test,y_pred))

# %% roc ve auc

from sklearn.metrics import roc_curve, auc
ypo,dpo, esikDeger=roc_curve(y_test, y_pred)
aucDegeri =auc(ypo,dpo)

plt.figure()
plt.plot(ypo,dpo, label='AUC %0.2f' % aucDegeri)
plt.plot([0,1],[0,1],'k--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('yanlış pozitif oranı(YPO)')
plt.ylabel('dogru pozitif oranı(DPO)')
plt.title('roc egrisi')
plt.legend(loc="best")
plt.show()
```

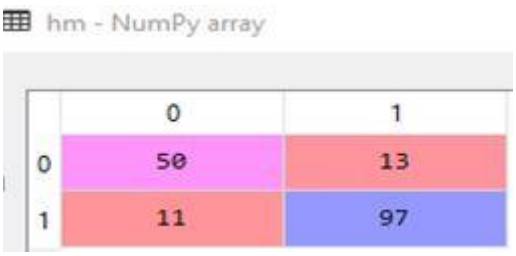
Liste 23

confusion_matrix() fonksiyonu ile oluşturulan hata matrisi hm adlı listeye atanmıştır. Elde edilen hata matrisine ait performans değerlendirme ölçütleri kesinlik, duyarlılık ve f1 skorlama değerlerini içeren sonuçlar listelenmiştir.

	precision	recall	F1_score	support
0	0.82	0.79	0.81	63
1	0.88	0.90	0.89	108
Micro avg	0.86	0.86	0.86	171
Macro avg	0.85	0.85	0.85	171
Weighted avg	0.86	0.85	0.86	171

Tablo a

Hata matrisi doğru pozitif, doğru negatif, yanlış pozitif ve yanlış negatif olmak üzere dört kısımdan meydana gelir.



Şekil b

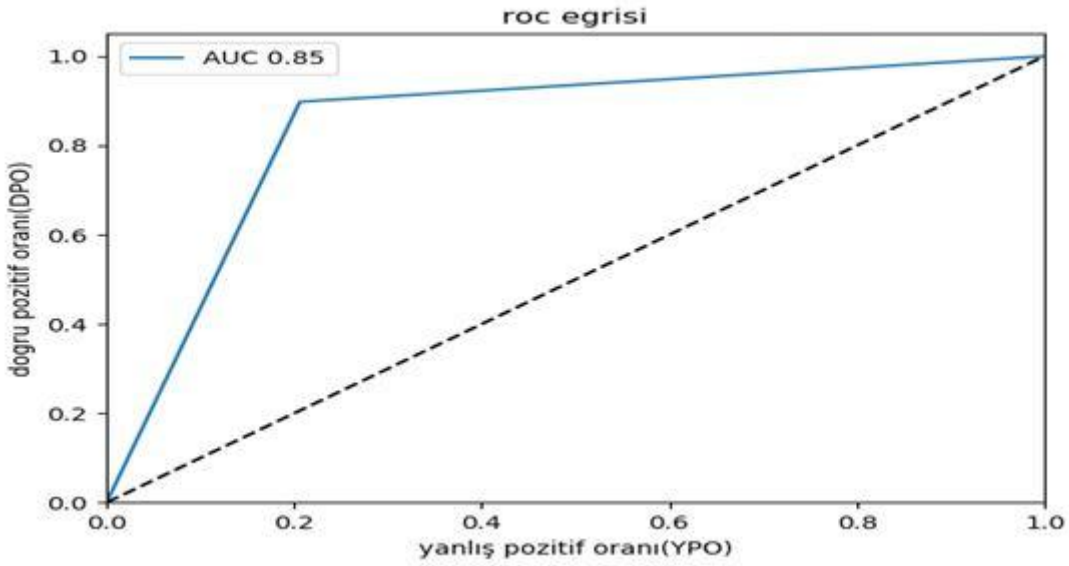
Doğru pozitif ve doğru negatif örneklerin sayısının çok olması sınıflandırma performansı için olumlu yönde etki yapacaktır.

Dengeli dağılıma sahip olmayan veri setleri için ROC eğrisi kullanılarak performans değerlendirmesi yapılabilir.

Doğru pozitif oranı ve yanlış pozitif oranı ROC eğrisini oluşturan iki temel unsurdur.

Doğru pozitif oranının yüksek, yanlış pozitif oranının düşük olması performans için olumlu olarak nitelendirilir. ROC eğrisinin altında kalan alanın 1'e yakın olması sınıflandırma performansı açısından hedeflenen bir sonuçtur.

AUC değeri 0 ile 1 arasında değişir istenen değeri ise 1'e olabildiğince yakın olmasıdır.



Şekil c

```
#%% ağırlıklar
for i in siniflandirici.layers:
    ilk_gizli_katman=siniflandirici.layers[0].get_weights()
    ikinci_gizli_katman=siniflandirici.layers[1].get_weights()
    cikti_katman=siniflandirici.layers[2].get_weights()
```

Liste 24

Liste 24'te üç adet ağırlık matrisi oluşmaktadır.

ilk_gizli_katman listesinde 5*6 boyutunda bir liste görülmektedir. Burada 5 değeri girdileri 6 değeri ise 1. Gizli katmandaki yapay nöron sayısını vermektedir.

6*1 boyutlu liste ise bias'a ağırlık değerlerini vermektedir. Ağırlık değerleri modeli oluşturan denklemin katsayılarını oluşturmaktadır.

MLP UYGULAMA 2

Öncelikle **veriseti** isimli DataFrame nesnesi pandas kütüphanesi yardımıyla oluşturulmuştur.

```
@author: Yusuf Ali
"""
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
veriseti=pd.read_csv("Breast_cancer_data.csv")
print(veriseti.info())

# %%
y=veriseti.diagnosis.values
x_data=veriseti.drop(["diagnosis"],axis=1)
```

Liste 25

Aşağıdaki görülen Liste 26' da veri setinin eğitim ve test kümelerine ayrılması ve öznitelik ölçeklendirme için uygulanan kodlar görülmektedir. Veri setinin %70'i eğitim %30'u test için ayrılmıştır. Veri ön işleme adımlarından olan öznitelik ölçeklendirme işlemi olan normalizasyon yapılmıştır.

```
# %% normalization
x=(x_data-np.min(x_data))/(np.max(x_data)-np.min(x_data))
# %%
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test=train_test_split(x,y,test_size=0.3,random_state=1)
```

Liste 26

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test=train_test_split(x,y,test_size=0.3,random_state=1)

#%%
from sklearn.neural_network import MLPClassifier

mlpc_model=MLPClassifier()
mlpc_model.fit(x_train,y_train)

# %%
print("score:",mlpc_model.score(x_test,y_test))

# %% model tuning

from sklearn.model_selection import GridSearchCV

mlpc_params={"alpha":[1,0.1,0.01,0.005],
             "hidden_layer_sizes":[(10,10),(100,100,100),(3,5)]}
# mlpc=MLPClassifier() #ilk çalışan
mlpc=MLPClassifier(activation="logistic",solver="lbfgs") #2. çalışan
mlpc_cv_model=GridSearchCV(mlpc,mlpc_params,cv=10,n_jobs=-1,verbose=2).fit(x_train,y_train)

# %%
print(mlpc_cv_model.best_params_)
# %%

mlpc_tuned=MLPClassifier(alpha=0.1,hidden_layer_sizes=(3,5),activation="logistic",solver="lbfgs").fit(x_train,y_train)
# %%
print("score:",mlpc_tuned.score(x_test,y_test))
#%%
import sklearn.metrics as metrics
y_pred=mlpc_model.predict(x_test)
```

Liste 27

Liste 27'deki kod bloğunda yapay sinir ağı modeli çok katmanlı perceptron ile oluşturulmuş. Grid search ile geliştirilmiştir. MLPClassifier, her adımda kayıp fonksiyonunun model parametreleri ile ilgili kısmi türevlerinin parametreleri güncelleştirmek için hesaplandığından yinelemeli olarak çalışır.

hidden_layer_sizes =parametresi gizli tabakadaki nöronların sayısını temsil eder.

alpha =normalleştirme terimi parametresidir.

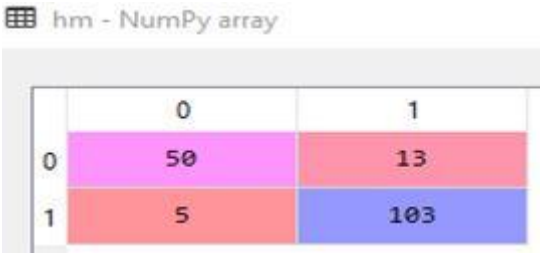
```
from sklearn.metrics import confusion_matrix, classification_report
hm=confusion_matrix(y_test, y_pred)
print(classification_report(y_test,y_pred))

#%%

from sklearn.metrics import roc_curve, auc
ypo,dpo, esikDeger=roc_curve(y_test, y_pred)
aucDegeri =auc(ypo,dpo)

plt.figure()
plt.plot(ypo,dpo, label='AUC %0.2f' % aucDegeri)
plt.plot([0,1],[0,1],'k--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('yanlış pozitif oranı(YPO)')
plt.ylabel('dogru pozitif oranı(DPO)')
plt.title('roc egrisi')
plt.legend(loc="best")
plt.show()
```

Liste 28

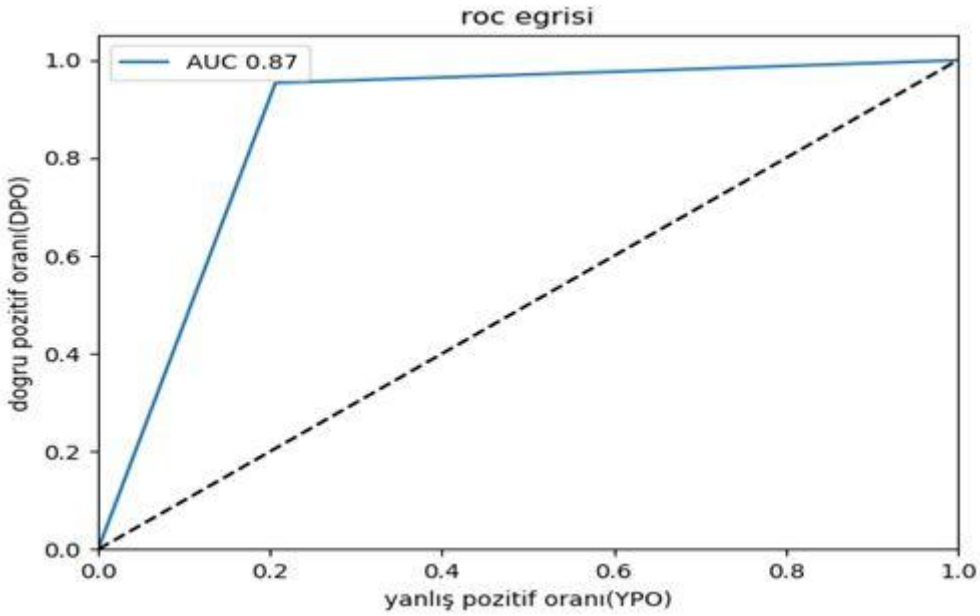
Liste 28'de: Keras uygulama 2'de uygulanan hata matrisi oluşturma ve roc eğrisi oluşturma kodları aynı şekilde yazılmıştır.

	precision	recall	F1_score	support
0	0.91	0.79	0.85	63
1	0.89	0.95	0.92	108
Micro avg	0.89	0.89	0.89	171
Macro avg	0.90	0.87	0.88	171
Weighted avg	0.90	0.89	0.89	171

Elde edilen hata matrisine ait performans değerlendirme ölçütleri kesinlik, duyarlılık ve f1 skorlama değerlerini içeren sonuçlar listelenmiştir.



Hata matrisi doğru pozitif, doğru negatif, yanlış pozitif ve yanlış negatif olmak üzere dört kısımdan meydana gelir.



Sonuç:

Keras ile elde edilen Auc 0.85 değeri ile MLP ile elde edilen AUC 0.87 değerleribirbirine oldukça yakındır.

Mlp'de uygulanan Grid Search Cross Validation'ın modeli geliştirdiği düşünülmektedir.