T.C.

KARABÜK ÜNİVERSİTESİ

LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

BİYOMEDİKAL MÜHENDİSLİĞİ YÜKSEK LİSANS PROĞRAMI

BİYOMEDİKAL MÜHENDİSLİĞİNDE YAPAY SİNİR AĞI UYGULAMALARI

PYTHON İLE MLP VE KERAS KÜTÜPHANESİYLE DERİN ÖĞRENME UYGULAMASI RAPORU

TALEH BİNNATOV
2028142023

DR. ÖĞR. ÜYESİ HAKAN YILMAZ

Kalp Hastalığı Tahmini

Veri seti seçimi

Dünya Sağlık Örgütü, Kalp hastalıkları nedeniyle her yıl dünya çapında 12 milyon ölüm meydana geldiğini tahmin etmektedir. Amerika Birleşik Devletleri ve diğer gelişmiş ülkelerdeki ölümlerin yarısı kardiyovasküler hastalıklardan kaynaklanıyor. Kardiyovasküler hastalıkların erken prognozu, yüksek riskli hastalarda yaşam tarzı değişiklikleri konusunda karar vermede yardımcı olabilir ve dolayısıyla komplikasyonları azaltabilir [1].

Bu araştırma, çeşitli özelliklere göre kalp hastalığının en ilgili / risk faktörlerini saptamayı ve genel riski tahmin etmeyi amaçlamaktadır. Veri seti Kaggle web sitesinde halka açık olarak mevcuttur ve Massachusetts, Framingham kasabasında yaşayanlar üzerinde devam eden bir kardiyovasküler çalışmadan alınmıştır. Sınıflandırma amacı, hastanın 10 yıllık gelecekteki koroner kalp hastalığı (KKH) riskine sahip olup olmadığını tahmin etmektir.

Veri seti 4.000'den fazla kayıt ve 15 özellik içeren hasta bilgilerini içermektedir. Her özellik, potansiyel bir risk faktörüdür. Veri setinde hem demografik hem davranışsal hem de tıbbi risk faktörleri vardır [2].

Eğitim için veri setinin 1271 satırı (yarısı 0, yarısı 1 sınıfı olacak şekilde) ve 8 özelliği (tıbbi faktörler) kullanılmıştır. Kullanılan sütunlar aşağıdaki gösterilmiştir.

Y	X			
TenYearCHD	currentSmoker	cigsPerDay Kişinin ortalama bir günde içtiği sigara sayısı	prevalentHyp Yaygın hipertansiyon	totChol Toplam kolesterol seviyesi
	sysBP Sistolik kan basıncı	diaBP Diyastolik kan basıncı	heartRate Kalp Atış Hızı	glucose Glikoz seviyesi

currentSmoker ve cigsPerDay sütunları veri setinde yarı yarıya dağılım gösterdiği için ve sütun sayısının azaltılmasının skoru düşüreceği düşünüldüğü için eğitime dahil edildi.

Veri hazırlık süreci

Genel bakış

Pandas ve Numpy kütüphaneleri import edildikten sonra df=pd.read_csv("farmington2.csv",sep=",") ile Excel'den veri çekilerek incelenmek üzere Dataframe'ye aktarıldı.

print(df.head()) komutlarıyla verisetine genel bakış yapıldı.

```
: #kütüphanelerin import edilmesi ve dosyanın Excel'den okunması
import numpy as np
import pandas as pd
df=pd.read_csv("farmington2.csv",sep=',')
print(df.head())
```

```
   Unnamed: 0  currentSmoker  cigsPerDay  prevalentHyp  totChol  sysBP  diaBP  \
0           0              1        30.0             1    225.0  150.0   95.0
1           1              0         0.0             0    205.0  138.0   71.0
2        1279              1        10.0             0    192.0   96.5   71.0
3           3              1        20.0             0    291.0  112.0   78.0
4        1273              0         0.0             0    250.0  117.5   75.0

   heartRate  glucose  TenYearCHD
0       65.0    103.0           1
1       60.0     85.0           1
2       61.0     68.0           0
3       80.0     89.0           1
4       75.0     91.0           0
```

Bakış sonrası gereksiz olduğu görülen ["Unnamed: 0"] sütunu drop edildi.

```
#gereksiz ["Unnamed: 0"] sütununun kaldırılması
df.drop(["Unnamed: 0"],axis=1,inplace=True)
print(df.tail())
```

```
      currentSmoker  cigsPerDay  prevalentHyp
1266              1        20.0             0
1267              0         0.0             1
1268              1        43.0             0
1269              1        20.0             1
1270              0         0.0             1

      heartRate  glucose  TenYearCHD
1266       79.0     78.0           0
1267       76.0     79.0           0
1268       75.0     75.0           0
1269       96.0     87.0           0
1270       95.0     83.0           0
```

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

```
#sayısal verilerin incelenmesi
print(df.describe())
```

```
       currentSmoker   cigsPerDay  prevalentHyp      totChol
count    1271.000000  1268.000000   1271.000000  1256.000000
mean        0.512195     9.906151      0.376869   238.607484
std         0.500048    12.578274      0.484792    45.003439
min         0.000000     0.000000      0.000000   107.000000
25%         0.000000     0.000000      0.000000   209.000000
50%         1.000000     1.000000      0.000000   237.000000
75%         1.000000    20.000000      1.000000   265.000000
max         1.000000    60.000000      1.000000   600.000000

             diaBP    heartRate      glucose   TenYearCHD
count  1271.000000  1271.000000  1161.000000  1271.000000
mean     84.338710    76.216365    84.875108     0.495673
std      13.136426    12.065879    31.649984     0.500178
min      48.000000    48.000000    40.000000     0.000000
25%      75.000000    68.000000    72.000000     0.000000
50%      82.500000    75.000000    78.000000     0.000000
75%      92.000000    84.000000    88.000000     1.000000
max     140.000000   140.000000   394.000000     1.000000
```

```
#Veri setine genel bakış
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1271 entries, 0 to 1270
Data columns (total 9 columns):
currentSmoker    1271 non-null int64
cigsPerDay       1268 non-null float64
prevalentHyp     1271 non-null int64
totChol          1256 non-null float64
sysBP            1271 non-null float64
diaBP            1271 non-null float64
heartRate        1271 non-null float64
glucose          1161 non-null float64
TenYearCHD       1271 non-null int64
dtypes: float64(6), int64(3)
memory usage: 89.4 KB
None
```

print(df.info()) komutuyla veri setine genel bakış yapıldı. İnceleme zamanı bazı sütunlarda boş değerler olduğu görüldü.

Özellik seçimi

Veri setinin 8 özelliği(sütunu) eğitim için kullanılmıştır. Sütunlar üzerinde herhangi değişiklik yapılmadı.

Veri temizliği ve dönüşümü

isnull().sum()) ile kayıp verilerin tespiti yapıldı.3 sütunda kayıp verilerin olduğu görüldü.

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

```
currentSmoker      0
cigsPerDay         3
prevalentHyp       0
totChol           15
sysBP              0
diaBP              0
heartRate          0
glucose          110
TenYearCHD         0
dtype: int64
128
```

Eksik verilerin doldurulması

Eksik verilerin doldurulmasında Forward Interpolasyon (ileri interpolasyon) yöntemi kullanılmıştır.

```
#İnterpolasyon yöntemiyle kayıp verilerin doldurulması
df["cigsPerDay"]=df["cigsPerDay"].interpolate()
df["totChol"]=df["totChol"].interpolate()
df["glucose"]=df["glucose"].interpolate()
```

Interpolasyon sonrası boş değerlerin tamamen doldurulduğu görüldü.

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

```
0
```

Veri tiplerinin dönüşümü

Veri setindeki sütunların veri tipleri float ve integer olduğundan veri tipleriyle ilgili değişiklik yapılmadı.

Tekrarlayan verilerin kaldırılması

Tekrarlayan veriler print(df[df.duplicated()]) komutuyla kontrol edildi ve tekrarlayan veri tespit edilmedi.

```
#veri tekrarı kontrolü
print(df[df.duplicated()])
```

```
Empty DataFrame
Columns: [currentSmoker, cigsPerDay,
Index: []
```

Aşırı verilerin kontrolü

Her sütun için ortalama değer, sapma değeri ve ortanca değeri kontrol edildi. Ortalama ve ortanca değerleri arasında çok fazla farklılık görülmedi.

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

```
9.933910306845004
12.578460794441465
1.0
```

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

```
238.5507474429583
44.82187824183549
237.0
```

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

```
136.72147915027537
24.899671180499684
132.0
```

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

```
84.33870967741936
13.136426349330872
82.5
```

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

```
76.21636506687648
12.065878864016032
75.0
```

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

```
85.02439024390245
31.009261443740677
79.0
```

Normalizasyon ve x, y değerlerinin tanımlanması

Çıkış değeri olarak (0,1) şeklinde sonuç veren TenyearCHD sütunu Y değeri olarak seçildi. Diğer sütunlar ise Normalizasyon işlemine tabi tutularak Giriş değeri olan X'e aktarıldı.

TenyearCHD sütunun normalize edilmiş hali kendisine eşit olacağından bu sütun için normalizasyon yapılmadı.

```
#Normalizasyon
y=df.TenYearCHD.values
x_data=df.drop(["TenYearCHD"],axis=1).values
x=(x_data-np.min(x_data))/(np.max(x_data)-np.min(x_data))
```

MLP ile Modelin eğitilmesi

X ve Y değerleri tanımlandıktan sonra veri seti eğitim ve test şeklinde bölündü. Veri setinin büyük olmamasından dolayı test boyutu 0,2 olarak seçildi. Ardından x_train, x_test, y_train, y_test şeklinde bölünen matrislerin boyutlarına bakıldı.

```
#Veri setinin train - test şeklinde ayrılması ve boyutları
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=1)
print("x_train:",x_train.shape)
print("y_train:",y_train.shape)
print("x_test:",x_test.shape)
print("y_test:",y_test.shape)
```

```
x_train: (1016, 8)
y_train: (1016,)
x_test: (255, 8)
y_test: (255,)
```

Öncelikle hiper-parametreler verilmeden Çok Katmanlı Perceptron (MLP) eğitimi yapıldı. Bunun için neural_network kütüphanesinden MLPClassifier import edildi ve eğitim verileri fit edildi.

```
#Parametre eklemeden yapılan MLP eğitimi
from sklearn.neural_network import MLPClassifier

mlpc=MLPClassifier(random_state=1)
mlpc.fit(x_train,y_train)
```

```
C:\Users\ABBSCBN\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer
c Optimizer: Maximum iterations (200) reached and the optimization hasn't conv
  % self.max_iter, ConvergenceWarning)
```

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=1, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)
```

Yapılan eğitim için Accuracy – Doğruluk , F1 skoru, Confusion matrix – Karışıklık matrisi gibi değerlendirme sonuçları aşağıda verilmiştir.

```
#Genel değerlendirilmesi - Accuracy, F1 skoru, Karışıklık Matrisi
import sklearn.metrics as metrics

y_pred=mlpc.predict(x_test)
acc=metrics.accuracy_score(y_test,y_pred)
print("Accuracy:",acc)
cm=metrics.confusion_matrix(y_test,y_pred)
print(cm)
cr=metrics.classification_report(y_test,y_pred)
print(cr)
```

```
Accuracy: 0.6235294117647059
[[91 31]
 [65 68]]
              precision    recall  f1-score   support

           0       0.58      0.75      0.65       122
           1       0.69      0.51      0.59       133

   micro avg       0.62      0.62      0.62       255
   macro avg       0.64      0.63      0.62       255
weighted avg       0.64      0.62      0.62       255
```

Eğitim sonucuna baktığımızda çok kötü sayılmayan acc değeri (0.623) görülmektedir. Fakat acc değeri modeli yorumlamak için yeterli değildir. Kullanılan diğer bir değerlendirme yöntemi olan Karışıklık matrisine baktığımızda ise False Negative ve False Positive kısımlarında sıkıntıların olduğu görülmektedir. FN ve FP değerlerini ola bildiğince minimum olması Karışılık matrisi için en önemli parametrelerdendir. F1 skoruna baktığımızda ise skora göre model 0 (Hasta olmayan) değerlerini 1(Hasta) değerlerine göre daha iyi tahmin etmektedir. Fakat bu skorlar kabul edilebilir bir skorlar değildir.

KFold Cross Validation

Eğitimde ezberlemeyi önlemek amacıyla KFold cv kullanılmıştır. Veriler seçilen K değeri kadar parçaya bölünerek modelinin görmediği veriler üzerindeki performansını görmek için kullanılmaktadır. Her seferinde daha önce kullanılmamış K kadar bölünmüş kadarı test için kalan ise eğitim için kullanılır. Çıkan metrik değerlerinin ortalaması alınır.

Küçük veri seti olduğundan K=3 olarak seçildi. Kfold sırasında MLP eğitimi de yapıldığından MLP için bazı Hiperparametreler verildi. Veri seti orta büyüklükte (1271) olduğundan Solver olarak sgd seçilmiştir. Çıkışlar 0 ve 1 şeklinde olduğundan (Sigmoid) Activation olarak logistic secilmiştir. Gizli katman boyutu olarak ise sırasıyla 3,5,3 nörondan oluşan 3 katman belirlenmiştir. Ortalama skor 0.475 olarak görüldü.

```
#Kfold cross validation ile veri seti analizi.K=3
from sklearn.model_selection import KFold
from sklearn.neural_network import MLPClassifier

scores=[]

kf=KFold(n_splits=3,random_state=1)
mlpc=MLPClassifier(solver="sgd",activation="logistic",hidden_layer_sizes=(3,5,3))

for train_indexler,test_indexler in kf.split(x):
    mlpc.fit(x[train_indexler],y[train_indexler])
    score=mlpc.score(x[test_indexler],y[test_indexler])
    scores.append(score)
    print(score)

print("ortalama score:",np.mean(scores))
```

```
0.5
0.4693396226415094
0.4562647754137116
ortalama score: 0.475201466018407
```

Kfold ile ikinci deneme K=10 olarak yapıldı. Bölünen parçaların skorlarının bir birine yakın olması beklenmektedir. Ortalama skor 0.516 olarak görüldü.

```
#Kfold cross validation ile veri seti analizi. K=10
from sklearn.model_selection import KFold
from sklearn.neural_network import MLPClassifier

scores=[]

kf=KFold(n_splits=10,random_state=1)
mlpc=MLPClassifier(solver="sgd",activation="logistic",hidden_layer_sizes=(3,5,3))

for train_indexler,test_indexler in kf.split(x):
    mlpc.fit(x[train_indexler],y[train_indexler])
    score=mlpc.score(x[test_indexler],y[test_indexler])
    scores.append(score)
    print(score)

print("ortalama score:",np.mean(scores))
```

```
0.5390625
0.48031496062992124
0.5433070866141733
0.5196850393700787
0.4566929133858268
0.5196850393700787
0.4645669291338583
0.4881889763779528
0.5511811023622047
0.5984251968503937
ortalama score: 0.5161109744094488
```

GridSearch Cross Validation

Sklearn Model_selection kütüphanesinden GridSearchCV import edildi. Modelde denenmesi istenen hiperparametreler ve değerleri için bütün kombinasyonlarını tek tek deneyen ve belirtilen metriğe göre en başarılı hiperparametreye sahip seti gösteren bu yöntem kullanıcı için zaman kaybını ortadan kaldırmaktadır.

MLP parametreleri olarak alpha için 0.1, 0.01, 0.001, 0.0001 olacak şekilde 4 parametre verildi. Gizli katman boyutu olarak (10,10,10), (3,3,3), (100,100), (3,5) olarak 4 parametre verildi. Ağırlık optimizasyonu olan solver için lbfgs, adam, sgd olarak 3 parametre verildi. Toplam 48*3=144 (K=3) defa model denendi.

```
#grid search cv ile en iyi parametrelerin belirlenmesi cv=3
from sklearn.model_selection import GridSearchCV

mlpc_params={"alpha":[0.1,0.01,0.001,0.0001],
             "hidden_layer_sizes":[(10,10,10),(3,3,3),(100,100),(3,5)],
             "solver":["lbfgs","adam","sgd"]}

mlpc=MLPClassifier(activation="logistic",random_state=1)

mlpc_cv_model=GridSearchCV(mlpc,mlpc_params,cv=3,n_jobs=-1,verbose=2).fit(x_train,y_train)
```

```
Fitting 3 folds for each of 48 candidates, totalling 144 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    4.2s
[Parallel(n_jobs=-1)]: Done 144 out of 144 | elapsed:    9.3s finished
```

En iyi parametreler mlpc_cv_model.best_params ile gösterildi.

```
#en iyi parametrelerin gösterimi
print(mlpc_cv_model.best_params_)
```

```
{'alpha': 0.1, 'hidden_layer_sizes': (10, 10, 10), 'solver': 'lbfgs'}
```

Bu parametrelerle MLP eğitimi yapıldı. Max iterasyon sayısı olarak 100 seçildi. Eğitim skoru olarak 0.62 görüldü.

```
#belirlenen parametrelerle model sonucu
mlpc1=MLPClassifier(alpha=0.1,hidden_layer_sizes=(10,10,10),solver="lbfgs",max_iter=100,random_state=1)
mlpc1.fit(x_train,y_train)
print("score:",mlpc1.score(x_test,y_test))
```

```
score: 0.6274509803921569
```

CV değeri 10 seçilerek Grid Search işlemi tekrarlandı. Deneme sayısı arttığından dolayı en iyi parametrelerin tespit edilmesi daha geç sonuçlandı.

```
#grid search cv ile en iyi parametrelerin belirlenmesi cv=10
from sklearn.model_selection import GridSearchCV

mlpc_params={"alpha":[0.1,0.01,0.001,0.0001],
             "hidden_layer_sizes":[(10,10,10),(3,5,3),(100,100),(3,5)],
             "solver":["lbfgs","adam","sgd"]}

mlpc=MLPClassifier(activation="relu",random_state=1)

mlpc_cv_model=GridSearchCV(mlpc,mlpc_params,cv=10,n_jobs=-1,verbose=2).fit(x_train,y_train)
```

```
Fitting 10 folds for each of 48 candidates, totalling 480 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    2.2s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   27.7s
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done 480 out of 480 | elapsed:  2.3min finished
C:\Users\ABBSCBN\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:841: Deprecatic
d` parameter will change from True to False in version 0.22 and will be removed in 0.24. This w
test-set sizes are unequal.
  DeprecationWarning)
```

CV=10 olduğunda en iyi parametreler aşağıdaki gibi görüldü.

```
#en iyi parametrelerin gösterimi cv=10
print(mlpc_cv_model.best_params_)
```

```
{'alpha': 0.0001, 'hidden_layer_sizes': (10, 10, 10), 'solver': 'lbfgs'}
```

CV=10 olduğunda MLP eğitimi. Eğitim skoru olarak 0.627 görüldü.

```
#belirlenen parametrelerle model sonucu
mlpc1=MLPClassifier(alpha=0.0001,hidden_layer_sizes=(10,10,10),solver="lbfgs",max_iter=1000,random_state=1)
mlpc1.fit(x_train,y_train)
print("score:",mlpc1.score(x_test,y_test))
```

```
score: 0.6274509803921569
```

Ardından bu parametrelerle max iterasyon sayısı 10 ve 100 olacak şekilde denemeler de yapıldı. Max_iter=10 için 0.6, max_iter=100 için 0.635 skoru görüldü ve eğitim için görülen en yüksek skor oldu.

```
#belirlenen parametrelerle model sonucu max_iter=10
mlpc2=MLPClassifier(alpha=0.0001,hidden_layer_sizes=(10,10,10),solver="lbfgs",max_iter=10,random_state=1)
mlpc2.fit(x_train,y_train)
print("score:",mlpc2.score(x_test,y_test))
```

```
score: 0.6
```

```
#belirlenen parametrelerle model sonucu max_iter=1000
mlpc3=MLPClassifier(alpha=0.0001,hidden_layer_sizes=(10,10,10),solver="lbfgs",max_iter=100,random_state=1)
mlpc3.fit(x_train,y_train)
print("score:",mlpc3.score(x_test,y_test))
```

```
score: 0.6352941176470588
```

Eğitimin değerlendirilmesi

Accuracy – Doğruluk , F1 skoru, Confusion matrix – Karışıklık matrisi ile yapıldı. Sonuçlar aşağıda verilmiştir.

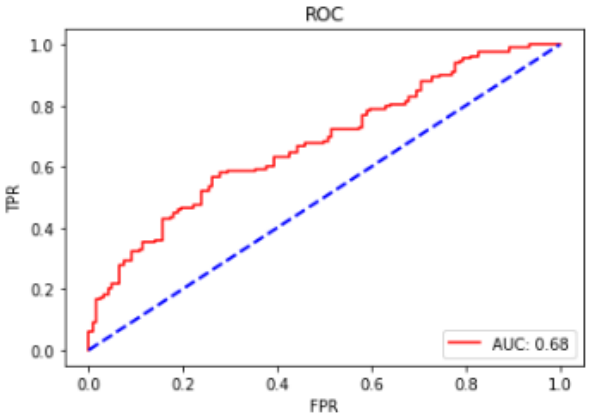
```
import sklearn.metrics as metrics

y_pred=mlpc3.predict(x_test)
acc=metrics.accuracy_score(y_test,y_pred)
print("Accuracy:",acc)
cm=metrics.confusion_matrix(y_test,y_pred)
print(cm)
cr=metrics.classification_report(y_test,y_pred)
print(cr)
```

```
Accuracy: 0.6352941176470588
[[84 38]
 [55 78]]
              precision    recall  f1-score   support

           0       0.60      0.69      0.64       122
           1       0.67      0.59      0.63       133

   micro avg       0.64      0.64      0.64       255
   macro avg       0.64      0.64      0.64       255
weighted avg       0.64      0.64      0.63       255
```

AUC ve ROC

Model için ROC çizimi yapıldı. False Positive Rate ve True Positive Rate oranlarına göre çizilen farklı sınıflar için olasılık eğrisidir. AUC ise ROC eğrisinin altında kalan alandır. Eğrinin kapsama alanı ne kadar büyükse o kadar iyi veri ayırtedilebilir.AUC'un max değeri 1'dir. Bizim eğitim için AUC =0.68 olarak hesaplanmıştır.

```
#roc and auc çizimi
import matplotlib.pyplot as plt
probs=mlpc3.predict_proba(x_test)
#print(probs)
probs=probs[:,1]
fpr,tpr,threshold=metrics.roc_curve(y_test,probs)
auc_value=metrics.auc(fpr,tpr)
plt.title("ROC")
plt.plot(fpr,tpr,label="AUC: "+str(round(auc_value,2)),color="red")
plt.plot([0,1],[0,1],color="blue",lw=2,linestyle="--")
plt.legend(loc="lower right")
plt.ylabel("TPR")
plt.xlabel("FPR")
plt.show()
```



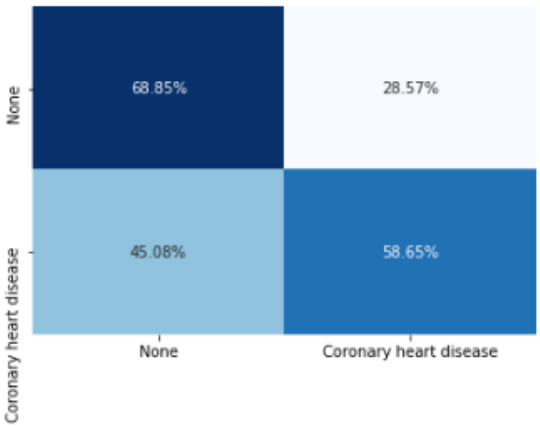
Karışıklık matrisi çizimi için Seaborn kütüphanesi import edildi. "None" ve "Coronary heart disease" olarak klas adları tanımlamdı. Mavi renkli, yüzde olarak virgülden sonra 2 basamak olacak şekilde Karışıklık matrisi çizdirildi.

```
#Seaborn kütüphanesi tanımlanması
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
```

```
class_adlari=["None","Coronary heart disease"]
sns.heatmap(cm/cm.sum(axis=1), annot=True, cbar=False, cmap="Blues",xticklabels=class_adlari,yticklabels=class_adlari,fmt=".2%")
```

Karışıklık matrisininden hasta olmayıp (None) hasta olmayarak tahmin edilenlerin(68,85) ve hasta olup (CHD) hasta olarak tahmin edilenlerin (58,65) oranları görülmektedir. Bundan ziyade hasta olmadığı halde CHD olarak etiketlenlerin toplama oranı %45.08, hasta olup da hasta değil şeklinde etiketlenenlerin oranı ise %28.57 dir.

```
<matplotlib.axes._subplots.AxesSubplot at 0x1aaa8a8cba8>
```



Keras ile modelin eğitilmesi ve değerlendirilmesi

Keras kütüphanesinden Sequential (Ardışık), öğrenme katmanları oluşturmak için Dense, Aktivasyon fonk için Activation, verileri azaltmak için Dropout, normalizasyo için BatchNormalizasyon, çizim için matplotlib.pyplot ve Numpy da import edildi.

```
#Keras kütüphaneleri import edilmesi
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, BatchNormalization
import matplotlib.pyplot as plt
import numpy as np
```

Veriler eğitim ve test olarak bölündü. Test_size=0.2

```
#modelin train - test çin ayrılması
from sklearn.model_selection import train_test_split
x_train, x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=1)
```

Eğitilen verinin boyutu kontrol edildi.

```
#Eğitilen veri boyutunun gösterilmesi
input_shape=x_train.shape[1:]
print(input_shape)
```

```
(8,)
```

Model katmanları aşağıdaki gibi oluşturuldu. İlk oluşturulan tensörde 16 nöron olan hidden layer, activation fonk olarak relu, input_dim olarak ise input matrisi bpyutu olarak 8 seçildi. Normalizasyon katmanı olarak BatchNormalizasyon eklendi. Ve her katman sonrası veriyi azaltmak için Dropout kullanıldı. Nöron sayısı 2. Katmanda 32'ye çıkarıldı. 3. Ve 4. Katmanda activation fonk olarak sigmoid seçildi ve nöron

```
#Model katmanlarının oluşturulması
model = Sequential()
model.add(Dense(16, activation='relu', input_dim=8))
model.add(BatchNormalization())
model.add(Dropout(0.25))

model.add(Dense(32, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.25))

model.add(Dense(64, activation='sigmoid'))
model.add(BatchNormalization())
model.add(Dropout(0.25))

model.add(Dense(128, activation='sigmoid'))
model.add(BatchNormalization())
model.add(Dropout(0.25))

model.add(Dense(256, activation='relu'))

model.add(Dense(10, activation='softmax'))
```

sayısı sıralı olarak artırıldı. Son katman olarak 10 nöronluk softmax seçildi.

Model compile edildi. Optimizer olarak veri sayısı 1000 üzerin olduğundan sgd seçildi. Loss olarak en olası eşleşen kategorinin bir kategori dizinini üretmesi için sparse_categorical_crossentropy kullanıldı. Metrik olarak ise acc seçildi.

```
#modelin compile edilmesi
model.compile(optimizer="sgd",loss="sparse_categorical_crossentropy",metrics=["accuracy"])
```

```
WARNING:tensorflow:From C:\Users\ABBSCBN\Anaconda3\lib\site-packages\keras\backend\tensorflow_ba
ax_v1 (from tensorflow.python.ops.math_ops) with keep_dims is deprecated and will be removed in
Instructions for updating:
keep_dims is deprecated, use keepdims instead
```

200 iterasyonla modelin fit edilmesine başlandı. Batch_size olarak 32 seçildi. Modelin kaç veri işleyeceğinin ölçüsü olan Batch_size'in yüksek olması hızlı işlem açısından iyidir, fakat Loss değerini artar. Aşağıdaki şekilde görüldüğü üzere ilk iterasyonda loss=1.35, acc=0.47. Modeli iyi hale gelmesi için loss değerinin olabildiğince düşük, acc değerinin ise yükselmesi istenmektedir.

```
#modelin fit edilmesi iterasyon sayı=200
egitim=model.fit(x_train,y_train,epochs=200,batch_size=32,validation_data=(x_test,y_test))
```

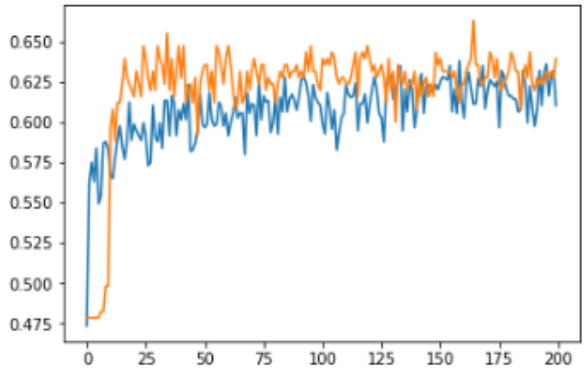
```
WARNING:tensorflow:From C:\Users\ABBSCBN\AppData\Roaming\Python\Python37\site-packages\tensorflow\python\ops\math_
6: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 1016 samples, validate on 255 samples
Epoch 1/200
1016/1016 [==============================] - 1s - loss: 1.3525 - acc: 0.4734 - val_loss: 1.5997 - val_acc: 0.4784
Epoch 2/200
1016/1016 [==============================] - 0s - loss: 0.8853 - acc: 0.5620 - val_loss: 1.4929 - val_acc: 0.4784
Epoch 3/200
1016/1016 [==============================] - 0s - loss: 0.8112 - acc: 0.5748 - val_loss: 1.4862 - val_acc: 0.4784
Epoch 4/200
```

200. iterasyonda loss=0.65, acc=0.61 olduğu görülmektedir. Modelin öğrenmede başarılı olduğu söylenebilir.

```
1016/1016 [==============================] - 0s - loss: 0.6491 - acc: 0.6280 - val_loss: 0.6432 - val_acc: 0.6314
Epoch 199/200
1016/1016 [==============================] - 0s - loss: 0.6508 - acc: 0.6319 - val_loss: 0.6412 - val_acc: 0.6275
Epoch 200/200
1016/1016 [==============================] - 0s - loss: 0.6574 - acc: 0.6102 - val_loss: 0.6479 - val_acc: 0.6392
```

Modelin eğitim sırasında acc ve val_acc değerlerinin artışı aşağıdaki grafikte verilmiştir. Grafikten görüldüğü üzere acc değerinde 0.475'ten 0.60 lara kadar yükselme gözlemlenmiştir. Turuncu ve mavi eğrilerin birbirine paralel olarak artması istemektedir.

```
#model acc ve va_acc değerlerinin artış grafiği
plt.plot(egitim.history["acc"])
plt.plot(egitim.history["val_acc"])
plt.show()
```



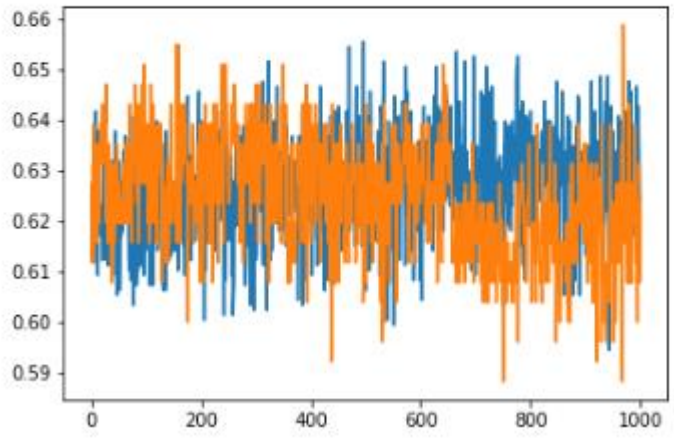
1000 iterasyonla aşağıdaki şekilde deneme yapıldı.

```
#modelin fit edilmesi iterasyon sayı=1000
egitim=model.fit(x_train,y_train,epochs=1000,batch_size=32,validation_data=(x_test,y_test))
```

```
Epoch 991/1000
1016/1016 [==============================] - 0s - loss: 0.6433 - acc: 0.6211 - val_loss: 0.6627 - val_acc: 0.6275
Epoch 992/1000
1016/1016 [==============================] - 0s - loss: 0.6441 - acc: 0.6388 - val_loss: 0.6605 - val_acc: 0.61180.633
Epoch 993/1000
1016/1016 [==============================] - 0s - loss: 0.6410 - acc: 0.6270 - val_loss: 0.6636 - val_acc: 0.6118
Epoch 994/1000
1016/1016 [==============================] - 0s - loss: 0.6452 - acc: 0.6467 - val_loss: 0.6561 - val_acc: 0.6118
Epoch 995/1000
1016/1016 [==============================] - 0s - loss: 0.6485 - acc: 0.6230 - val_loss: 0.6649 - val_acc: 0.6000
Epoch 996/1000
1016/1016 [==============================] - 0s - loss: 0.6445 - acc: 0.6270 - val_loss: 0.6549 - val_acc: 0.6235
Epoch 997/1000
1016/1016 [==============================] - 0s - loss: 0.6466 - acc: 0.6427 - val_loss: 0.6619 - val_acc: 0.6196
Epoch 998/1000
1016/1016 [==============================] - 0s - loss: 0.6513 - acc: 0.6348 - val_loss: 0.6495 - val_acc: 0.6314
Epoch 999/1000
1016/1016 [==============================] - 0s - loss: 0.6512 - acc: 0.6280 - val_loss: 0.6686 - val_acc: 0.6078
Epoch 1000/1000
1016/1016 [==============================] - 0s - loss: 0.6450 - acc: 0.6240 - val_loss: 0.6504 - val_acc: 0.6235
```

Fazla iterasyonla acc değeri çok fazla yükselmediğinden aşağıdaki gibi grafik elde edildi.

```
plt.plot(egitim.history["acc"])
plt.plot(egitim.history["val_acc"])
plt.show()
```



Meme Kanseri Hastalarında Ölüm Sebebi Tahmini

Veri seti seçimi

Bu veri setinde Metabrik veri tabanı tarafından 2.509 meme kanseri hastasının klinik profilleri verilmiştir. Veri setide 34 özellik mevcuttur.

Bu araştırma, meme kanseri hastalarının ölüm nedenlerine göre hastalığın genel ölüm sebebini tahmin etmeyi amaçlamaktadır. Veri seti Kaggle web sitesinde halka açık olarak mevcuttur. Living, Death of Cause ve Death of Disease şeklinde 3 class'tan oluşan sınıflandırma amacı, çeşitli özelliklere kanser hastalarının ölüm sebebininin tahmin edilmesidir.

Veri seti 2509 satır ve 34 sütundan oluşmaktadır. Her özellik, potansiyel bir risk faktörüdür [3].

Sütunlarda eksik veriler olduğundan eğitimde veri setinin 1309 satırı ve 8 özelliği kullanılmıştır. Kullanılan sütunlar aşağıdaki gösterilmiştir.

Y	X			
Patient's Vital Status	Tumor Stage	Tumor Size	Neoplasm Histologic Grade	Lymp nodes examined positive
	Mutation count	Nottingham prognostic index	Overall survival (Months)	Relapse free status (Months)

Veri hazırlık süreci

Genel bakış

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

print(df.head()) komutlarıyla verisetine genel bakış yapıldı.

```
#kütüphanelerin import edilmesi ve dosyanın Excel'den okunması
import numpy as np
import pandas as pd
df=pd.read_csv("bcm.csv")
print(df.head())
```

```
  Patient ID  Age at Diagnosis Type of Breast Surgery    Cancer Type  \
0    MB-0000             75.65             Mastectomy  Breast Cancer
1    MB-0002             43.19      Breast Conserving  Breast Cancer
2    MB-0005             48.87             Mastectomy  Breast Cancer
3    MB-0006             47.68             Mastectomy  Breast Cancer
4    MB-0008             76.97             Mastectomy  Breast Cancer

                     Cancer Type Detailed Cellularity Chemotherapy  \
0          Breast Invasive Ductal Carcinoma         NaN           No
1          Breast Invasive Ductal Carcinoma        High           No
2          Breast Invasive Ductal Carcinoma        High          Yes
3  Breast Mixed Ductal and Lobular Carcinoma    Moderate          Yes
4  Breast Mixed Ductal and Lobular Carcinoma        High          Yes
```

```
#Veri setine genel bakış
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2509 entries, 0 to 2508
Data columns (total 34 columns):
Patient ID                          2509 non-null object
Age at Diagnosis                    2498 non-null float64
Type of Breast Surgery              1955 non-null object
Cancer Type                         2509 non-null object
Cancer Type Detailed                2509 non-null object
Cellularity                         1917 non-null object
Chemotherapy                        1980 non-null object
Pam50 + Claudin-low subtype         1980 non-null object
Cohort                              2498 non-null float64
ER status measured by IHC           2426 non-null object
ER Status                           2469 non-null object
Neoplasm Histologic Grade           2388 non-null float64
HER2 status measured by SNP6        1980 non-null object
HER2 Status                         1980 non-null object
Tumor Other Histologic Subtype      2374 non-null object
Hormone Therapy                     1980 non-null object
Inferred Menopausal State           1980 non-null object
Integrative Cluster                 1980 non-null object
Primary Tumor Laterality            1870 non-null object
Lymph nodes examined positive       2243 non-null float64
Mutation Count                      2357 non-null float64
Nottingham prognostic index         2287 non-null float64
Oncotree Code                       2509 non-null object
Overall Survival (Months)           1981 non-null float64
Overall Survival Status             1981 non-null object
PR Status                           1980 non-null object
Radio Therapy                       1980 non-null object
Relapse Free Status (Months)        2388 non-null float64
Relapse Free Status                 2488 non-null object
Sex                                 2509 non-null object
3-Gene classifier subtype           1764 non-null object
Tumor Size                          2360 non-null float64
Tumor Stage                         1788 non-null float64
vital_status                        1980 non-null object
dtypes: float64(10), object(24)
memory usage: 666.5+ KB
None
```

print(df.info()) komutuyla verisetinin sütunlarının bilgileri incelendi.

print(df.describe()) komutuyla sayısal veriler incelendi.

```
#Veri setindeki sayısal verilere genel bakış
print(df.describe())
```

```
       Age at Diagnosis       Cohort  Neoplasm Histologic Grade  \
count       2498.000000  2498.000000                2388.000000
mean          60.420300     2.900320                   2.412060
std           13.032997     1.962216                   0.649363
min           21.930000     1.000000                   1.000000
25%           50.920000     1.000000                   2.000000
50%           61.110000     3.000000                   3.000000
75%           70.000000     4.000000                   3.000000
max           96.290000     9.000000                   3.000000

       Lymph nodes examined positive  Mutation Count  \
count                    2243.000000     2357.000000
mean                        1.950513        5.578702
std                         4.017774        3.967967
min                         0.000000        1.000000
25%                         0.000000        3.000000
50%                         0.000000        5.000000
75%                         2.000000        7.000000
max                        45.000000       80.000000
```

Özellik seçimi

Kullanılacak sütunlar `loc` ile seçildi.

```
#Kullanılacak sütunların seçilmesi
df=df.loc[:,["Neoplasm Histologic Grade","Lymph nodes examined positive","Mutati
print(df.head())
```

```
   Neoplasm Histologic Grade  Lymph nodes examined positive  Mutation Count  \
0                        3.0                           10.0             NaN
1                        3.0                            0.0             2.0
2                        2.0                            1.0             2.0
3                        2.0                            3.0             1.0
4                        3.0                            8.0             2.0

   Nottingham prognostic index  Overall Survival (Months)  \
0                        6.044                 140.500000
1                        4.020                  84.633333
2                        4.030                 163.700000
3                        4.050                 164.933333
4                        6.080                  41.366667

   Relapse Free Status (Months)  Tumor Size  Tumor Stage     vital_status
0                        138.65        22.0          2.0           Living
1                         83.52        10.0          1.0           Living
2                        151.28        15.0          2.0  Died of Disease
3                        162.76        25.0          2.0           Living
4                         18.55        40.0          2.0  Died of Disease
```

Veri temizliği ve dönüşümü

Eksik verilerin tespiti yapıldı. Birçok sütunda eksik veri olduğu görüldü.

```
#Eksik veri kontrolü
print(df.isnull().sum())
```

```
Neoplasm Histologic Grade         121
Lymph nodes examined positive     266
Mutation Count                    152
Nottingham prognostic index       222
Overall Survival (Months)         528
Relapse Free Status (Months)      121
Tumor Size                        149
Tumor Stage                       721
vital_status                      529
dtype: int64
```

```
#Eksik verilerin silinmesi
df.dropna(inplace=True)
print(df.head())
```

Veri seti için veri doldurma işleminin yapılması eksik verinin fazla olmasından uygun görülmedi. Eksik verilerin olduğu satırlar veri setinden çıkarıldı.

Satırlar silindikten sonra eksik veri kkontrölü tekrar yapıldı. Hiç eksik verinin olmadığı görüldü.

```
#Eksik veri kontrolü
print(df.isnull().sum())
```

```
Neoplasm Histologic Grade         0
Lymph nodes examined positive     0
Mutation Count                    0
Nottingham prognostic index       0
Overall Survival (Months)         0
Relapse Free Status (Months)      0
Tumor Size                        0
Tumor Stage                       0
vital_status                      0
dtype: int64
```

`Reset_index` ile indekler resetlendi.

```
#İndekslerin sıfırlanması
df.reset_index(inplace=True,drop=True)
```

Dtypes ile veri tipleri kontrol edildi. Vital_status dışındaki sütunların float tipinde olduğu görüldü.

```
#Veri tipi kontrolü
df.dtypes
```

```
Neoplasm Histologic Grade         float64
Lymph nodes examined positive     float64
Mutation Count                    float64
Nottingham prognostic index       float64
Overall Survival (Months)         float64
Relapse Free Status (Months)      float64
Tumor Size                        float64
Tumor Stage                       float64
vital_status                       object
dtype: object
```

Unique() output sınıfı olan vital_status'un veri çeşidine bakıldı

```
#output sınıfının çeşidinin gösterilmesi
print(df.vital_status.unique())
```

```
['Living' 'Died of Disease' 'Died of Other Causes']
```

Yazılan kodla sonuçlar Living - 0, Died of disease - 2, Died of other causes - 1 olarak değiştirildi. Died of disease bizim için daha önemli olduğundan 2 olarak seçildi.

```
#Output sınıfının Living=0, Died of Disease=2, Died of Other Causes=1 şeklinde değiştirilmesi
df.vital_status=[0 if each =="Living" else 2 if each=="Died of Disease" else 1 for each in df.vital_status]
print(df.head())
```

Duplicated() ile veri tekrarı kontrolü yapıldı.

```
#veri tekrarı kontrolü
print(df[df.duplicated()])
```

```
Empty DataFrame
Columns: [Neoplasm Histologic Grade, Lymph nodes examined pos
al (Months), Relapse Free Status (Months), Tumor Size, Tumor
Index: []
```

İşlemler sonrasında describe() ile sayısal veri kontrolü tekrardan yapıldı.

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

```
       Neoplasm Histologic Grade  Lymph nodes examined positive  \
count                1309.000000                    1309.000000
mean                    2.442322                       1.870130
std                     0.641872                       3.849215
min                     1.000000                       0.000000
25%                     2.000000                       0.000000
50%                     3.000000                       0.000000
75%                     3.000000                       2.000000
max                     3.000000                      41.000000

       Mutation Count  Nottingham prognostic index  Overall Survival (Months)  \
count     1309.000000                  1309.000000                1309.000000
mean         5.471352                     4.118338                 127.629132
std          3.808908                     1.061604                  78.424011
min          1.000000                     2.002000                   0.100000
25%          3.000000                     3.052000                  61.433333
50%          5.000000                     4.046000                 117.666667
75%          7.000000                     5.044000                 189.133333
max         46.000000                     6.360000                 351.000000

       Relapse Free Status (Months)   Tumor Size  Tumor Stage  vital_status
count                   1309.000000  1309.000000  1309.000000   1309.000000
mean                     111.868953    25.937739     1.757830      0.909855
std                       78.981497    15.013709     0.620885      0.876713
min                        0.000000     1.000000     1.000000      0.000000
25%                       41.220000    17.000000     1.000000      0.000000
50%                       99.740000    22.000000     2.000000      1.000000
75%                      173.030000    30.000000     2.000000      2.000000
max                      346.380000   180.000000     4.000000      2.000000
```

Normalizasyon yapıldı ve X, Ydeğerleri tanımlandı.

```
#Normalizasyon
y=df.vital_status.values
x_data=df.drop(["vital_status"],axis=1).values
x=(x_data-np.min(x_data))/(np.max(x_data)-np.min(x_data))
```

Veri seti train – test şeklinde bölündü. test_size=0.3. Eğitim ve test matrislerinin boyutlarına `reshape` ile bakıldı.

```
#Veri setinin train - test şeklinde ayrılması ve boyutları
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)
print("x_train:",x_train.shape)
print("y_train:",y_train.shape)
print("x_test:",x_test.shape)
print("y_test:",y_test.shape)
```

```
x_train: (916, 8)
y_train: (916,)
x_test: (393, 8)
y_test: (393,)
```

MLP ile modelin eğitilmesi ve değerlendirilmesi

Hiperparametreler eklemeden aşağıdaki gibi MLP eğitimi yapıldı.

```
#Parametre eklenmeden yapılan MLP eğitimi
from sklearn.neural_network import MLPClassifier

mlpc=MLPClassifier(random_state=1)
mlpc.fit(x_train,y_train)
```

```
C:\Users\ABBSCBN\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_
c Optimizer: Maximum iterations (200) reached and the optimization hasn't conve
  % self.max_iter, ConvergenceWarning)
```

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=1, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)
```

Accuracy, F1 skoru, Karışıklık matrisi ile model değerlendirildi. Acc olarak 0.58 hesaplandı. Karışıklık matrisine bakınca ise 1 değerini tahmin etmede sıkıntıların olduğu görüldü.

```
#Genel değerlendirilmesi - Accuracy, F1 skoru, Karışıklık Matrisi
import sklearn.metrics as metrics

y_pred=mlpc.predict(x_test)
acc=metrics.accuracy_score(y_test,y_pred)
print("Accuracy:",acc)
cm=metrics.confusion_matrix(y_test,y_pred)
print(cm)
cr=metrics.classification_report(y_test,y_pred)
print(cr)
```

```
Accuracy: 0.5877862595419847
[[146   0  47]
 [ 62   0  29]
 [ 24   0  85]]
              precision    recall  f1-score   support

           0       0.63      0.76      0.69       193
           1       0.00      0.00      0.00        91
           2       0.53      0.78      0.63       109

   micro avg       0.59      0.59      0.59       393
   macro avg       0.39      0.51      0.44       393
weighted avg       0.46      0.59      0.51       393
```

Kfold Cross Validation

Ezberlemeyi önlemek amacıyla kullanılan Kfold cv öncelikle K=3 olacak şekilde yapıldı. MLP için hiperparametreler olarak solver için deneme amaçlı sgd (veri sayısı 1000'in altında olduğundan lbfgs daha uygun), aktivasyon fonk olarak multi class olduğu için relu, gizli katman olarak (3,5,3) nöronlardan oluşan 3 katman ve iterasyon sayısı olarak 1000 seçildi. Bölünen kısım skorları yakın değerler çıktı. Ortalama skor=0.58

```
#Kfold cross validation ile veri seti analizi. K=3
from sklearn.model_selection import KFold

scores=[]

kf=KFold(n_splits=3,random_state=1)
mlpc=MLPClassifier(solver="sgd",activation="relu",hidden_layer_sizes=(3,5,3),max_iter=1000)

for train_indexler,test_indexler in kf.split(x):
    mlpc.fit(x[train_indexler],y[train_indexler])
    score=mlpc.score(x[test_indexler],y[test_indexler])
    scores.append(score)
    print(score)

print("ortalama score:",np.mean(scores))
```

```
0.5606407322654462
0.6077981651376146
0.5894495412844036
ortalama score: 0.5859628128958215
```

Kfold için ikinci denem K=10 olacak şekilde yapıldı. MLP hiperparametreleri olarak alpha=0.01, solver olarak lbfgs, aktivasyon fonk olarak relu, gizli katamn sayısı = (10,10,10) olacak şekilde 3 kataman, max iterasyon sayısı olarak ise 2000 seçildi. Bölünen parçalarda herhangi sıkıntı görülmedi. Ortalama skor olarak 0.63 hesaplandı.

```
#Kfold cross validation ile veri seti analizi. K=10
from sklearn.model_selection import KFold

scores=[]

kf=KFold(n_splits=10,random_state=1)
mlpc=MLPClassifier(alpha=0.01,solver="lbfgs",activation="relu",hidden_layer_sizes=(10,10,10),max_iter=200)

for train_indexler,test_indexler in kf.split(x):
    mlpc.fit(x[train_indexler],y[train_indexler])
    score=mlpc.score(x[test_indexler],y[test_indexler])
    scores.append(score)
    print(score)

print("ortalama score:",np.mean(scores))
```

```
0.7404580152671756
0.6259541984732825
0.7099236641221374
0.6717557251908397
0.7786259541984732
0.549618320610687
0.6641221374045801
0.6412213740458015
0.5648854961832062
0.4230769230769231
ortalama score: 0.6369641808573107
```

Grid Search Cross Validation

En iyi hiperparametre setinin denenerek seçilmesi için Gridsearch cv kullanıldı. Cv=3 olacak şekilde yapılan validasyon işleminde alpha, gizli katman boyutu, solver, max iterasyon sayısı gibi parametrelere denenmesi için farklı değerler verildi. Tüm bu denemeler veri seti multiclass(3 output – 0,1,2) olduğundan aktivasyon fonk = relu ile yapıldı.

```
#grid search cv ile en iyi parametrelerin belirlenmesi cv=3
from sklearn.model_selection import GridSearchCV

mlpc_params={"alpha":[0.1,0.01,0.001,0.0001],
             "hidden_layer_sizes":[(10,10,10),(3,3,3),(100,100)],
             "solver":["lbfgs","adam","sgd"],
             "max_iter":[20,200,2000]
             }

mlpc=MLPClassifier(activation="relu",random_state=1)

mlpc_cv_model=GridSearchCV(mlpc,mlpc_params,cv=3,n_jobs=-1,verbose=2).fit(x_train,y_train)
```

```
Fitting 3 folds for each of 108 candidates, totalling 324 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    6.7s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   51.2s
[Parallel(n_jobs=-1)]: Done 324 out of 324 | elapsed:  3.6min finished
C:\Users\ABBSCBN\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:841: Deprecad` parameter will change from True to False in version 0.22 and will be removed in 0.24. Thitest-set sizes are unequal.
  DeprecationWarning)
```

Model için en iyi parametreler aşağıdaki gibi görüldü.

```
#cv=3 için en iyi parametrelerin gösterimi
print(mlpc_cv_model.best_params_)
```

```
{'alpha': 0.0001, 'hidden_layer_sizes': (10, 10, 10), 'max_iter': 200, 'solver': 'lbfgs'}
```

Belirlenen parametrelerle yapılan MLP eğitimi zamanı 0.67'lik skor görüldü.

```
#belirlenen parametrelerle model sonucu
mlpc1=MLPClassifier(alpha=0.0001,hidden_layer_sizes=(10,10,10),solver="lbfgs",max_iter=200,random_state=1)
mlpc1.fit(x_train,y_train)
print("score:",mlpc1.score(x_test,y_test))
```

```
score: 0.6717557251908397
```

İyileşme sadece acc değerinde değil F1 skoru ve Karışıklık matrisi için de söz konusu. Karışıklık matrisinde 0 ve 2 değerinin tahmin edilmesinde modelin normal olduğu söylenebilir, fakat 0 ve 1 ve 0 ve 2 değerlerini karıştırdığı matrisden görülmektedir.

```
#Modelin yeni parametrelerle değerlendirilmesi - Accuracy, F1 skoru, Karışıklık Matrisi
import sklearn.metrics as metrics

y_pred=mlpc1.predict(x_test)
acc=metrics.accuracy_score(y_test,y_pred)
print("Accuracy:",acc)
cm=metrics.confusion_matrix(y_test,y_pred)
print(cm)
cr=metrics.classification_report(y_test,y_pred)
print(cr)
```

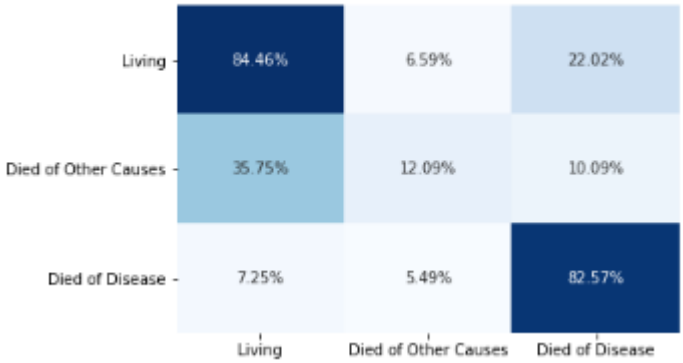
```
Accuracy: 0.6717557251908397
[[163   6  24]
 [ 69  11  11]
 [ 14   5  90]]
              precision    recall  f1-score   support

           0       0.66      0.84      0.74       193
           1       0.50      0.12      0.19        91
           2       0.72      0.83      0.77       109

   micro avg       0.67      0.67      0.67       393
   macro avg       0.63      0.60      0.57       393
weighted avg       0.64      0.67      0.62       393
```

Seaborn kütüphanesiyle cm çizimi

Karışıklık matrisinin görselleştirilmesi için seaborn kütüphanesi import edildi. Matris satır ve sütunları Living, Died of other causes, Died of Disease olacak şekilde isimlendirildi.

```
#Seaborn kütüphanesi tanımlanması
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
import seaborn as sns
class_adlari=["Living","Died of Other Causes","Died of Disease"]
sns.heatmap(cm/cm.sum(axis=1), annot=True, cbar=False, cmap="Blues",xticklabels=class_adlari,yticklabels=class_adlari,fmt=".2%")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x149fae56eb8>
```



Keras ile modelin eğitilmesi ve değerlendirilmesi

Öncelikle keras kütüphanesinde Sequential ve katmanlar import edildi. Optimizer olarak ise Stochastic Gradient Descent (SGD) import edildi. Model eğitim ve test şeklinde bölündü (test_size=0.2). Girilen veri boyuutu kontrol edildi.

```
#Keras kütüphaneleri import edilmesi
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization
import matplotlib.pyplot as plt
from keras.optimizers import SGD
```

```
#modelin train - test çin ayrılması
from sklearn.model_selection import train_test_split
x_train, x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=1)
```

```
#Eğitilen veri boyutunun gösterilmesi
input_shape=x_train.shape[1:]
print(input_shape)
```

```
(8,)
```

Dense ile 16 nörondan oluşan katman oluşturuldu. Aktivasyon fonksiyonu olarak genelde relu son olarak ise softmax seçilmiştir. Katmanların nöron sayıları sırayla artırıldı. Her katmana veri sayısını belirlenen oranda azalması için Dropout eklendi. Optimizer olarak eklenen SGD içerisinde lr(öğrenme hızı)=0.01, ilgili yönde gradyan inişini hızlandırması için momentum=0.9, güncellemesi için decay, Nesterov momentumu eklendi.

```
#Model katmanlarının eğitilmesi
model = Sequential()
model.add(Dense(16, activation='relu', input_dim=8))
model.add(Dropout(0.25))

model.add(Dense(32, activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(64, activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(256, activation='relu'))
model.add(Dropout(0.25))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.add(Dense(64, activation='relu'))

model.add(Dense(10, activation='softmax'))
```

Modelin compile edilmesi optimizer=sgd, loss=sparse_categorical_crossentropy, metril=acc ile yapıldı. Veri seti çok büyük olmadığından adam kullanılmadı.

```
#modelin compile edilmesi
model.compile(optimizer="sgd",loss="sparse_categorical_crossentropy",metrics=["accuracy"])
```

Daha sonra model 1000 iterasyonda fit edildi. Hızlı işlem için batch_size=128 seçildi. Batch_size değerinin çok yüksek olması hızlı işlem açısından iyidir fakat loss değerini artırmaktadır.

```
#modelin fit edilmesi iterasyon sayı=1000
egitim=model.fit(x_train,y_train,epochs=1000,batch_size=128,validation_data=(x_test,y_test))
```

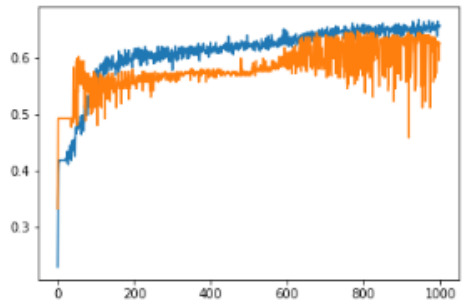
```
Train on 1047 samples, validate on 262 samples
Epoch 1/1000
1047/1047 [==============================] - 0s - loss: 2.2706 - acc: 0.2283 - val_loss: 2.2435 - val_acc: 0.3321
Epoch 2/1000
1047/1047 [==============================] - 0s - loss: 2.2225 - acc: 0.3448 - val_loss: 2.1918 - val_acc: 0.4924
Epoch 3/1000
```

1000 iterasyonun ardından loss değeri 2.2706'dan 0.80'lere indi, acc ise 0.22'den 0.65' e yükseldi.

```
Epoch 999/1000
1047/1047 [==============================] - 0s - loss: 0.8133 - acc: 0.6543 - val_loss: 0.8610 - val_acc: 0.6260
Epoch 1000/1000
1047/1047 [==============================] - 0s - loss: 0.8085 - acc: 0.6581 - val_loss: 0.8516 - val_acc: 0.6260
```

Acc ve val_acc değerlerinin değişim grafiği çizdirildi.

```
#model acc ve va_acc değerlerinin artış grafiği
plt.plot(egitim.history["acc"])
plt.plot(egitim.history["val_acc"])
plt.show()
```



Deneme amaçlı model katmanları tekrardan oluşturuldu. Bu sefer lr 0.001' e düşürüldü. Optimizer olarak ise adam seçildi.

```
#Model katmanlarının eğitilmesi
model = Sequential()
model.add(Dense(16, activation='relu', input_dim=8))
model.add(Dropout(0.25))

model.add(Dense(32, activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(64, activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(256, activation='relu'))
model.add(Dropout(0.25))

sgd = SGD(lr=0.001, decay=1e-6, momentum=0.9, nesterov=True)
model.add(Dense(64, activation='relu'))

model.add(Dense(10, activation='softmax'))

#modelin compile edilmesi
model.compile(optimizer="adam",loss="sparse_categorical_crossentropy",metrics=["accuracy"])
```

```
#modelin compile edilmesi
model.compile(optimizer="sgd",loss="sparse_categorical_crossentropy",metrics=["accuracy"])
```

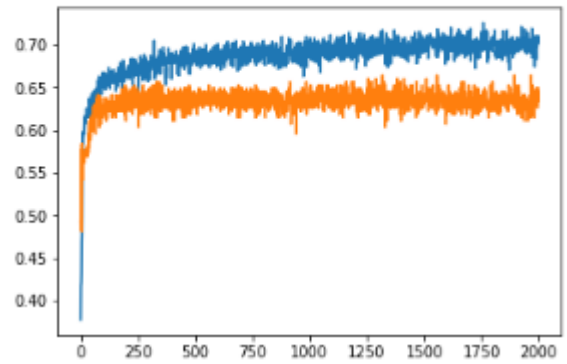
Aynı parametrelerle model tekrar compile edildi ve 2000 iterasyonla fit edildi. Loss değerinin 0.68'e düştüğü, acc değerinin ise 0.70'e kadar yükseldiği görüldü.

```
#modelin fit edilmesi iterasyon sayı=2000
egitim=model.fit(x_train,y_train,epochs=2000,batch_size=128,validation_data=(x_test,y_test))
```

```
Epoch 1991/2000
1047/1047 [==============================] - 0s - loss: 0.7270 - acc: 0.6829 - val_loss: 0.8316 - val_acc: 0.6298
Epoch 1992/2000
1047/1047 [==============================] - 0s - loss: 0.7084 - acc: 0.7001 - val_loss: 0.8419 - val_acc: 0.6374
Epoch 1993/2000
1047/1047 [==============================] - 0s - loss: 0.6959 - acc: 0.6991 - val_loss: 0.8053 - val_acc: 0.6489
Epoch 1994/2000
1047/1047 [==============================] - 0s - loss: 0.6887 - acc: 0.7106 - val_loss: 0.7952 - val_acc: 0.6336
Epoch 1995/2000
1047/1047 [==============================] - 0s - loss: 0.6974 - acc: 0.6982 - val_loss: 0.8167 - val_acc: 0.6412
Epoch 1996/2000
1047/1047 [==============================] - 0s - loss: 0.6751 - acc: 0.7096 - val_loss: 0.8157 - val_acc: 0.6260
Epoch 1997/2000
1047/1047 [==============================] - 0s - loss: 0.6871 - acc: 0.7001 - val_loss: 0.8083 - val_acc: 0.6298
Epoch 1998/2000
1047/1047 [==============================] - 0s - loss: 0.6824 - acc: 0.7039 - val_loss: 0.8066 - val_acc: 0.6450
Epoch 1999/2000
1047/1047 [==============================] - 0s - loss: 0.6862 - acc: 0.7106 - val_loss: 0.7885 - val_acc: 0.6336
Epoch 2000/2000
1047/1047 [==============================] - 0s - loss: 0.6838 - acc: 0.7039 - val_loss: 0.7941 - val_acc: 0.6489
```

Eğitimin acc ve val_acc değerlerindeki değişim aşağıdaki gibi elde edildi.

```
#model acc ve va_acc değerlerinin artış grafiği
plt.plot(egitim.history["acc"])
plt.plot(egitim.history["val_acc"])
plt.show()
```



Son olarak aşağıdaki gibi tahmin değerlerine göre karışıklık matrisi çizdirildi.

```
import sklearn.metrics as metrics
y_test=y_test.reshape(-1,1)

y_pred=model.predict(x_test)
y_pred=np.argmax(y_pred,axis=1)

y_true=np.argmax(y_test,axis=1)
cm=metrics.confusion_matrix(y_pred,y_true)
print(cm)
```

```
[[159   0   0]
 [ 15   0   0]
 [ 88   0   0]]
```

Colab ile MLP ve Keras uygulaması

Google colab kütüphanesinden drive import edildi. Eşleşme kodu girilerek bağlantı sağlandı. Gerekli keras ve katman oluşturma kütüphaneleri, pandas ve numpy kütüphaneleri import edildi. Drive'de bulunan excel dosyası eğitim için Colab'a yüklendi [4].

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!ls drive/My\ Drive/colab
```

```
ysa
```

```
from keras.models import Sequential
from keras.layers import Dense, Activation,Dropout
from keras.layers import BatchNormalization
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
df=pd.read_csv("drive/My Drive/colab/ysa/newbcm.csv",sep=";")
```

Df.head ile veri setine bakıldıktan sonra gereksiz olduğu düşünülen Unnamed:0 sütunu drop edildi.

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

```
   Unnamed: 0  Neoplasm Histologic Grade  ...  Tumor Stage  vital_status
0           1                        3.0  ...          1.0             0
1           2                        2.0  ...          2.0             2
2           3                        2.0  ...          2.0             0
3           4                        3.0  ...          2.0             2
4           5                        3.0  ...          4.0             2

[5 rows x 10 columns]
```

```
df.drop(["Unnamed: 0"],axis=1,inplace=True)
print(df.head())
```

```
   Neoplasm Histologic Grade  ...  vital_status
0                        3.0  ...             0
1                        2.0  ...             2
2                        2.0  ...             0
3                        3.0  ...             2
4                        3.0  ...             2

[5 rows x 9 columns]
```

Veri setine df.info ile genel bakış yapıldıktan sonra Normalizasyon işlemi yapıldı. Normalizasyon sonrası parametre eklenmeden MLP eğitimi yapıldı. Acc, F1 skoru ve Karışıklık Matrisi yöntemleriyle model değerlendirildi. Skorlar aşağıdaki gibidir.

```
#Parametre eklenmeden yapılan MLP eğitimi
from sklearn.neural_network import MLPClassifier

mlpc=MLPClassifier(random_state=1)
mlpc.fit(x_train,y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_percep
  % self.max_iter, ConvergenceWarning)
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100,), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=200,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=1, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
```

```
#Genel değerlendirilmesi - Accuracy, F1 skoru, Karışıklık Matrisi
import sklearn.metrics as metrics

y_pred=mlpc.predict(x_test)
acc=metrics.accuracy_score(y_test,y_pred)
print("Accuracy:",acc)
cm=metrics.confusion_matrix(y_test,y_pred)
print(cm)
cr=metrics.classification_report(y_test,y_pred)
```

```
Accuracy: 0.5877862595419847
[[146   0  47]
 [ 62   0  29]
 [ 24   0  85]]
```

Grid search ile cv=3 değerinde en iyi hiperparametre denendi.

```
#grid search cv ile en iyi parametrelerin belirlenmesi cv=3
from sklearn.model_selection import GridSearchCV

mlpc_params={"alpha":[0.1,0.01,0.001,0.0001],
             "hidden_layer_sizes":[(10,10,10),(3,3,3),(100,100)],
             "solver":["lbfgs","adam","sgd"],
}

mlpc=MLPClassifier(activation="relu",random_state=1)

mlpc_cv_model=GridSearchCV(mlpc,mlpc_params,cv=3,n_jobs=-1,verbose=2).fit(x_train,y_train)
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   13.2s
[Parallel(n_jobs=-1)]: Done 108 out of 108 | elapsed:   42.2s finished
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:470:
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Belirlenen parametreler ekranda yazdırıldı. Bu parametrelerle max_iter=200 değerinde 0.66 skoru elde edildi.

```
#cv=3 için en iyi parametrelerin gösterimi
print(mlpc_cv_model.best_params_)
```

```
{'alpha': 0.1, 'hidden_layer_sizes': (100, 100), 'solver': 'lbfgs'}
```

```
#belirlenen parametrelerle model sonucu
mlpc1=MLPClassifier(alpha=0.1,hidden_layer_sizes=(100,100),solver="lbfgs",max_iter=200,random_state=1)
mlpc1.fit(x_train,y_train)
print("score:",mlpc1.score(x_test,y_test))
```

```
score: 0.6615776081424937
```

Keras

Keras eğitimi için aşağıdaki gibi katmanlar oluşturuldu. Katmanlardaki nöron sayısı sırayla artırıldı. Multi class olduğu için aktivasyon fonksiyonu olarak relu seçildi. İnput_dim olarak x değeri boyutuna uygun olarak 8 seçildi. Her katman sonrası 0.25 lik Dropout (sayıyı azaltmak için) kullanıldı. Son olarak softmax kullanıldı. Model compile edildi. 1000 iterasyonda batch_size=16 (yavaş) şeklinde fit edildi.

```
#Model katmanlarının eğitilmesi
model = Sequential()
model.add(Dense(16, activation='relu', input_dim=8))
model.add(Dropout(0.25))

model.add(Dense(32, activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(64, activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(256, activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(64, activation='relu'))

model.add(Dense(10, activation='softmax'))
```

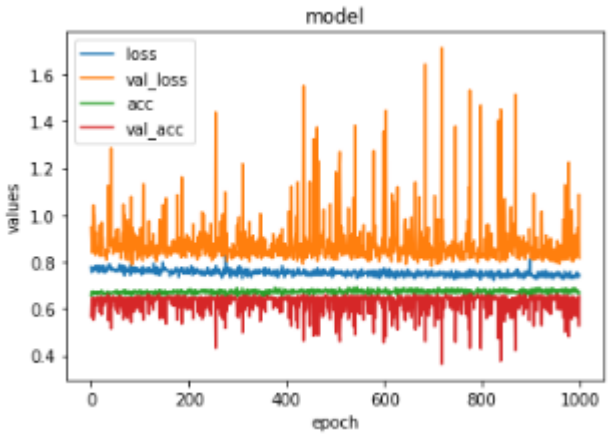
```
#modelin compile edilmesi
model.compile(optimizer="sgd",loss="sparse_categorical_crossentropy",metrics=["accuracy"])
```

```
#modelin fit edilmesi iterasyon sayı=1000
egitim=model.fit(x_train,y_train,epochs=1000,batch_size=16,validation_data=(x_test,y_test))
```

Fit sonrası model gelişiminin görselliştirilmesi amacıyla aşağıdaki kodlar kullanıldı.

```
plt.plot(egitim.history["loss"])
plt.plot(egitim.history["val_loss"])
plt.plot(egitim.history["accuracy"])
plt.plot(egitim.history["val_accuracy"])
plt.title('model')
plt.ylabel('values')
plt.xlabel('epoch')
plt.legend(['loss','val_loss','acc','val_acc'],loc='lower_right')
plt.show()

import sklearn.metrics as metrics
import seaborn as sns
Y_pred=model.predict(x_test)
Y_pred_classes=np.argmax(Y_pred,axis=1)
Y_true=np.argmax(y_test,axis=1)
cm=metrics.confusion_matrix(Y_true,Y_pred_classes)

cr=metrics.classification_report(Y_true,Y_pred_classes)
print(cr)
acc=metrics.accuracy_score(Y_true,Y_pred_classes)
print(acc)
```

1000 iterasyon sonrası aşağıdaki gibi grafik elde edildi.



KAYNAKÇA

[1] https://www.who.int/en/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds)

[2] https://www.kaggle.com/dileep070/heart-disease-prediction-using-logistic-regression

[3] https://www.kaggle.com/gunesevitan/breast-cancer-metabric

[4]https://colab.research.google.com/drive/1QNYkktl6Y_k7xrwnhbRFJLUiZg6xlqQs?usp=sharing