

**KASTAMONU ÜNİVERSİTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ MAKİNE ÖĞRENMESİ DERSİ ÖDEV RAPORU**

**ÖDEV**

**Makine Öğrenmesi Uygulamaları**

**ÖDEV TARİHİ**

**18.01.2021**

**DERSİN SORUMLUSU**

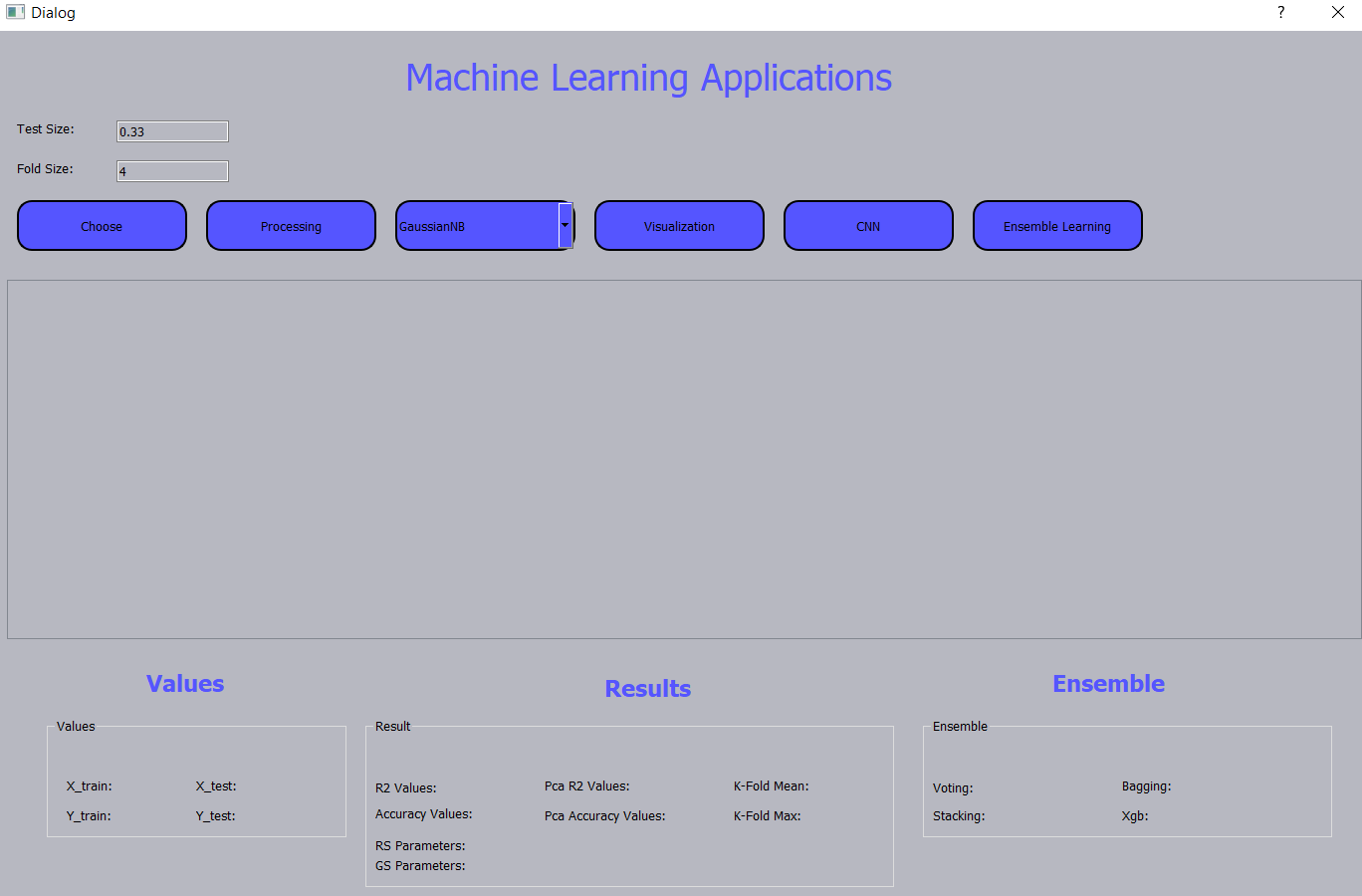
**Kemal AKYOL**

**RAPORU YAZAN ÖĞRENCİ**

**174410037**

**Mustafa Said ÇELİK**

**A)Ekran Görüntüleri**



**Arayüz açıklaması:**

-Test Size alanına test verilerinin yüzdelik olarak ne kadar olacağı belirlenir.

-Fold Size alanına K-fold’un fold sayısı belirlenir.

-Choose butonu ile ilgili veriseti seçilir ve tabloya dolar. Burada seçilen veriseti bir telekomünikasyon şirketinin her müşterisi için çeşitli bilglerini içerir. Sonuç olarak müşteri kaybı oluşup oluşmadığı bilgisi vardır.

-Processing butonu ile ilgili algoritmayla işlem yapılır ve sonuçlar arayüze gelir.

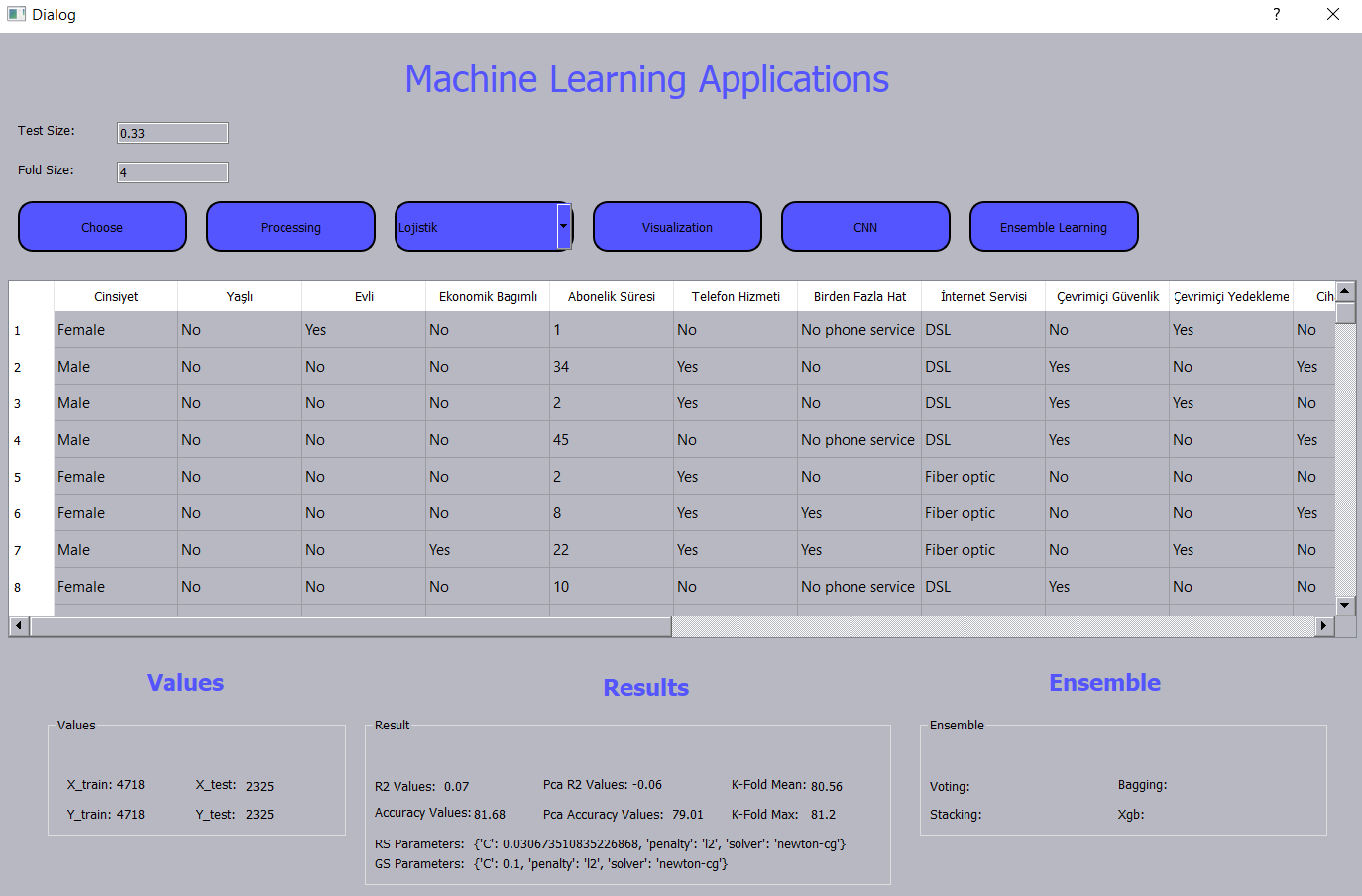
-ComboBox ile ilgili algoritma seçilir.

-Visualization butonu, veri ve algoritmalar ile ilgili grafikleri gösterir.

-Cnn butonu ile YSA algoritması işlemi gerçekleşir.

-Ensemble butonu ile toplu öğrenme algoritmalarının sonuçları gelir.

**1) Processing Butonuna Basıldığında Yürtülen İşlem**

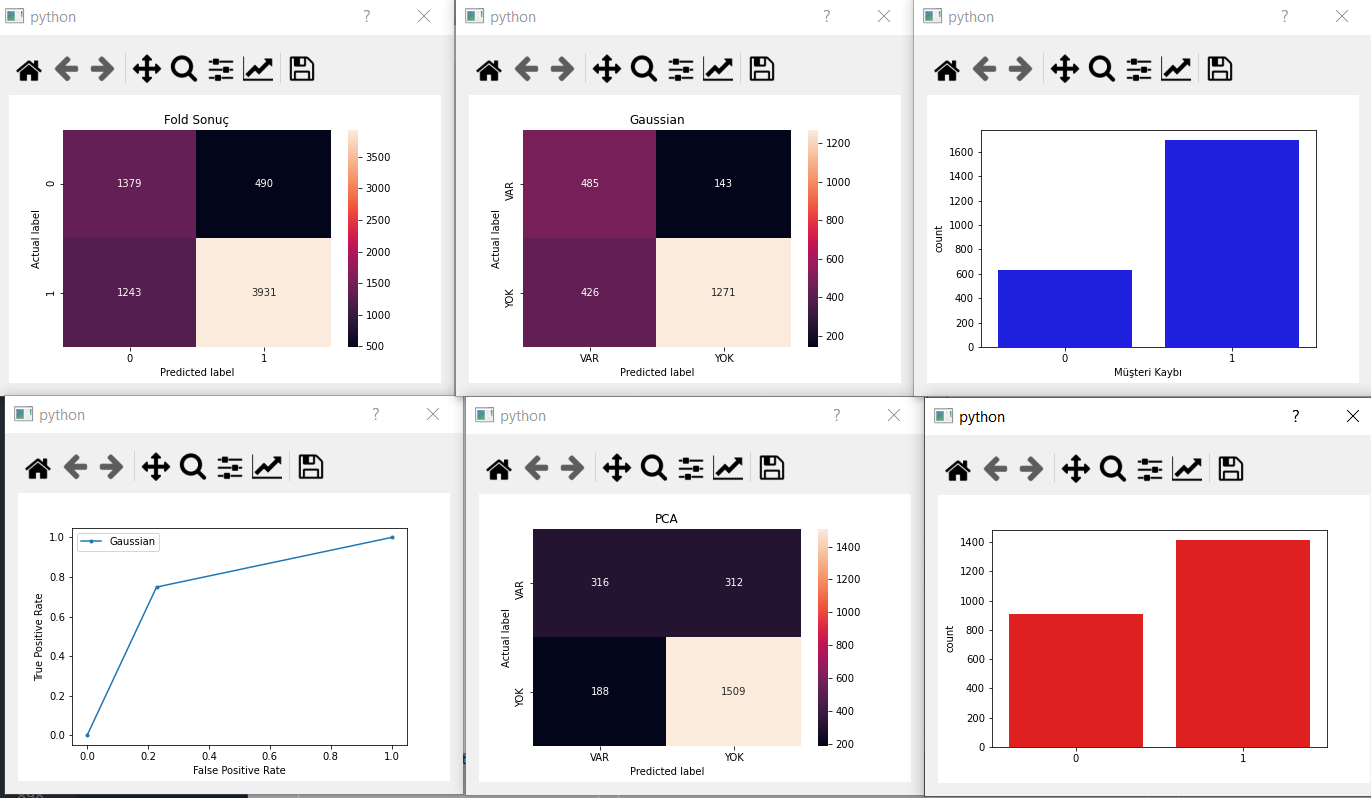


-Örnek olarak Logistic regression algoritması çalıştırıldığında verilen test size değerine göre test ve trainler ayrılır ve Values alanında gösterilir.

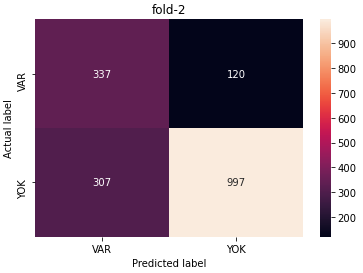
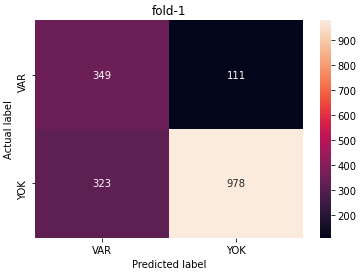
-Result alanında hold-out ile test size değerine göre yapılan işlem sonucu çıkan Logistic regression accuracy ve r2 kare değeri, pca(5) accuracy ve r2 değeri, fold size değerine göre K-fold ortalama ve maximum değeri, RandomizedSearchCV ve GridSearchCV sonucunda çıkan parametreleri gösterilmiştir.

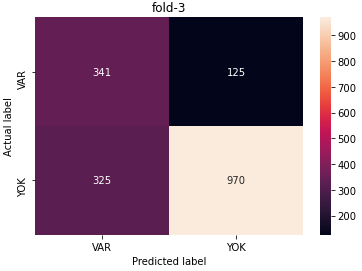
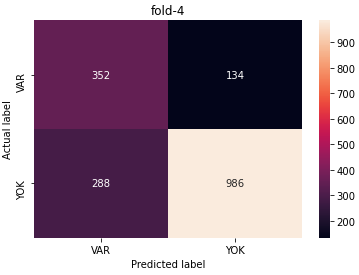
-Bu işlemlerin bir arada yapılmasının sebebi yapılan bu uygulama makine öğrenmesinin yanında bir analiz uygulaması olmasıdır. Her algoritma için bütün işlemlerin sonuçları gösterilir. Sonuçlara bakılarak hangi işlemin sonucunun daha başarılı olduğu anlaşılmaktadır.

-Örnek olarak yukarıda gösterilen Logistic regression algoritmasında Hold-out sonucunda işlemle alınan algoritma sonucu en başarılı accuracy değerini vermiştir.

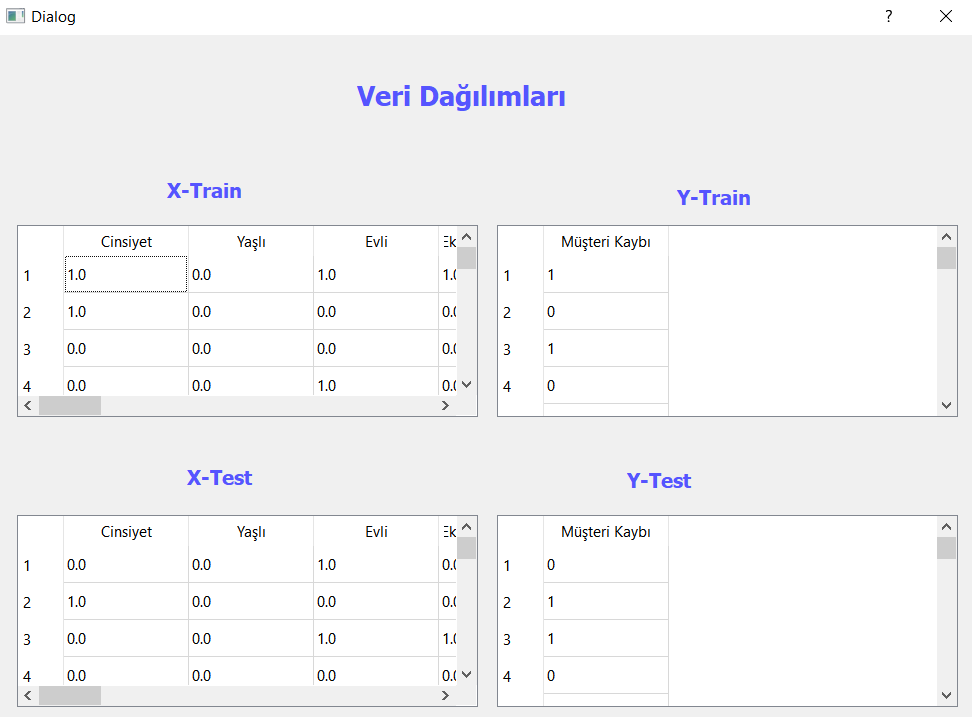


-İşlem sonucunda grafikler Overlapped matrix, hold-out sonucunda oluşan confusion matrix ve roc eğrisi, Logistic regression algoritmasına pca(5) sonucunda oluşan confusion matrixi, hold-out sonucunda Logistic regression ile tahmin ve gerçek verilerin grafiğe dökümü gözükmektedir. Fonkisyonlar hazır bulunmaktadır, k-fold sonucunda çıkan overlapped matix ve pca sonucnda çıkan confusion matrix içinde roc eğrisi ve grafik gösterimleri tek satır kod ile yapılabilinirdi ama çok fazla görsel olduğu için bu kadar yapılmıştır.

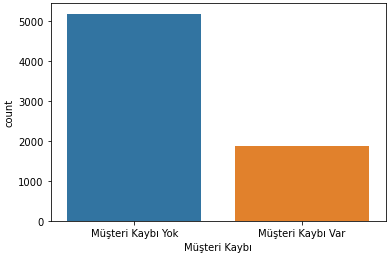
 

-Fold size değerine göre çıkan foldların confusion matrix grafikleri gözükmektedir.

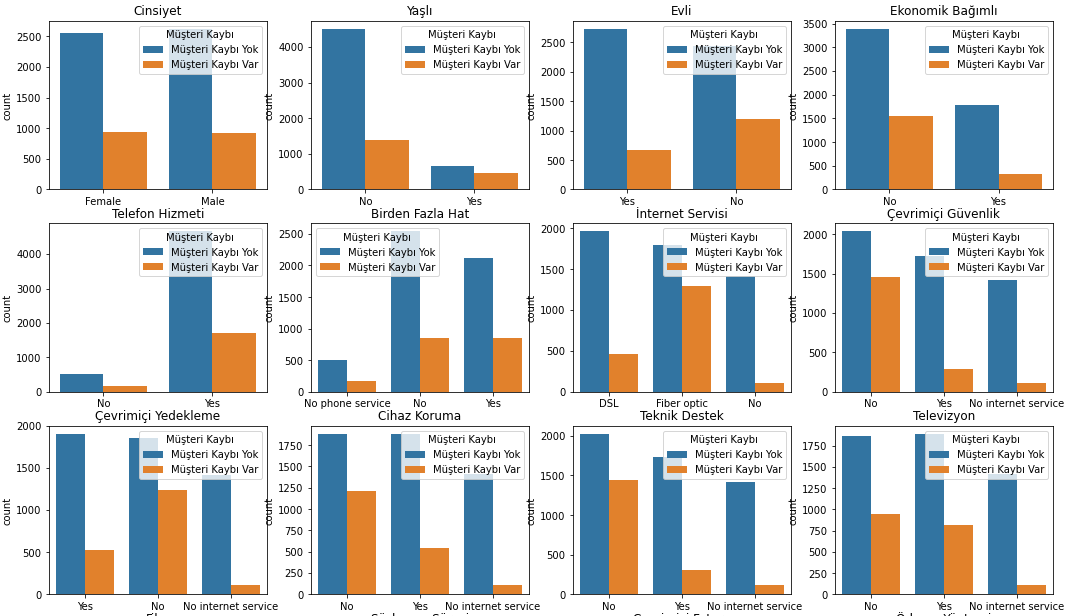


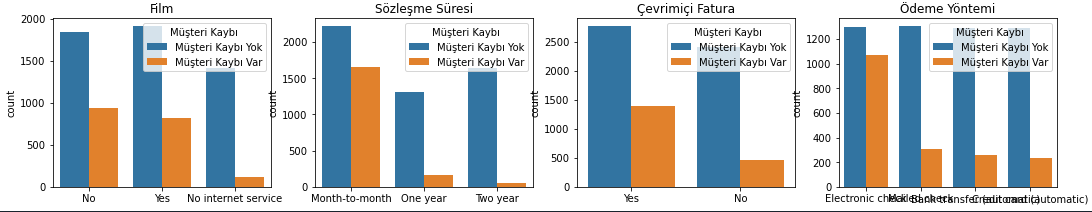
-Son olarak hold-out sonucunda test size değerine göre çekilen veriler gösterilir.

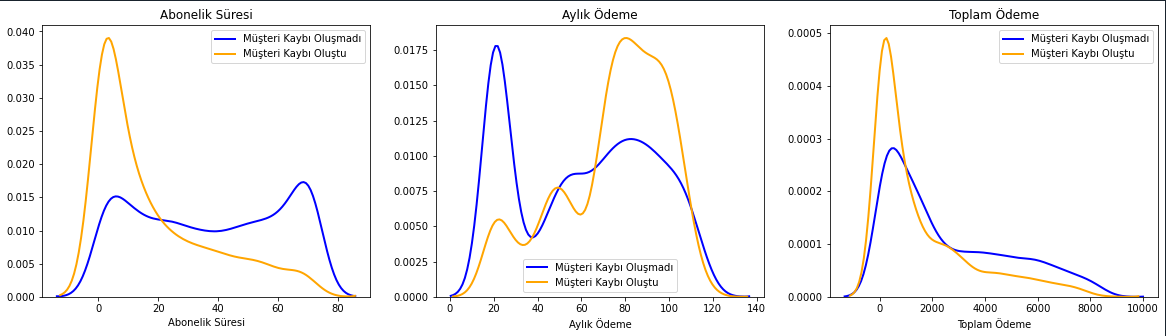
**2) Visualization Butonuna Basıldığında Yürtülen İşlem**



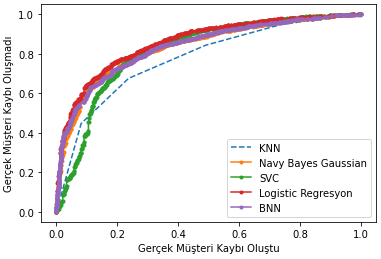
-Müşteri kaybının grafiğe dökümü gösterilmiştir.

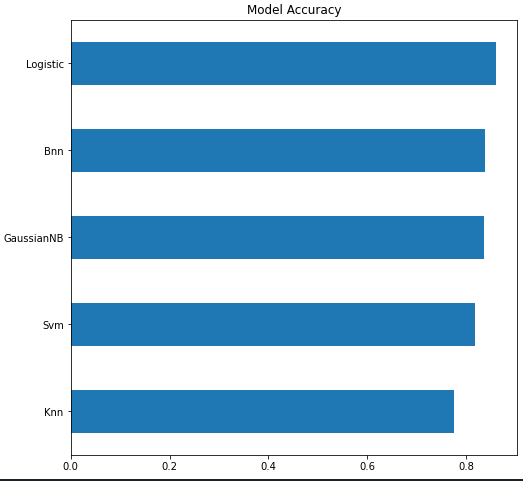






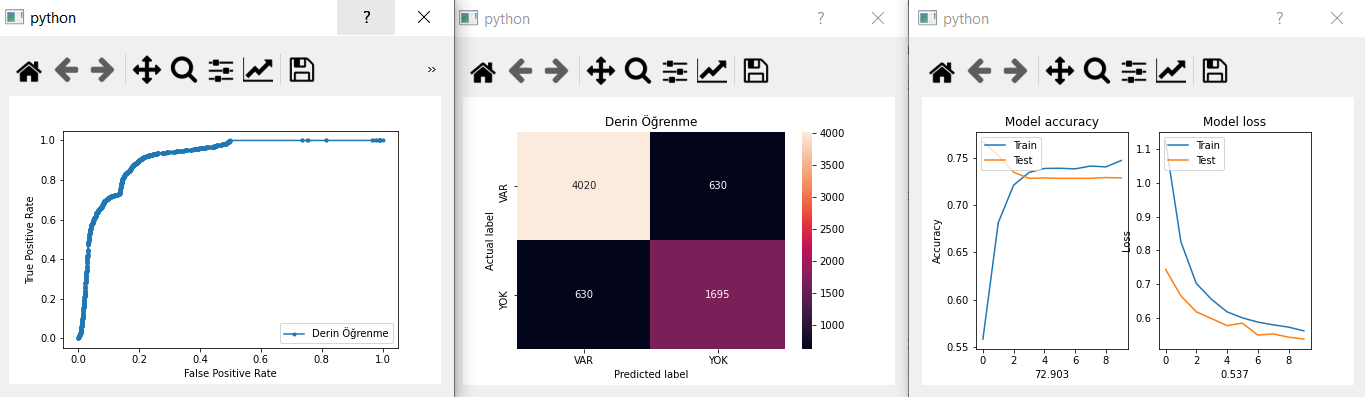
-Bütün özniteliklerin müşteri kaybı ile bağlantısının grafiğe dökümü gösterilmiştir.





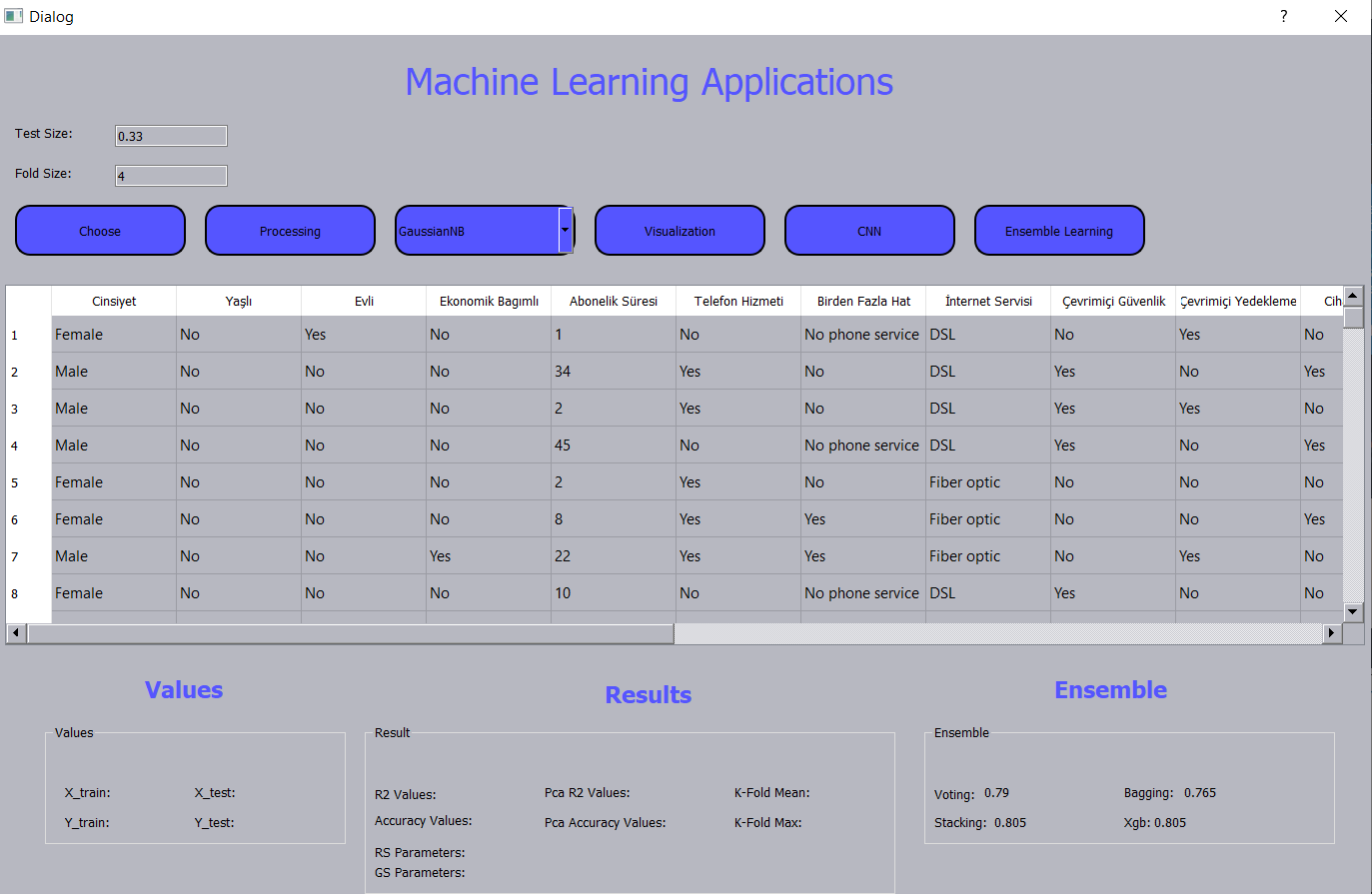
-Bütün algoritmaların roc eğrisi gösterimi ve başarı analizi yapılmıştır.

**3) Cnn Butonuna Basıldığında Yürtülen İşlem**

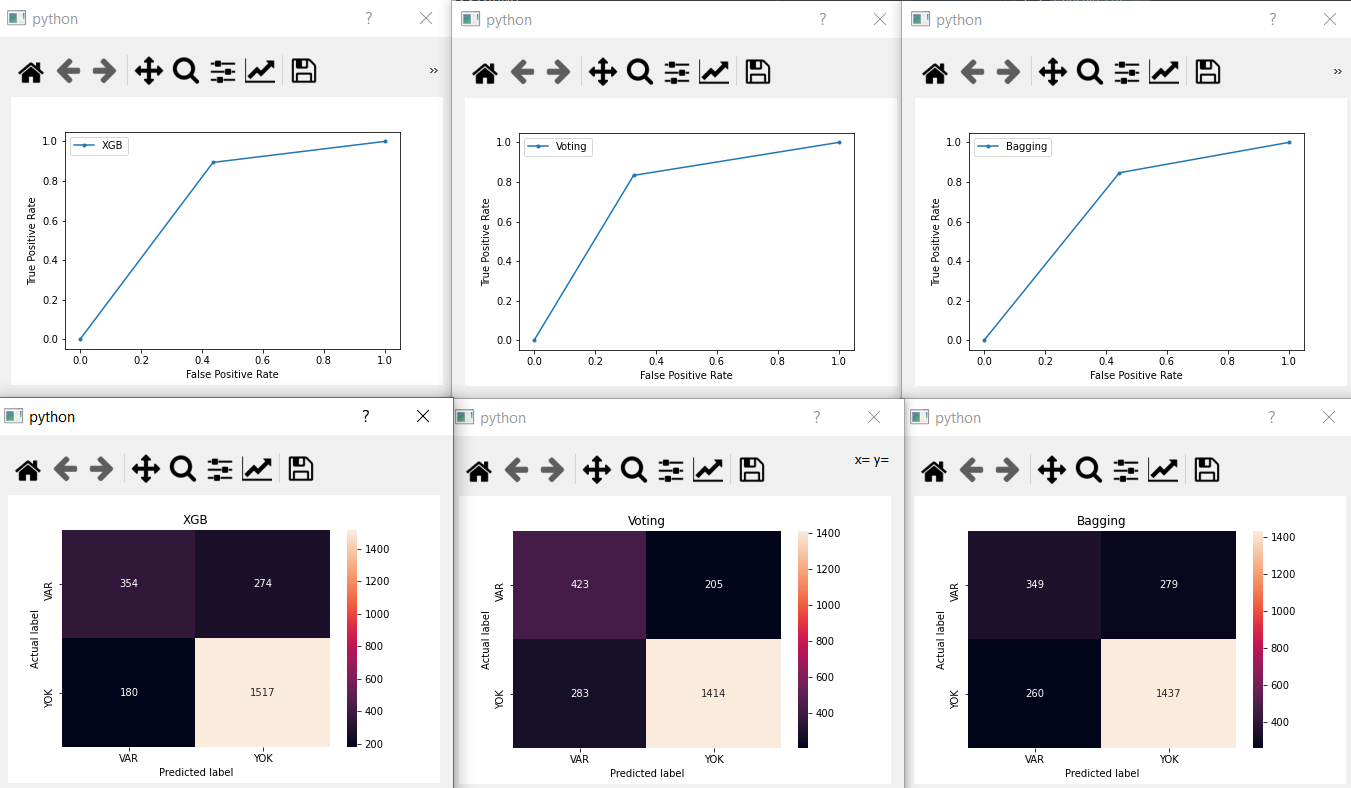


-YSA algoritmasıyla Roc eğrisi, Confusion matrix, accuracy ve loss grafikleri gösterilmiştir.

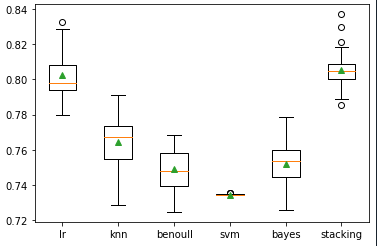
**4) Ensemble Butonuna Basıldığında Yürtülen İşlem**



-İlgili Label’a ilgili sonuçlar yazılır ve grafikler gösterilir.



-Xgb, Voiting, Bagging algoritmalarının confusion matrix ve roc grafikleri gösterilmiştir.



-Stacking algoritması sonucu grafiğe dökümü gösterilmiştir.

**B) KODLAR VE AÇIKLAMALARI**

**1)Ön Hazırlık Kodları**

**def** labeEnc**(**self**):**

**from** sklearn**.**preprocessing **import** LabelEncoder

self**.**encoded **=** self**.**veriler**.**apply**(lambda** x**:** LabelEncoder**().**fit\_transform**(**x**)** **if** x**.**dtype **==** 'object' **else** x**)**

**def** aykiriVeri**(**self**):**

Müşteri\_Kaybı\_Yaşandı**=**self**.**encoded**.**loc**[**self**.**encoded**[**'Müşteri Kaybı'**].**abs**()>**0**]**

# print(Müşteri\_Kaybı\_Yaşandı)

Q1 **=** Müşteri\_Kaybı\_Yaşandı**[**'Toplam Ödeme'**].**quantile**(**0.25**)**

Q3 **=** Müşteri\_Kaybı\_Yaşandı**[**'Toplam Ödeme'**].**quantile**(**0.75**)**

IQR **=** Q3 **-** Q1

Q**=**Q3**+(**1.5**\***IQR**)**

encoded\_out **=** self**.**encoded**[~((**self**.**encoded**[**'Toplam Ödeme'**]** **<** **(**Q3 **+** 1.5 **\*** IQR**)))&(**self**.**encoded**[**'Müşteri Kaybı'**]>**0**)]**

# print(encoded\_out.head(8000))

# Aykırı veriler çağırıldı.Grafikte gözlemlediğimiz yoğunluktaki aykırı veri sayısı 109'muş.

self**.**encoded**.**drop**(**self**.**encoded**[~((**self**.**encoded**[**'Toplam Ödeme'**]** **<** **(**Q3 **+** 1.5 **\*** IQR**)))&(**self**.**encoded**[**'Müşteri Kaybı'**]>**0**)].**index**,** inplace**=True)**

# print(self.encoded.head(8000))

Q1\_A **=** Müşteri\_Kaybı\_Yaşandı**[**'Abonelik Süresi'**].**quantile**(**0.25**)**

Q3\_A **=** Müşteri\_Kaybı\_Yaşandı**[**'Abonelik Süresi'**].**quantile**(**0.75**)**

IQR\_A **=** Q3\_A **-** Q1\_A

# print( IQR\_A)

Q\_A**=**Q3\_A**+(**1.5**\***IQR\_A**)**

# print(Q\_A)

encoded\_A\_out **=** self**.**encoded**[~((**self**.**encoded**[**'Abonelik Süresi'**]** **<** **(**Q3\_A **+** 1.5 **\*** IQR\_A**)))&(**self**.**encoded**[**'Müşteri Kaybı'**]>**0**)]**

# print(encoded\_A\_out.head(8000))

self**.**encoded**.**drop**(**self**.**encoded**[~((**self**.**encoded**[**'Abonelik Süresi'**]** **<** **(**Q3\_A **+** 1.5 **\*** IQR\_A**)))&(**self**.**encoded**[**'Müşteri Kaybı'**]>**0**)].**index**,** inplace**=True)**

# # print(self.encoded.head(8000))

**def** onHazirlik**(**self**):**

self**.**labeEnc**()**

self**.**aykiriVeri**()**

x **=** self**.**encoded**.**drop**(**'Müşteri Kaybı'**,** axis **=** 1**)**

y **=** self**.**encoded**[**'Müşteri Kaybı'**]**

**return** x**,**y

-Her algoritma başında onHazırlik fonksiyonu çağırlır. LabelEncoder işlemi yapılır. Aykırı veriler tamizlenir ve artık veriler kullanıma hazır durma getirilir.

**2)Choose Butonuna Basıldığında Yürütülen Kodlar**

**def** csvYukle**(**self**,**veridoldur**):**

c**=**len**(**veridoldur**.**columns**)**

r**=**len**(**veridoldur**.**values**)**

self**.**tableWidget**.**setColumnCount**(**c**)**

self**.**tableWidget**.**setRowCount**(**r**)**

colmnames**=[**"Cinsiyet"**,**"Yaşlı"**,**"Evli"**,**"Ekonomik Bagımlı"**,**"Abonelik Süresi"**,**"Telefon Hizmeti"**,**"Birden Fazla Hat"**,**

"İnternet Servisi"**,**"Çevrimiçi Güvenlik"**,**"Çevrimiçi Yedekleme"**,**"Cihaz Koruma"**,**"Teknik Destek"**,**"Televizyon"**,**

"Film"**,**"Sözleşme Süresi"**,**"Çevrimiçi Fatura"**,**"Ödeme Yöntemi"**,**"Aylık Ödeme"**,**"Toplam Ödeme"**,**"Müşteri Kaybı"**]**

self**.**tableWidget**.**setHorizontalHeaderLabels**(**colmnames**)**

**for** i**,**row **in** enumerate**(**veridoldur**):**

**for** j**,**cell **in** enumerate**(**veridoldur**.**values**):**

self**.**tableWidget**.**setItem**(**j**,**i**,** QtWidgets**.**QTableWidgetItem**(**str**(**cell**[**i**])))**

**def** ekleme**(**self**):**

file\_name**,** \_ **=** QFileDialog**.**getOpenFileName**(**self**,** 'Open Image File'**,** r".\Desktop"**,** ".xls(\*.xls) .csv(\*.csv)"**)**

self**.**veriler **=** pd**.**read\_csv**(**file\_name**)**

self**.**veriler**.**rename**(**columns**={**'customerID'**:**'Müşteri ID'**,**'gender'**:**'Cinsiyet'**,**'SeniorCitizen'**:**'Yaşlı'**,**'Partner'**:**'Evli'**,**

'Dependents'**:**'Ekonomik Bağımlı'**,**'tenure'**:**'Abonelik Süresi'**,**'PhoneService'**:**'Telefon Hizmeti'**,**

'MultipleLines'**:**'Birden Fazla Hat'**,**'InternetService'**:**'İnternet Servisi'**,**

'OnlineSecurity'**:**'Çevrimiçi Güvenlik'**,**'OnlineBackup'**:**'Çevrimiçi Yedekleme'**,**

'DeviceProtection'**:**'Cihaz Koruma'**,**'TechSupport'**:**'Teknik Destek'**,**'StreamingTV'**:**'Televizyon'**,**

'StreamingMovies'**:**'Film'**,**'Contract'**:**'Sözleşme Süresi'**,**'PaperlessBilling'**:**'Çevrimiçi Fatura'**,**

'PaymentMethod'**:**'Ödeme Yöntemi'**,**'MonthlyCharges'**:**'Aylık Ödeme'**,**'TotalCharges'**:**'Toplam Ödeme'**,**

'Churn'**:**'Müşteri Kaybı'**},**inplace**=True)**

self**.**veriler**.**drop**(**'Müşteri ID'**,** axis**=**1**,** inplace**=True)**

self**.**veriler**[**"Müşteri Kaybı"**]=** self**.**veriler**[**"Müşteri Kaybı"**].**replace**(**"No"**,**"Müşteri Kaybı Yok"**)**

self**.**veriler**[**"Müşteri Kaybı"**]=** self**.**veriler**[**"Müşteri Kaybı"**].**replace**(**"Yes"**,**"Müşteri Kaybı Var"**)**

self**.**veriler**[**"Yaşlı"**]=** self**.**veriler**[**"Yaşlı"**].**replace**(**0**,** "No"**)**

self**.**veriler**[**"Yaşlı"**]=** self**.**veriler**[**"Yaşlı"**].**replace**(**1**,** "Yes"**)**

# print(self.veriler)

self**.**veriler**[**'Toplam Ödeme'**]** **=** pd**.**to\_numeric**(** self**.**veriler**[**'Toplam Ödeme'**],** errors**=**'coerce'**)**

self**.**veriler**[**'Toplam Ödeme'**]** **=** self**.**veriler**[**'Toplam Ödeme'**].**fillna**(**value**=**0**)**

self**.**veriler**[**'Yaşlı'**]** **=** self**.**veriler**[**'Yaşlı'**].**astype**(**'object'**)**

self**.**csvYukle**(**self**.**veriler**)**

-İlk olarak ekleme fonksiyonu çalışır ve excel dosyası çekilir. Column isimleri türkçeleştirilir ve csvYukle fonksiyonu çağırılır ve orada tabloya yazdırılır.

**3)Proccessing Butonuna Basıldığında Yürütülen Kodlar**

**def** regAlgm**(**self**):**

**from** sklearn**.**model\_selection **import** cross\_val\_score

sec**=**self**.**comboBox**.**currentText**()**

**if** sec**==**'GaussianNB'**:**

x**,**y**=**self**.**onHazirlik**()**

**from** sklearn**.**naive\_bayes **import** GaussianNB

test\_size1**=**float**(**self**.**lineEdit**.**text**())**

fold\_size**=**int**(**self**.**lineEdit\_2**.**text**())**

self**.**foldsizee**=**fold\_size

x\_train**,** x\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**x**,** y**,** test\_size **=** test\_size1**,** random\_state **=** 42**)**

# yenipncre=Windoww2()

self**.**w**.**csvYukle**(**x\_train**,**y\_train**,**x\_test**,**y\_test**)**

# yenipncre.show()

self**.**label\_22**.**setText**(**str**(**len**(**x\_train**)))**

self**.**label\_23**.**setText**(**str**(**len**(**y\_train**)))**

self**.**label\_24**.**setText**(**str**(**len**(**x\_test**)))**

self**.**label\_25**.**setText**(**str**(**len**(**x\_test**)))**

**from** sklearn**.**preprocessing **import** StandardScaler

sc\_X**=**StandardScaler**()**

x\_train**=**sc\_X**.**fit\_transform**(**x\_train**)**

x\_test**=**sc\_X**.**transform**(**x\_test**)**

NBG **=** GaussianNB**()**

NBG**.**fit**(**x\_train**,** y\_train**)**

y\_pred **=** NBG**.**predict**(**x\_test**)**

self**.**pltPredict**(**y\_test**,** y\_pred**)**

self**.**pltTrue**(**y\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**label\_5**.**setText**(**str**(**round**(**acc**,**2**)))**

self**.**label\_4**.**setText**(**str**(**round**(**r2\_score**(**y\_test**,**y\_pred**),**2**)))**

X\_train2**,**X\_test2 **=**self**.**pcaIslem**(**x\_train**,**x\_test**,**5**)**

classifier2 **=** GaussianNB**()**

classifier2**.**fit**(**X\_train2**,**y\_train**)**

y\_pred2**=**classifier2**.**predict**(**X\_test2**)**

self**.**Cmatrix2**(**y\_test**,**y\_pred2**,**"PCA"**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred2**)\***100

self**.**label\_9**.**setText**(**str**(**round**(**acc**,**2**)))**

self**.**label\_8**.**setText**(**str**(**round**(**r2\_score**(**y\_test**,**y\_pred2**),**2**)))**

self**.**Cmatrix**(**y\_test**,**y\_pred**,**"Gaussian"**)**

**print(**confusion\_matrix**(**y\_test**,** y\_pred**))**

self**.**pltRoc2**(**y\_test**,**y\_pred**,**"Gaussian"**)**

**from** sklearn**.**model\_selection **import** KFold

**from** numpy **import** mean

**from** sklearn**.**model\_selection **import** cross\_val\_score

X **=** x**.**values

y **=** y**.**values

kf **=** KFold**(**n\_splits**=**fold\_size**)**

kf**.**get\_n\_splits**(**X**)**

sayma**=**0

**for** train\_index**,** test\_index **in** kf**.**split**(**X**):**

sayma**+=**1

# print("TRAIN:", train\_index, "TEST:", test\_index)

x\_train**,** x\_test **=** X**[**train\_index**],** X**[**test\_index**]**

y\_train**,** y\_test **=** y**[**train\_index**],** y**[**test\_index**]**

NBG **=** GaussianNB**()**

NBG**.**fit**(**x\_train**,** y\_train**)**

y\_pred **=** NBG**.**predict**(**x\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**CmatrixFold**(**y\_test**,**y\_pred**,**"fold-"**+**str**(**sayma**))**

self**.**kfoldCmatrix**(**y\_test**,** y\_pred**,**"Fold Sonuç"**)**

**print(**acc**)**

model **=** GaussianNB**()**

scores **=** cross\_val\_score**(**model**,** X**,** y**,** scoring**=**'accuracy'**,** cv**=**kf**,** n\_jobs**=-**1**)**

self**.**label\_12**.**setText**(**str**(**round**(**mean**(**scores**\***100**),**2**)))**

self**.**label\_13**.**setText**(**str**(**round**(**scores**.**max**()\***100**,**2**)))**

**print(**'Accuracy: %.3f (%.3f)' **%** **(**mean**(**scores**),** scores**.**max**()))**

**print(**"--------------------"**)**

# dogru

# from sklearn.model\_selection import GridSearchCV, RandomizedSearchCV

# from sklearn.pipeline import Pipeline

# pipeline = Pipeline([

# ('clf', GaussianNB())

# ])

# parameters = {

# 'clf\_\_priors': [None],

# 'clf\_\_var\_smoothing': [0.00000001]

# }

# cv = GridSearchCV(pipeline, param\_grid=parameters)

# cv.fit(x\_train, y\_train)

# print(cv.best\_params\_)

###############################

**if** sec**==**"BernoulliNB"**:**

x**,**y**=**self**.**onHazirlik**()**

test\_size1**=**float**(**self**.**lineEdit**.**text**())**

fold\_size**=**int**(**self**.**lineEdit\_2**.**text**())**

self**.**foldsizee**=**fold\_size

x\_train**,** x\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**x**,** y**,** test\_size **=** test\_size1**,** random\_state **=** 42**)**

self**.**w**.**csvYukle**(**x\_train**,**y\_train**,**x\_test**,**y\_test**)**

self**.**label\_22**.**setText**(**str**(**len**(**x\_train**)))**

self**.**label\_23**.**setText**(**str**(**len**(**y\_train**)))**

self**.**label\_24**.**setText**(**str**(**len**(**x\_test**)))**

self**.**label\_25**.**setText**(**str**(**len**(**x\_test**)))**

**from** sklearn**.**preprocessing **import** StandardScaler

sc\_X**=**StandardScaler**()**

x\_train**=**sc\_X**.**fit\_transform**(**x\_train**)**

x\_test**=**sc\_X**.**fit\_transform**(**x\_test**)**

**from** sklearn**.**naive\_bayes **import** BernoulliNB

NBB **=** BernoulliNB**()**

NBB**.**fit**(**x\_train**,** y\_train**)**

y\_pred **=** NBB**.**predict**(**x\_test**)**

self**.**pltPredict**(**y\_test**,** y\_pred**)**

self**.**pltTrue**(**y\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**label\_5**.**setText**(**str**(**round**(**acc**,**2**)))**

self**.**label\_4**.**setText**(**str**(**round**(**r2\_score**(**y\_test**,**y\_pred**),**2**)))**

X\_train2**,**X\_test2 **=**self**.**pcaIslem**(**x\_train**,**x\_test**,**5**)**

classifier2 **=** BernoulliNB**()**

classifier2**.**fit**(**X\_train2**,**y\_train**)**

y\_pred2**=**classifier2**.**predict**(**X\_test2**)**

self**.**Cmatrix2**(**y\_test**,**y\_pred2**,**"PCA"**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred2**)\***100

self**.**label\_9**.**setText**(**str**(**round**(**acc**,**2**)))**

self**.**label\_8**.**setText**(**str**(**round**(**r2\_score**(**y\_test**,**y\_pred2**),**2**)))**

self**.**Cmatrix**(**y\_test**,**y\_pred**,**"BernoulliNB"**)**

**print(**confusion\_matrix**(**y\_test**,** y\_pred**))**

self**.**pltRoc2**(**y\_test**,**y\_pred**,**"BernoulliNB"**)**

**from** sklearn**.**model\_selection **import** KFold

**from** numpy **import** mean

**from** sklearn**.**model\_selection **import** cross\_val\_score

X **=** x**.**values

y **=** y**.**values

kf **=** KFold**(**n\_splits**=**fold\_size**)**

kf**.**get\_n\_splits**(**X**)**

sayma**=**0

**for** train\_index**,** test\_index **in** kf**.**split**(**X**):**

sayma**+=**1

# print("TRAIN:", train\_index, "TEST:", test\_index)

x\_train**,** x\_test **=** X**[**train\_index**],** X**[**test\_index**]**

y\_train**,** y\_test **=** y**[**train\_index**],** y**[**test\_index**]**

NBG **=** BernoulliNB**()**

NBG**.**fit**(**x\_train**,** y\_train**)**

y\_pred **=** NBG**.**predict**(**x\_test**)**

self**.**CmatrixFold**(**y\_test**,**y\_pred**,**"fold-"**+**str**(**sayma**))**

self**.**kfoldCmatrix**(**y\_test**,** y\_pred**,**"Fold Sonuç"**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

**print(**acc**)**

model **=** BernoulliNB**()**

scores **=** cross\_val\_score**(**model**,** X**,** y**,** scoring**=**'accuracy'**,** cv**=**kf**,** n\_jobs**=-**1**)**

self**.**label\_12**.**setText**(**str**(**round**(**mean**(**scores**\***100**),**2**)))**

self**.**label\_13**.**setText**(**str**(**round**(**scores**.**max**()\***100**,**2**)))**

**print(**'Accuracy: %.3f (%.3f)' **%** **(**mean**(**scores**),** scores**.**max**()))**

**print(**"--------------------"**)**

# dogru

# from sklearn.model\_selection import GridSearchCV, RandomizedSearchCV

# from sklearn.pipeline import Pipeline

# from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer

# pipeline = Pipeline(steps=[

# ('bnb', BernoulliNB())

# ])

# parameters = {

# 'bnb\_\_fit\_prior': [False,True],

# 'bnb\_\_alpha': [0.0,0.5,1.0],

# }

# x,y=self.onHazirlik()

# x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.33, random\_state = 42)

# cv = GridSearchCV(pipeline, param\_grid=parameters)

# cv.fit(x, y)

# print(cv.best\_params\_)

# import pandas as pd

# import numpy as np

# from sklearn import preprocessing

# from sklearn.model\_selection import GridSearchCV, RandomizedSearchCV

# from sklearn.svm import SVC as svc

# from sklearn.metrics import make\_scorer, roc\_auc\_score

# from scipy import stats

# # DATA PREPARATION

# # DEFINE MODEL AND PERFORMANCE MEASURE

# mdl = BernoulliNB()

# auc = make\_scorer(roc\_auc\_score)

# # GRID SEARCH FOR 20 COMBINATIONS OF PARAMETERS

# grid\_list = {"C": np.arange(2, 10, 2),

# "gamma": np.arange(0.1, 1, 0.2)}

# grid\_search = GridSearchCV(mdl, param\_grid = grid\_list, n\_jobs = 4, cv = 3, scoring = auc)

# grid\_search.fit(x, y)

# print(grid\_search.cv\_results\_)

# # RANDOM SEARCH FOR 20 COMBINATIONS OF PARAMETERS

# rand\_list = {"C": stats.uniform(2, 10),

# "gamma": stats.uniform(0.1, 1)}

# rand\_search = RandomizedSearchCV(mdl, param\_distributions = rand\_list, n\_iter = 20, n\_jobs = 4, cv = 3, random\_state = 2017, scoring = auc)

# rand\_search.fit(x, y)

# print(rand\_search.cv\_results\_)

**if** sec**==**"Knn"**:**

x**,**y**=**self**.**onHazirlik**()**

test\_size1**=**float**(**self**.**lineEdit**.**text**())**

fold\_size**=**int**(**self**.**lineEdit\_2**.**text**())**

self**.**foldsizee**=**fold\_size

x\_train**,** x\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**x**,** y**,** test\_size **=** test\_size1**,** random\_state **=** 42**)**

self**.**w**.**csvYukle**(**x\_train**,**y\_train**,**x\_test**,**y\_test**)**

self**.**label\_22**.**setText**(**str**(**len**(**x\_train**)))**

self**.**label\_23**.**setText**(**str**(**len**(**y\_train**)))**

self**.**label\_24**.**setText**(**str**(**len**(**x\_test**)))**

self**.**label\_25**.**setText**(**str**(**len**(**x\_test**)))**

**from** sklearn**.**preprocessing **import** StandardScaler

sc\_X**=**StandardScaler**()**

x\_train**=**sc\_X**.**fit\_transform**(**x\_train**)**

x\_test**=**sc\_X**.**fit\_transform**(**x\_test**)**

**from** sklearn**.**neighbors **import** KNeighborsClassifier

knn **=** KNeighborsClassifier**(**n\_neighbors**=**5**)**

knn**.**fit**(**x\_train**,** y\_train**)**

y\_pred**=** knn**.**predict**(**x\_test**)**

self**.**pltPredict**(**y\_test**,** y\_pred**)**

self**.**pltTrue**(**y\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**label\_5**.**setText**(**str**(**round**(**acc**,**2**)))**

self**.**label\_4**.**setText**(**str**(**round**(**r2\_score**(**y\_test**,**y\_pred**),**2**)))**

X\_train2**,**X\_test2 **=**self**.**pcaIslem**(**x\_train**,**x\_test**,**5**)**

classifier2 **=** KNeighborsClassifier**(**n\_neighbors**=**5**)**

classifier2**.**fit**(**X\_train2**,**y\_train**)**

y\_pred2**=**classifier2**.**predict**(**X\_test2**)**

self**.**Cmatrix2**(**y\_test**,**y\_pred2**,**"PCA"**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred2**)\***100

self**.**label\_9**.**setText**(**str**(**round**(**acc**,**2**)))**

self**.**label\_8**.**setText**(**str**(**round**(**r2\_score**(**y\_test**,**y\_pred2**),**2**)))**

self**.**Cmatrix**(**y\_test**,**y\_pred**,**"Knn"**)**

**print(**confusion\_matrix**(**y\_test**,** y\_pred**))**

self**.**pltRoc2**(**y\_test**,**y\_pred**,**"Knn"**)**

**from** sklearn**.**model\_selection **import** KFold

**from** numpy **import** mean

**from** sklearn**.**model\_selection **import** cross\_val\_score

X **=** x**.**values

y **=** y**.**values

kf **=** KFold**(**n\_splits**=**fold\_size**)**

kf**.**get\_n\_splits**(**X**)**

sayma**=**0

**for** train\_index**,** test\_index **in** kf**.**split**(**X**):**

sayma**+=**1

# print("TRAIN:", train\_index, "TEST:", test\_index)

x\_train**,** x\_test **=** X**[**train\_index**],** X**[**test\_index**]**

y\_train**,** y\_test **=** y**[**train\_index**],** y**[**test\_index**]**

NBG **=** KNeighborsClassifier**(**n\_neighbors**=**5**)**

NBG**.**fit**(**x\_train**,** y\_train**)**

y\_pred **=** NBG**.**predict**(**x\_test**)**

self**.**CmatrixFold**(**y\_test**,**y\_pred**,**"fold-"**+**str**(**sayma**))**

self**.**kfoldCmatrix**(**y\_test**,** y\_pred**,**"Fold Sonuç"**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

**print(**acc**)**

model **=** KNeighborsClassifier**(**n\_neighbors**=**5**)**

scores **=** cross\_val\_score**(**model**,** X**,** y**,** scoring**=**'accuracy'**,** cv**=**kf**,** n\_jobs**=-**1**)**

self**.**label\_12**.**setText**(**str**(**round**(**mean**(**scores**\***100**),**2**)))**

self**.**label\_13**.**setText**(**str**(**round**(**scores**.**max**()\***100**,**2**)))**

**print(**'Accuracy: %.3f (%.3f)' **%** **(**mean**(**scores**),** scores**.**max**()))**

**print(**"--------------------"**)**

# dogru grid

# from sklearn.datasets import make\_blobs

# from sklearn.model\_selection import RepeatedStratifiedKFold

# from sklearn.model\_selection import GridSearchCV

# from sklearn.neighbors import KNeighborsClassifier

# # define dataset

# X, y = x,y

# # define models and parameters

# model =KNeighborsClassifier(n\_neighbors=5)

# n\_neighbors = range(1, 21, 2)

# weights = ['uniform', 'distance']

# metric = ['euclidean', 'manhattan', 'minkowski']

# # define grid search

# grid = dict(n\_neighbors=n\_neighbors,weights=weights,metric=metric)

# cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)

# grid\_search = GridSearchCV(estimator=model, param\_grid=grid, n\_jobs=-1, cv=cv, scoring='accuracy',error\_score=0)

# grid\_result = grid\_search.fit(X, y)

# # summarize results

# print("Best: %f using %s" % (grid\_result.best\_score\_, grid\_result.best\_params\_))

# means = grid\_result.cv\_results\_['mean\_test\_score']

# stds = grid\_result.cv\_results\_['std\_test\_score']

# params = grid\_result.cv\_results\_['params']

# for mean1, stdev, param in zip(means, stds, params):

# print("%f (%f) with: %r" % (mean1, stdev, param))

**if** sec**==**'Lojistik'**:**

x**,**y**=**self**.**onHazirlik**()**

test\_size1**=**float**(**self**.**lineEdit**.**text**())**

fold\_size**=**int**(**self**.**lineEdit\_2**.**text**())**

self**.**foldsizee**=**fold\_size

x\_train**,** x\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**x**,** y**,** test\_size **=** test\_size1**,** random\_state **=** 42**)**

self**.**w**.**csvYukle**(**x\_train**,**y\_train**,**x\_test**,**y\_test**)**

self**.**label\_22**.**setText**(**str**(**len**(**x\_train**)))**

self**.**label\_23**.**setText**(**str**(**len**(**y\_train**)))**

self**.**label\_24**.**setText**(**str**(**len**(**x\_test**)))**

self**.**label\_25**.**setText**(**str**(**len**(**x\_test**)))**

**from** sklearn**.**preprocessing **import** StandardScaler

sc**=**StandardScaler**()**

x\_train**=**sc**.**fit\_transform**(**x\_train**)**

x\_test**=**sc**.**fit\_transform**(**x\_test**)**

**from** sklearn**.**linear\_model **import** LogisticRegression

Logistic\_Regression **=** LogisticRegression**(**C**=**0.5**,**tol**=**0.1**,**multi\_class**=**'multinomial'**,**solver**=**'newton-cg'**,**penalty**=**'l2'**,**max\_iter**=**100**)**

Logistic\_Regression**.**fit**(**x\_train**,** y\_train**)**

y\_pred**=**Logistic\_Regression**.**predict**(**x\_test**)**

self**.**pltPredict**(**y\_test**,** y\_pred**)**

self**.**pltTrue**(**y\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

**print(**acc**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**label\_5**.**setText**(**str**(**round**(**acc**,**2**)))**

self**.**label\_4**.**setText**(**str**(**round**(**r2\_score**(**y\_test**,**y\_pred**),**2**)))**

X\_train2**,**X\_test2 **=**self**.**pcaIslem**(**x\_train**,**x\_test**,**5**)**

classifier2 **=** LogisticRegression**(**C**=**0.5**,**tol**=**0.1**,**multi\_class**=**'multinomial'**,**solver**=**'newton-cg'**,**penalty**=**'l2'**,**max\_iter**=**100**)**

classifier2**.**fit**(**X\_train2**,**y\_train**)**

y\_pred2**=**classifier2**.**predict**(**X\_test2**)**

self**.**Cmatrix2**(**y\_test**,**y\_pred2**,**"PCA"**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred2**)\***100

self**.**label\_9**.**setText**(**str**(**round**(**acc**,**2**)))**

self**.**label\_8**.**setText**(**str**(**round**(**r2\_score**(**y\_test**,**y\_pred2**),**2**)))**

self**.**Cmatrix**(**y\_test**,**y\_pred**,**"Lojistik"**)**

**print(**confusion\_matrix**(**y\_test**,** y\_pred**))**

self**.**pltRoc2**(**y\_test**,**y\_pred**,**"Lojistik"**)**

**from** sklearn**.**model\_selection **import** KFold

**from** numpy **import** mean

**from** sklearn**.**model\_selection **import** cross\_val\_score

X **=** x**.**values

y **=** y**.**values

kf **=** KFold**(**n\_splits**=**fold\_size**)**

kf**.**get\_n\_splits**(**X**)**

sayma**=**0

**for** train\_index**,** test\_index **in** kf**.**split**(**X**):**

sayma**+=**1

# print("TRAIN:", train\_index, "TEST:", test\_index)

x\_train**,** x\_test **=** X**[**train\_index**],** X**[**test\_index**]**

y\_train**,** y\_test **=** y**[**train\_index**],** y**[**test\_index**]**

NBG **=**LogisticRegression**(**C**=**0.5**,**tol**=**0.1**,**multi\_class**=**'multinomial'**,**solver**=**'newton-cg'**,**penalty**=**'l2'**,**max\_iter**=**100**)**

NBG**.**fit**(**x\_train**,** y\_train**)**

y\_pred **=** NBG**.**predict**(**x\_test**)**

self**.**CmatrixFold**(**y\_test**,**y\_pred**,**"fold-"**+**str**(**sayma**))**

self**.**kfoldCmatrix**(**y\_test**,** y\_pred**,**"Fold Sonuç"**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

**print(**acc**)**

model **=**LogisticRegression**(**C**=**0.5**,**tol**=**0.1**,**multi\_class**=**'multinomial'**,**solver**=**'newton-cg'**,**penalty**=**'l2'**,**max\_iter**=**100**)**

scores **=** cross\_val\_score**(**model**,** X**,** y**,** scoring**=**'accuracy'**,** cv**=**kf**,** n\_jobs**=-**1**)**

self**.**label\_12**.**setText**(**str**(**round**(**mean**(**scores**\***100**),**2**)))**

self**.**label\_13**.**setText**(**str**(**round**(**scores**.**max**()\***100**,**2**)))**

**print(**'Accuracy: %.3f (%.3f)' **%** **(**mean**(**scores**),** scores**.**max**()))**

# https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/

# dogru çalışıyor

# from scipy.stats import loguniform

# from sklearn.linear\_model import LogisticRegression

# from sklearn.model\_selection import RepeatedStratifiedKFold

# from sklearn.model\_selection import RandomizedSearchCV

# X, y = x,y

# # define model

# model = LogisticRegression()

# # define evaluation

# cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)

# # define search space

# space = dict()

# space['solver'] = ['newton-cg', 'lbfgs', 'liblinear']

# space['penalty'] = ['none', 'l1', 'l2', 'elasticnet']

# space['C'] = loguniform(1e-5, 100)

# # define search

# search = RandomizedSearchCV(model, space, n\_iter=50, scoring='accuracy', n\_jobs=-1, cv=cv, random\_state=1)

# # execute search

# result = search.fit(X, y)

# # summarize result

# print('Best Score: %s' % result.best\_score\_)

# print('Best Hyperparameters: %s' % result.best\_params\_)

**print(**"--------------------"**)**

# ########grid

# dogru

# from sklearn.linear\_model import LogisticRegression

# from sklearn.model\_selection import RepeatedStratifiedKFold

# from sklearn.model\_selection import GridSearchCV

# X, y = x,y

# # define model

# model = LogisticRegression()

# # define evaluation

# cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)

# # define search space

# space = dict()

# space['solver'] = ['newton-cg', 'lbfgs', 'liblinear']

# space['penalty'] = ['none', 'l1', 'l2', 'elasticnet']

# space['C'] = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100]

# # define search

# search = GridSearchCV(model, space, scoring='accuracy', n\_jobs=-1, cv=cv)

# # execute search

# result = search.fit(X, y)

# # summarize result

# print('Best Score: %s' % result.best\_score\_)

# print('Best Hyperparameters: %s' % result.best\_params\_)

**if** sec**==**'Svc'**:**

**from** sklearn**.**svm **import** SVC

x**,**y**=**self**.**onHazirlik**()**

test\_size1**=**float**(**self**.**lineEdit**.**text**())**

fold\_size**=**int**(**self**.**lineEdit\_2**.**text**())**

self**.**foldsizee**=**fold\_size

x\_train**,** x\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**x**,** y**,** test\_size **=** test\_size1**,** random\_state **=** 42**)**

self**.**w**.**csvYukle**(**x\_train**,**y\_train**,**x\_test**,**y\_test**)**

self**.**label\_22**.**setText**(**str**(**len**(**x\_train**)))**

self**.**label\_23**.**setText**(**str**(**len**(**y\_train**)))**

self**.**label\_24**.**setText**(**str**(**len**(**x\_test**)))**

self**.**label\_25**.**setText**(**str**(**len**(**x\_test**)))**

**from** sklearn**.**preprocessing **import** StandardScaler

sc\_X**=**StandardScaler**()**

x\_train**=**sc\_X**.**fit\_transform**(**x\_train**)**

x\_test**=**sc\_X**.**fit\_transform**(**x\_test**)**

NBB **=** SVC**(**kernel**=**'rbf'**)**

NBB**.**fit**(**x\_train**,** y\_train**)**

y\_pred **=** NBB**.**predict**(**x\_test**)**

self**.**pltPredict**(**y\_test**,** y\_pred**)**

self**.**pltTrue**(**y\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**label\_5**.**setText**(**str**(**round**(**acc**,**2**)))**

self**.**label\_4**.**setText**(**str**(**round**(**r2\_score**(**y\_test**,**y\_pred**),**2**)))**

X\_train2**,**X\_test2 **=**self**.**pcaIslem**(**x\_train**,**x\_test**,**5**)**

classifier2 **=** SVC**(**kernel**=**'rbf'**)**

classifier2**.**fit**(**X\_train2**,**y\_train**)**

y\_pred2**=**classifier2**.**predict**(**X\_test2**)**

self**.**Cmatrix2**(**y\_test**,**y\_pred2**,**"PCA"**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred2**)\***100

self**.**label\_9**.**setText**(**str**(**round**(**acc**,**2**)))**

self**.**label\_8**.**setText**(**str**(**round**(**r2\_score**(**y\_test**,**y\_pred2**),**2**)))**

self**.**Cmatrix**(**y\_test**,**y\_pred**,**"SVC"**)**

**print(**confusion\_matrix**(**y\_test**,** y\_pred**))**

self**.**pltRoc2**(**y\_test**,**y\_pred**,**"SVC"**)**

**from** sklearn**.**model\_selection **import** KFold

**from** numpy **import** mean

**from** sklearn**.**model\_selection **import** cross\_val\_score

**from** sklearn**.**preprocessing **import** StandardScaler

X **=** x**.**values

sc\_X**=**StandardScaler**()**

X**=**sc\_X**.**fit\_transform**(**X**)**

y **=** y**.**values

kf **=** KFold**(**n\_splits**=**fold\_size**)**

kf**.**get\_n\_splits**(**X**)**

sayma**=**0

**for** train\_index**,** test\_index **in** kf**.**split**(**X**):**

sayma**+=**1

# print("TRAIN:", train\_index, "TEST:", test\_index)

x\_train**,** x\_test **=** X**[**train\_index**],** X**[**test\_index**]**

y\_train**,** y\_test **=** y**[**train\_index**],** y**[**test\_index**]**

NBG **=** SVC**(**kernel**=**'rbf'**)**

NBG**.**fit**(**x\_train**,** y\_train**)**

y\_pred **=** NBG**.**predict**(**x\_test**)**

self**.**CmatrixFold**(**y\_test**,**y\_pred**,**"fold-"**+**str**(**sayma**))**

self**.**kfoldCmatrix**(**y\_test**,** y\_pred**,**"Fold Sonuç"**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

**print(**acc**)**

model **=** SVC**(**kernel**=**'rbf'**)**

scores **=** cross\_val\_score**(**model**,** X**,** y**,** scoring**=**'accuracy'**,** cv**=**kf**,** n\_jobs**=-**1**)**

self**.**label\_12**.**setText**(**str**(**round**(**mean**(**scores**\***100**),**2**)))**

self**.**label\_13**.**setText**(**str**(**round**(**scores**.**max**()\***100**,**2**)))**

**print(**'Accuracy: %.3f (%.3f)' **%** **(**mean**(**scores**),** scores**.**max**()))**

**print(**"--------------------"**)**

# hem grit hem random dogru

# import pandas as pd

# import numpy as np

# from sklearn import preprocessing

# from sklearn.model\_selection import GridSearchCV, RandomizedSearchCV

# from sklearn.svm import SVC as svc

# from sklearn.metrics import make\_scorer, roc\_auc\_score

# from scipy import stats

# # DATA PREPARATION

# # DEFINE MODEL AND PERFORMANCE MEASURE

# mdl = svc(probability = True, random\_state = 1)

# auc = make\_scorer(roc\_auc\_score)

# # GRID SEARCH FOR 20 COMBINATIONS OF PARAMETERS

# grid\_list = {"C": np.arange(2, 10, 2),

# "gamma": np.arange(0.1, 1, 0.2)}

# grid\_search = GridSearchCV(mdl, param\_grid = grid\_list, n\_jobs = 4, cv = 3, scoring = auc)

# grid\_search.fit(x, y)

# print(grid\_search.cv\_results\_)

# # RANDOM SEARCH FOR 20 COMBINATIONS OF PARAMETERS

# rand\_list = {"C": stats.uniform(2, 10),

# "gamma": stats.uniform(0.1, 1)}

# rand\_search = RandomizedSearchCV(mdl, param\_distributions = rand\_list, n\_iter = 20, n\_jobs = 4, cv = 3, random\_state = 2017, scoring = auc)

# rand\_search.fit(x, y)

# print(rand\_search.cv\_results\_)

###################################################

# dogru sadece grid

# from sklearn.datasets import make\_blobs

# from sklearn.model\_selection import RepeatedStratifiedKFold

# from sklearn.model\_selection import GridSearchCV

# from sklearn.svm import SVC

# # define dataset

# X, y = x,y

# # define model and parameters

# model = SVC()

# kernel = ['poly', 'rbf', 'sigmoid']

# C = [50, 10, 1.0, 0.1, 0.01]

# gamma = ['scale']

# # define grid search

# grid = dict(kernel=kernel,C=C,gamma=gamma)

# cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)

# grid\_search = GridSearchCV(estimator=model, param\_grid=grid, n\_jobs=-1, cv=cv, scoring='accuracy',error\_score=0)

# grid\_result = grid\_search.fit(X, y)

# # summarize results

# print("Best: %f using %s" % (grid\_result.best\_score\_, grid\_result.best\_params\_))

# means = grid\_result.cv\_results\_['mean\_test\_score']

# stds = grid\_result.cv\_results\_['std\_test\_score']

# params = grid\_result.cv\_results\_['params']

# for mean1, stdev, param in zip(means, stds, params):

# print("%f (%f) with: %r" % (mean1, stdev, param))

# print("bitti")

**def** showPredfgr6**(**self**,**y\_test**,**y\_pred**,**isim**):**

self**.**predfigr6**=** Windoww**()**

cm **=** confusion\_matrix**(**y\_test**,** y\_pred**)**

classNames **=** **[**'VAR'**,**'YOK'**]**

cm\_data **=** pd**.**DataFrame**(**cm**,**index **=** classNames**,**

columns **=** classNames**)**

# plt.figure(figsize = (5,4))

sns**.**heatmap**(**cm\_data**,** annot**=True,**fmt**=**"d"**)**

plt**.**title**(**isim**)**

plt**.**ylabel**(**'Actual label'**)**

plt**.**xlabel**(**'Predicted label'**)**

plt**.**show**()**

self**.**predfigr6**.**show**()**

**def** kfoldCmatrix**(**self**,** y\_test**,** y\_pred**,**baslik**):**

**if** self**.**foldsizee**==**int**(**self**.**lineEdit\_2**.**text**()):**

self**.**cm2**=[[**0**,** 0**],**

**[**0**,** 0**]]**

cm **=** confusion\_matrix**(**y\_test**,** y\_pred**)**

**if** self**.**foldsizee **!=**0**:**

self**.**cm2 **+=** cm

self**.**foldsizee **-=**1

**if(**self**.**foldsizee **==** 0**):**

# a1=Windoww()

self**.**showPredfgr5**(**baslik**)**

# a1.show()

**def** showEnsm**(**self**,**y\_test**,**y\_pred**,**baslik**):**

self**.**predfigr3 **=** Windoww**()**

**from** sklearn**.**metrics **import** roc\_curve

**from** sklearn**.**metrics **import** roc\_auc\_score

**from** matplotlib **import** pyplot

lr\_auc **=** roc\_auc\_score**(**y\_test**,** y\_pred**)**

# summarize scores

**print(**'ALGRTM: ROC AUC=%.3f' **%** **(**lr\_auc**))**

# calculate roc curves

lr\_fpr**,** lr\_tpr**,** \_ **=** roc\_curve**(**y\_test**,** y\_pred**)**

# plot the roc curve for the model

pyplot**.**plot**(**lr\_fpr**,** lr\_tpr**,** marker**=**'.'**,** label**=**baslik**)**

# axis labels

pyplot**.**xlabel**(**'False Positive Rate'**)**

pyplot**.**ylabel**(**'True Positive Rate'**)**

# show the legend

pyplot**.**legend**()**

pyplot**.**show**()**

self**.**predfigr3**.**show**()**

-Proccessing butonuna basıldığında regAlgm fonksiyonu çağırılır. Her algoritma için arayüzde verilen test size ve fold size değerlerine göre veriler ayrılır. İlgili algoritmaya göre fit ve predict işlemi yapılır. Sonucu gerekli labellara yazdılır, confusion matrix ve roc eğrisi gösterilir. Pca(5) için tekrar fit ve predict işlemleri yapılır sonucu gerekli labellara yazdılrılır ve confusion matrix olarak gösterilir. K-fold için fold size göre fit ve predict işlemleri yapılır. Her fold sonucu confusion matrix olarak gösterilir ve bütün foldların confusion matrixleri toplanarak overlapped matrix elde edilir. Son olarak GridSearchCV ve RandomizedSearchCV ile işlem yapılır. En uygun sonuç parametreler ile bulunur.

**4)Visualization Butonuna Basıldığında Yürütülen Kodlar**

**def** gorsel**(**self**):**

sns**.**countplot**(**x **=** "Müşteri Kaybı"**,** data **=** self**.**veriler**)**

Kategorik **=** self**.**veriler**.**select\_dtypes**(**include**=**'object'**).**drop**(**'Müşteri Kaybı'**,** axis**=**1**).**columns**.**tolist**()**

# Sayısal = self.veriler.select\_dtypes(exclude='object').columns.tolist()

plt**.**figure**(**figsize**=(**18**,**18**))**

**for** i**,**c **in** enumerate**(**Kategorik**):**

plt**.**subplot**(**5**,**4**,**i**+**1**)**

sns**.**countplot**(**self**.**veriler**[**c**],** hue**=**self**.**veriler**[**'Müşteri Kaybı'**])**

plt**.**title**(**c**)**

plt**.**xlabel**(**''**)**

plt**.**figure**(**figsize**=(**20**,**5**))**

**for** i**,**c **in** enumerate**([**'Abonelik Süresi'**,** 'Aylık Ödeme'**,** 'Toplam Ödeme'**]):**

plt**.**subplot**(**1**,**3**,**i**+**1**)**

sns**.**distplot**(**self**.**veriler**[**self**.**veriler**[**'Müşteri Kaybı'**]** **==** 'Müşteri Kaybı Yok'**][**c**],** kde**=True,** color**=**'blue'**,** hist**=False,** kde\_kws**=**dict**(**linewidth**=**2**),** label**=**'Müşteri Kaybı Oluşmadı'**)**

sns**.**distplot**(**self**.**veriler**[**self**.**veriler**[**'Müşteri Kaybı'**]** **==** 'Müşteri Kaybı Var'**][**c**],** kde**=True,** color**=**'Orange'**,** hist**=False,** kde\_kws**=**dict**(**linewidth**=**2**),** label**=**'Müşteri Kaybı Oluştu'**)**

plt**.**title**(**c**)**

plt**.**show**()**

**from** sklearn**.**model\_selection **import** train\_test\_split

**from** sklearn**.**metrics **import** roc\_curve

**from** sklearn**.**metrics **import** roc\_auc\_score

**from** matplotlib **import** pyplot

**from** sklearn**.**datasets **import** make\_classification

**from** sklearn**.**naive\_bayes **import** GaussianNB

x**,**y**=**self**.**onHazirlik**()**

test\_size1**=**float**(**self**.**lineEdit**.**text**())**

trainx**,** testx**,** trainy**,** testy **=** train\_test\_split**(**x**,** y**,** test\_size**=**test\_size1**,** random\_state**=**42**)**

NBG **=** GaussianNB**()**

**from** sklearn**.**naive\_bayes **import** BernoulliNB

BNN **=** BernoulliNB**()**

**from** sklearn**.**neighbors **import** KNeighborsClassifier

KNN**=** KNeighborsClassifier**(**n\_neighbors**=**5**)**

**from** sklearn**.**linear\_model **import** LogisticRegression

Logistic\_Regression **=** LogisticRegression**(**C**=**0.5**,**tol**=**0.1**,**multi\_class**=**'multinomial'**,**solver**=**'newton-cg'**,**penalty**=**'l2'**,**max\_iter**=**100**)**

**from** sklearn**.**svm **import** SVC

svc1 **=** SVC**(**probability**=True)**

x**,** y **=** make\_classification**(**n\_samples**=**1000**,** n\_classes**=**2**,** random\_state**=**1**)**

KNN\_tahmin **=** **[**0 **for** \_ **in** range**(**len**(**testy**))]**

NBG\_tahmin **=** **[**0 **for** \_ **in** range**(**len**(**testy**))]**

svc1\_tahmin **=** **[**0 **for** \_ **in** range**(**len**(**testy**))]**

Logistic\_Regression\_tahmin **=** **[**0 **for** \_ **in** range**(**len**(**testy**))]**

BNN\_tahmin **=** **[**0 **for** \_ **in** range**(**len**(**testy**))]**

**from** sklearn**.**preprocessing **import** StandardScaler

sc\_X**=**StandardScaler**()**

trainx**=**sc\_X**.**fit\_transform**(**trainx**)**

testx**=**sc\_X**.**transform**(**testx**)**

model **=** NBG

model**.**fit**(**trainx**,** trainy**)**

self**.**allmodelss**.**append**(**model**)**

model2 **=** KNN

model2**.**fit**(**trainx**,** trainy**)**

self**.**allmodelss**.**append**(**model2**)**

model3**=**svc1

model3**.**fit**(**trainx**,** trainy**)**

self**.**allmodelss**.**append**(**model3**)**

model4**=**Logistic\_Regression

model4**.**fit**(**trainx**,** trainy**)**

self**.**allmodelss**.**append**(**model4**)**

model5**=**BNN

model5**.**fit**(**trainx**,** trainy**)**

self**.**allmodelss**.**append**(**model5**)**

NBG\_tahmin **=** model**.**predict\_proba**(**testx**)**

KNN\_tahmin **=** model2**.**predict\_proba**(**testx**)**

svc1\_tahmin **=** model3**.**predict\_proba**(**testx**)**

Logistic\_Regression\_tahmin**=** model4**.**predict\_proba**(**testx**)**

BNN\_tahmin**=** model5**.**predict\_proba**(**testx**)**

NBG\_tahmin **=** NBG\_tahmin**[:,** 1**]**

KNN\_tahmin **=** KNN\_tahmin**[:,** 1**]**

svc1\_tahmin **=** svc1\_tahmin**[:,** 1**]**

Logistic\_Regression\_tahmin **=** Logistic\_Regression\_tahmin**[:,** 1**]**

BNN\_tahmin**=** BNN\_tahmin**[:,** 1**]**

KNN\_hassasiyet **=** roc\_auc\_score**(**testy**,** KNN\_tahmin**)**

NBG\_hassasiyet **=** roc\_auc\_score**(**testy**,** NBG\_tahmin**)**

svc1\_hassasiyet **=** roc\_auc\_score**(**testy**,** svc1\_tahmin**)**

Logistic\_Regression\_hassasiyet **=** roc\_auc\_score**(**testy**,** Logistic\_Regression\_tahmin**)**

BNN\_hassasiyet**=** roc\_auc\_score**(**testy**,** BNN\_tahmin**)**

**print(**'KNN: ROC AUC=%.3f' **%** **(**KNN\_hassasiyet**))**

**print(**'Navy Bayes Gaussian: ROC AUC=%.3f' **%** **(**NBG\_hassasiyet**))**

**print(**'SVC: ROC AUC=%.3f' **%** **(**svc1\_hassasiyet**))**

**print(**'Logistic Regresyon: ROC AUC=%.3f' **%** **(**Logistic\_Regression\_hassasiyet**))**

**print(**'BNN: ROC AUC=%.3f' **%** **(**BNN\_hassasiyet**))**

KNN\_fpr**,** KNN\_tpr**,** \_ **=** roc\_curve**(**testy**,** KNN\_tahmin**)**

NBG\_fpr**,** NBG\_tpr**,** \_ **=** roc\_curve**(**testy**,** NBG\_tahmin**)**

svc1\_fpr**,** svc1\_tpr**,** \_ **=** roc\_curve**(**testy**,** svc1\_tahmin**)**

Logistic\_Regression\_fpr**,** Logistic\_Regression\_tpr**,** \_ **=** roc\_curve**(**testy**,** Logistic\_Regression\_tahmin**)**

BNN\_fpr**,** BNN\_tpr**,** \_ **=** roc\_curve**(**testy**,** BNN\_tahmin**)**

pyplot**.**plot**(**KNN\_fpr**,** KNN\_tpr**,** linestyle**=**'--'**,** label**=**'KNN'**)**

pyplot**.**plot**(**NBG\_fpr**,** NBG\_tpr**,** marker**=**'.'**,** label**=**'Navy Bayes Gaussian'**)**

pyplot**.**plot**(**svc1\_fpr**,** svc1\_tpr**,** marker**=**'.'**,** label**=**'SVC'**)**

pyplot**.**plot**(**Logistic\_Regression\_fpr**,** Logistic\_Regression\_tpr**,** marker**=**'.'**,** label**=**'Logistic Regresyon'**)**

pyplot**.**plot**(**BNN\_fpr**,** BNN\_tpr**,** marker**=**'.'**,** label**=**'BNN'**)**

pyplot**.**xlabel**(**'Gerçek Müşteri Kaybı Oluştu'**)**

pyplot**.**ylabel**(**'Gerçek Müşteri Kaybı Oluşmadı'**)**

pyplot**.**legend**()**

pyplot**.**show**()**

**print(**"Başarı Değerleri"**)**

model\_accuracy **=** pd**.**Series**(**data**=[**KNN\_hassasiyet**,**NBG\_hassasiyet**,**svc1\_hassasiyet**,**Logistic\_Regression\_hassasiyet**,**BNN\_hassasiyet**],**

index**=[**'Knn'**,**'GaussianNB'**,**'Svm'**,**'Logistic'**,**'Bnn'**])**

fig**=** plt**.**figure**(**figsize**=(**8**,**8**))**

model\_accuracy**.**sort\_values**().**plot**.**barh**()**

plt**.**title**(**'Model Accuracy'**)**

plt**.**show**()**

**print(**"--------------------"**)**

-İlk olarak sns ile müşteri kaybı grafiği gösterilir. İlk iki for döngüsünde müşteri kaybının diğer öznitelikleri ile bağlantısını içeren grafikler gösterilir. Uygulamada kullanılan beş algoritma roc grafiğinde işleme alınır ve çıktısı gösterilir. Son olarak bu algoritmaların başarıları çubuk grafikte sergilenir.

**5)Cnn Butonuna Basıldığında Yürütülen Kodlar**

x**,**y**=**self**.**onHazirlik**()**

test\_size1**=**float**(**self**.**lineEdit**.**text**())**

x\_train**,** x\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**x**,** y**,** test\_size **=**test\_size1 **,** random\_state **=** 42**)**

**from** keras**.**utils **import** to\_categorical

y\_train **=** to\_categorical**(**y\_train**,** 2**)**

y\_test**=** to\_categorical**(**y\_test**,** 2**)**

**from** keras**.**models **import** Sequential

**from** keras**.**layers **import** Dense**,**Dropout**,**BatchNormalization**,**Activation

#modeli oluşturalım

model **=** Sequential**()**

#eğitim verisinde kaç tane stun yani model için girdi sayısı var onu alalım

n\_cols **=** x\_train**.**shape**[**1**]**

#model katmanlarını ekleyelim

model**.**add**(**Dense**(**16**,** input\_shape**=(**n\_cols**,)))**

model**.**add**(**Activation**(**"relu"**))**

model**.**add**(**BatchNormalization**())**

model**.**add**(**Dropout**(**0.5**))**

model**.**add**(**Dense**(**9**))**

model**.**add**(**Activation**(**"relu"**))**

model**.**add**(**BatchNormalization**())**

model**.**add**(**Dropout**(**0.5**))**

model**.**add**(**Dense**(**6**))**

model**.**add**(**Activation**(**"relu"**))**

model**.**add**(**BatchNormalization**())**

model**.**add**(**Dropout**(**0.5**))**

model**.**add**(**Dense**(**2**,** activation**=**'softmax'**))**

model**.**summary**()**

model**.**compile**(**optimizer**=**'adam'**,** loss**=**'categorical\_crossentropy'**,** metrics**=[**'accuracy'**])**

history **=**model**.**fit**(**x\_train**,**

y\_train**,**

validation\_data**=(**x\_test**,** y\_test**),**

batch\_size**=**16**,**

shuffle**=True,**

verbose**=**1**,**

epochs**=**10**)**

score **=** model**.**evaluate**(**x\_test**,** y\_test**,** verbose**=**0**)**

**print(**'Test loss:'**,** score**[**0**])**

**print(**'Test accuracy:'**,** score**[**1**])**

**from** matplotlib **import** pyplot **as** plt

# Plot training & validation accuracy values

# plt.figure(figsize=(14,3))

plt**.**subplot**(**1**,** 2**,** 1**)**

plt**.**plot**(**history**.**history**[**'accuracy'**])**

plt**.**plot**(**history**.**history**[**'val\_accuracy'**])**

plt**.**title**(**'Model accuracy'**)**

plt**.**ylabel**(**'Accuracy'**)**

plt**.**xlabel**(**str**(**round**(**score**[**1**]\***100**,**3**)))**

plt**.**legend**([**'Train'**,** 'Test'**],** loc**=**'upper left'**)**

# Plot training & validation loss values

plt**.**subplot**(**1**,** 2**,** 2**)**

plt**.**plot**(**history**.**history**[**'loss'**])**

plt**.**plot**(**history**.**history**[**'val\_loss'**])**

plt**.**title**(**'Model loss'**)**

plt**.**ylabel**(**'Loss'**)**

plt**.**xlabel**(**str**(**round**(**score**[**0**],**3**)))**

plt**.**legend**([**'Train'**,** 'Test'**],** loc**=**'upper left'**)**

plt**.**show**()**

a**.**show**()**

**print(**'----Sonuç-----'**)**

score **=** model**.**evaluate**(**x\_test**,** y\_test**,** verbose**=**0**)**

**print(**'Test loss:'**,** score**[**0**])**

**print(**'Test accuracy:'**,** score**[**1**])**

# print(len(y\_pred))

y\_pred **=** model**.**predict**(**x\_test**)**

y\_test **=** y\_test**.**reshape**(-**1**,** 1**)**

y\_pred**=**y\_pred**.**reshape**(-**1**,** 1**)**

**print(**confusion\_matrix**(**y\_test**,** y\_pred**.**round**()))**

y\_pred2**=**y\_pred**.**round**()**

self**.**Cmatrix**(**y\_test**,**y\_pred2**,**"Derin Öğrenme"**)**

self**.**pltRoc2**(**y\_test**,**y\_pred**,**"Derin Öğrenme"**)**

-YSA ile model oluşturulur epoch sayısı 10, batch size 16 olan bir eğitim gerçekleştirilir. Sonuç olarak acc-loss grafikleri, confusion matrixi ve roc eğrisi gösterilir.

**6)Ensemble Butonuna Basıldığında Yürütülen Kodlar**

**def** get\_stacking**(**self**):**

**from** sklearn**.**linear\_model **import** LogisticRegression

**from** sklearn**.**neighbors **import** KNeighborsClassifier

**from** sklearn**.**svm **import** SVC

**from** sklearn**.**naive\_bayes **import** GaussianNB

**from** sklearn**.**ensemble **import** StackingClassifier

**from** sklearn**.**naive\_bayes **import** BernoulliNB

level0 **=** list**()**

level0**.**append**((**'lr'**,** LogisticRegression**(**C**=**0.5**,**tol**=**0.1**,**multi\_class**=**'multinomial'**,**solver**=**'newton-cg'**,**penalty**=**'l2'**,**max\_iter**=**100**)))**

level0**.**append**((**'knn'**,** KNeighborsClassifier**(**n\_neighbors**=**5**)))**

level0**.**append**((**'benoull'**,** BernoulliNB**()))**

level0**.**append**((**'svm'**,** SVC**(**kernel**=**'rbf'**)))**

level0**.**append**((**'bayes'**,** GaussianNB**()))**

# define meta learner model

level1 **=**LogisticRegression**()**

# define the stacking ensemble

model **=** StackingClassifier**(**estimators**=**level0**,** final\_estimator**=**level1**,** cv**=**5**)**

**return** model

**def** get\_models**(**self**):**

**from** sklearn**.**model\_selection **import** RepeatedStratifiedKFold

**from** sklearn**.**linear\_model **import** LogisticRegression

**from** sklearn**.**neighbors **import** KNeighborsClassifier

**from** sklearn**.**svm **import** SVC

**from** sklearn**.**naive\_bayes **import** GaussianNB

**from** sklearn**.**naive\_bayes **import** BernoulliNB

models **=** dict**()**

models**[**'lr'**]** **=** LogisticRegression**(**C**=**0.5**,**tol**=**0.1**,**multi\_class**=**'multinomial'**,**solver**=**'newton-cg'**,**penalty**=**'l2'**,**max\_iter**=**100**)**

models**[**'knn'**]** **=** KNeighborsClassifier**(**n\_neighbors**=**5**)**

models**[**'benoull'**]** **=** BernoulliNB**()**

models**[**'svm'**]** **=** SVC**(**kernel**=**'rbf'**)**

models**[**'bayes'**]** **=** GaussianNB**()**

models**[**'stacking'**]** **=** self**.**get\_stacking**()**

**return** models

**def** evaluate\_model**(**self**,**model**,** X**,** y**):**

# from sklearn.model\_selection import RepeatedKFold

**from** sklearn**.**model\_selection **import** cross\_val\_score

**from** sklearn**.**model\_selection **import** RepeatedStratifiedKFold

cv **=** RepeatedStratifiedKFold**(**n\_splits**=**10**,** n\_repeats**=**3**,** random\_state**=**1**)**

scores **=** cross\_val\_score**(**model**,** X**,** y**,** scoring**=**'accuracy'**,** cv**=**cv**,** n\_jobs**=-**1**,** error\_score**=**'raise'**)**

**return** scores

**def** ensembleLearning**(**self**):**

**from** sklearn**.**linear\_model **import** LogisticRegression

**from** sklearn**.**neighbors **import** KNeighborsClassifier

**from** sklearn**.**svm **import** SVC

**from** sklearn**.**naive\_bayes **import** GaussianNB

**from** sklearn**.**naive\_bayes **import** BernoulliNB

**from** numpy **import** mean

**from** numpy **import** std

**from** matplotlib **import** pyplot

test\_size1**=**float**(**self**.**lineEdit**.**text**())**

X**,** y **=** self**.**onHazirlik**()**

**print(**"StackingClassifier"**)**

models **=** self**.**get\_models**()**

# evaluate the models and store results

results**,** names **=** list**(),** list**()**

**for** name**,** model **in** models**.**items**():**

scores **=** self**.**evaluate\_model**(**model**,** X**,** y**)**

results**.**append**(**scores**)**

names**.**append**(**name**)**

self**.**label\_30**.**setText**(**str**(**round**(**mean**(**scores**),**3**)))**

**print(**'>%s %.3f (%.3f)' **%** **(**name**,** mean**(**scores**),** std**(**scores**)))**

# plot model performance for comparison

pyplot**.**boxplot**(**results**,** labels**=**names**,** showmeans**=True)**

pyplot**.**show**()**

**print(**"\*\*\*\*\*\*\*\*\*\*"**)**

**print(**"BaggingClassifier"**)**

**from** sklearn**.**ensemble **import** BaggingClassifier

**from** sklearn **import** tree

x\_train**,** x\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**X**,** y**,** test\_size **=**test\_size1 **,** random\_state **=** 42**)**

model**=**BaggingClassifier**(**tree**.**DecisionTreeClassifier**(**random\_state**=**1**))**

model**.**fit**(**x\_train**,** y\_train**)**

y\_test**=**np**.**array**(**y\_test**)**

y\_pred**=**model**.**predict**(**x\_test**)**

**print(**round**(**accuracy\_score**(**y\_test**,**y\_pred**),**3**))**

self**.**label\_33**.**setText**(**str**(**round**(**accuracy\_score**(**y\_test**,**y\_pred**),**3**)))**

self**.**Cmatrix**(**y\_test**,**y\_pred**,**"Bagging"**)**

self**.**pltRoc2**(**y\_test**,**y\_pred**,**"Bagging"**)**

**print(**"\*\*\*\*\*\*\*\*\*\*"**)**

**print(**"VotingClassifier"**)**

**from** sklearn**.**linear\_model **import** LogisticRegression

**from** sklearn**.**naive\_bayes **import** GaussianNB

**from** sklearn**.**ensemble **import** RandomForestClassifier**,** VotingClassifier

clf1 **=** LogisticRegression**(**C**=**0.5**,**tol**=**0.1**,**multi\_class**=**'multinomial'**,**solver**=**'newton-cg'**,**penalty**=**'l2'**,**max\_iter**=**100**)**

clf2 **=** KNeighborsClassifier**(**n\_neighbors**=**5**)**

clf3 **=** GaussianNB**()**

clf4**=** SVC**(**probability**=True)**

clf5**=** BernoulliNB**()**

X **=** X

y **=** y

x\_train**,** x\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**X**,** y**,** test\_size **=**test\_size1 **,** random\_state **=** 42**)**

eclf1 **=** VotingClassifier**(**estimators**=[**

**(**'lr'**,** clf1**),** **(**'knn'**,** clf2**),** **(**'gnb'**,** clf3**),(**'svc'**,** clf4**),(**'bnn'**,** clf5**)],** voting**=**'soft'**)**

eclf1 **=** eclf1**.**fit**(**x\_train**,** y\_train**)**

np**.**array\_equal**(**eclf1**.**named\_estimators\_**.**lr**.**predict**(**X**),**

eclf1**.**named\_estimators\_**[**'lr'**].**predict**(**X**))**

eclf2 **=** VotingClassifier**(**estimators**=[**

**(**'lr'**,** clf1**),** **(**'knn'**,** clf2**),** **(**'gnb'**,** clf3**),(**'svc'**,** clf4**),(**'bnn'**,** clf5**)],**

voting**=**'soft'**)**

eclf2 **=** eclf2**.**fit**(**x\_train**,**y\_train**)**

eclf3 **=** VotingClassifier**(**estimators**=[**

**(**'lr'**,** clf1**),** **(**'knn'**,** clf2**),** **(**'gnb'**,** clf3**),(**'svc'**,** clf4**),(**'bnn'**,** clf5**)],**

voting**=**'soft'**,** weights**=[**1**,**1**,**1**,**1**,**1**],**

flatten\_transform**=True)**

eclf3 **=** eclf3**.**fit**(**x\_train**,**y\_train**)**

# print(eclf3.predict(X))

y\_pred**=**eclf3**.**predict**(**x\_test**)**

score **=** accuracy\_score**(**y\_test**,** y\_pred**)**

**print(**round**(**score**,**3**))**

self**.**label\_29**.**setText**(**str**(**round**(**score**,**3**)))**

self**.**Cmatrix**(**y\_test**,**y\_pred**,**"Voting"**)**

self**.**pltRoc2**(**y\_test**,**y\_pred**,**"Voting"**)**

**print(**"\*\*\*\*\*\*\*\*\*\*"**)**

**print(**"XGBClassifier"**)**

x\_train**,** x\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**X**,** y**,** test\_size **=**test\_size1 **,** random\_state **=** 42**)**

**import** xgboost **as** xgb

model**=**xgb**.**XGBClassifier**(**random\_state**=**1**,**learning\_rate**=**0.01**)**

model**.**fit**(**x\_train**,** y\_train**)**

y\_test**=**np**.**array**(**y\_test**)**

y\_pred**=**model**.**predict**(**x\_test**)**

score **=** accuracy\_score**(**y\_test**,** y\_pred**)**

**print(**round**(**score**,**3**))**

self**.**label\_34**.**setText**(**str**(**round**(**score**,**3**)))**

self**.**Cmatrix**(**y\_test**,**y\_pred**,**"XGB"**)**

self**.**pltRoc2**(**y\_test**,**y\_pred**,**"XGB"**)**

-İlk olarak ensembleLearning fonksiyonu çalışır ve ilk işleme alınacak algritma stacking’dir. Modeller çağırılır ve işleme alınır. Diğer algoritmaların sonuç değerlerinden daha yüksek olan stacking değeri oluşturulur. Sırasıyla Baggining, Voiting ve Xgb algoritmaları işleme alınır ve her biri için roc eğrisi ve confusion matrixleri gösterilir.