KASTAMONU ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

MAKİNE ÖĞRENMESİ DERSİ ÖDEV

RAPORU

ÖDEV

Makine Öğrenmesi Uygulamaları

ÖDEV TARİHİ

18.01.2021

DERSİN SORUMLUSU

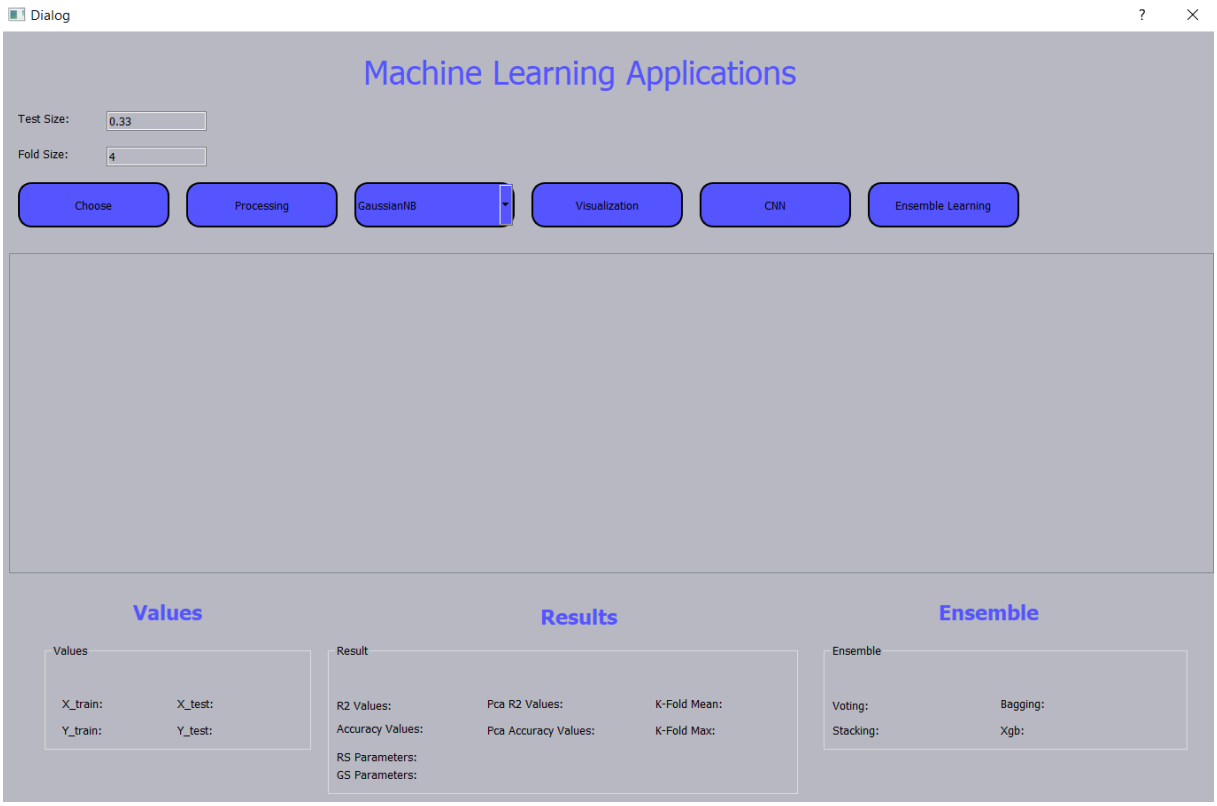
Kemal AKYOL

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.

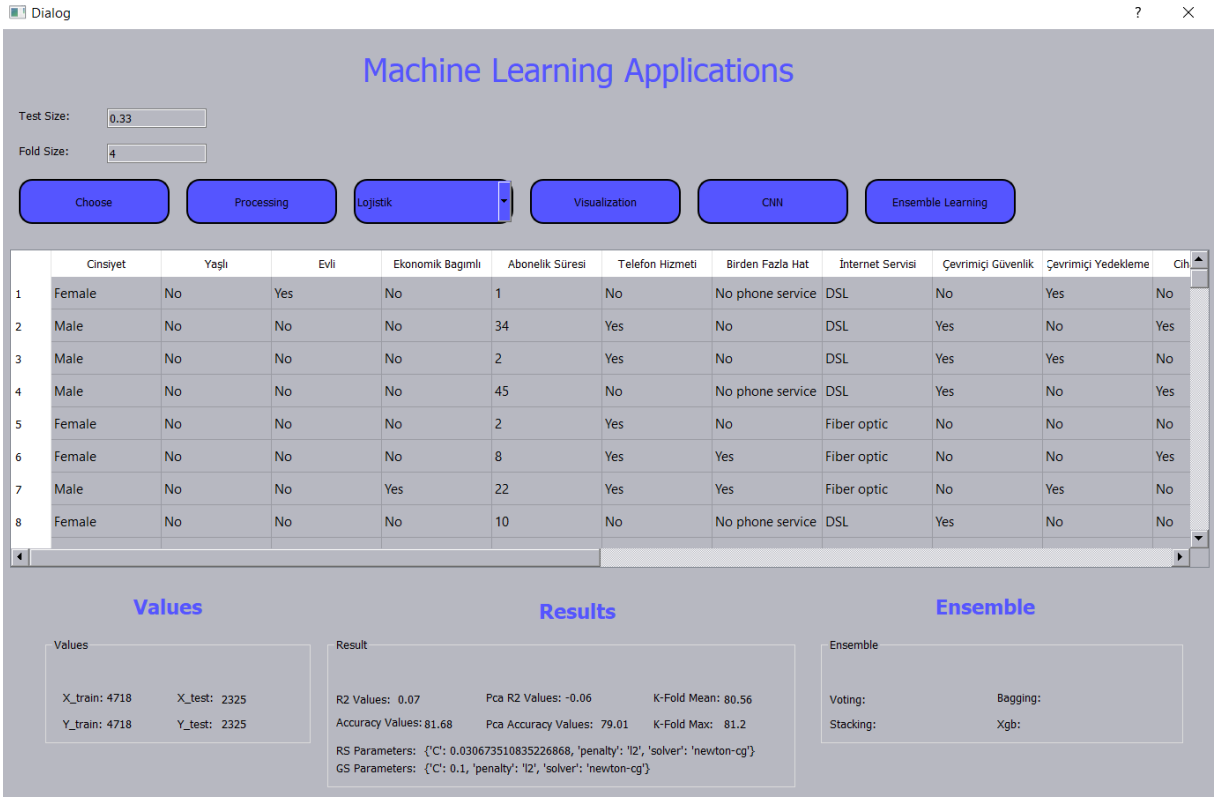
-ComboBox ile ilgili algoritma seçilir.

-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



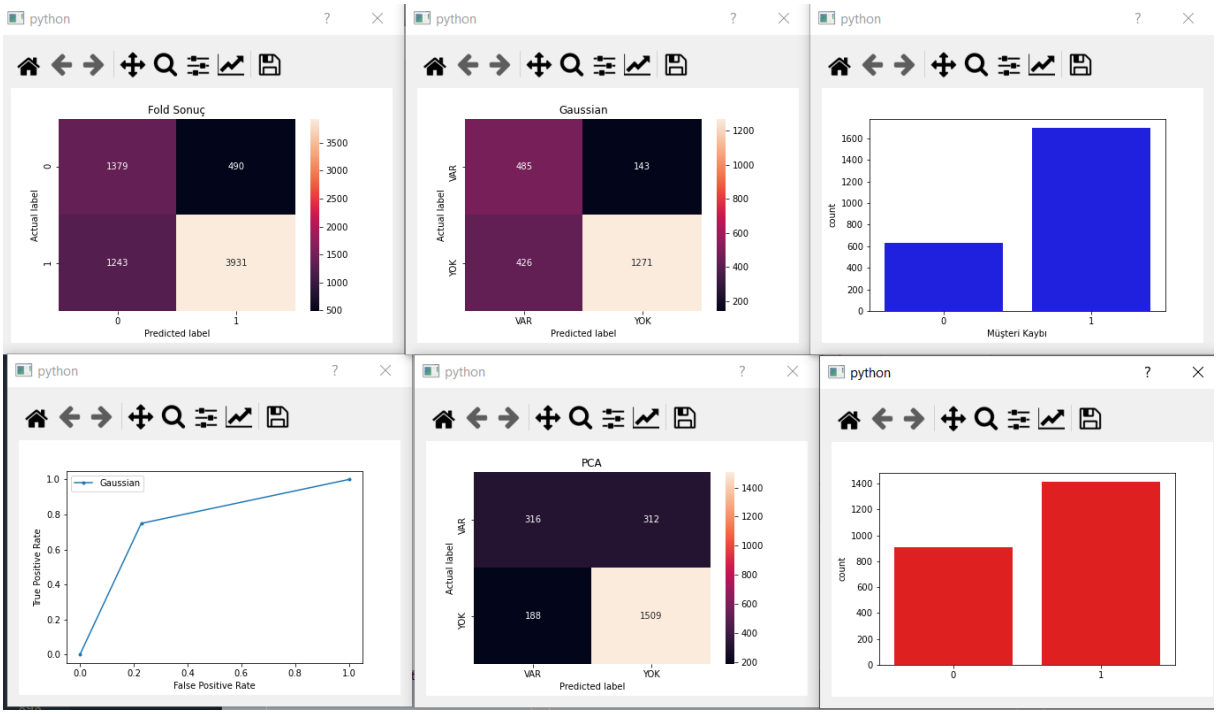
	Cinsiyet	Yaşlı	Evli	Ekonomik Bagımlı	Abonelik Süresi	Telefon Hizmeti	Birden Fazla Hat	İnternet Servisi	Çevrimiçi Güvenlik	Çevrimiçi Yedekleme	Cih
1	Female	No	Yes	No	1	No	No phone service	DSL	No	Yes	No
2	Male	No	No	No	34	Yes	No	DSL	Yes	No	Yes
3	Male	No	No	No	2	Yes	No	DSL	Yes	Yes	No
4	Male	No	No	No	45	No	No phone service	DSL	Yes	No	Yes
5	Female	No	No	No	2	Yes	No	Fiber optic	No	No	No
6	Female	No	No	No	8	Yes	Yes	Fiber optic	No	No	Yes
7	Male	No	No	Yes	22	Yes	Yes	Fiber optic	No	Yes	No
8	Female	No	No	No	10	No	No phone service	DSL	Yes	No	No

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

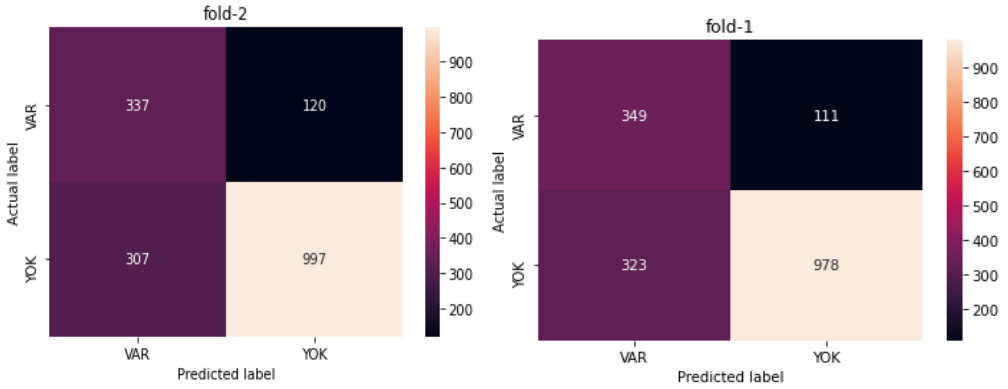
-Result alanında hold-out ile test size değerine göre yapılan işlem sonucu çıkan Logistic regression accuracy ve r2 kare değeri, pca(5) accuracy ve r2 değeri, fold size değerine göre K-fold ortalama ve maximum değeri, RandomizedSearchCV ve GridSearchCV sonucunda çıkan parametreleri gösterilmiştir.

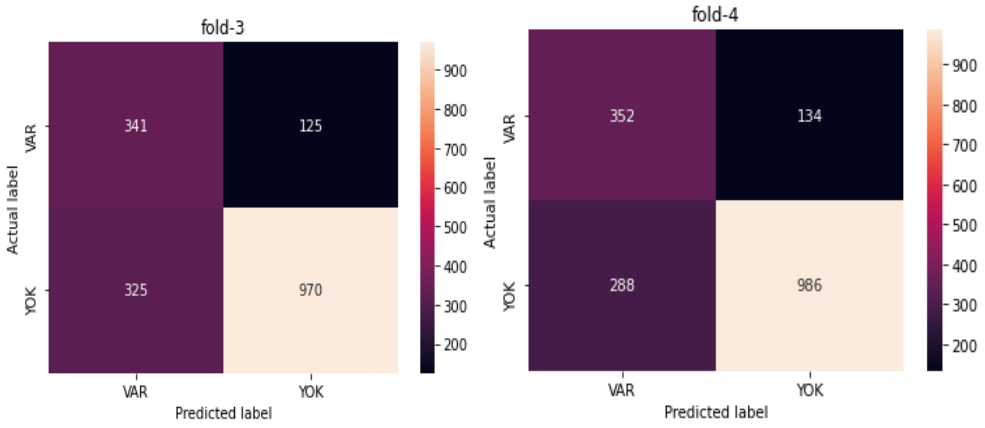
-Bu işlemlerin bir arada yapılmasının sebebi yapılan bu uygulama makine öğrenmesinin yanında bir analiz uygulaması olmasıdır. Her algoritma için bütün işlemlerin sonuçları gösterilir. Sonuçlara bakılarak hangi işlemin sonucunun daha başarılı olduğu anlaşılmaktadır.

-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.

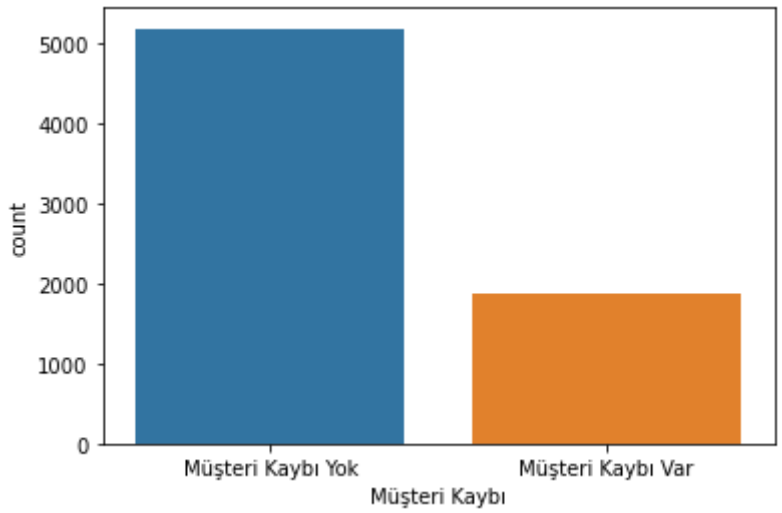




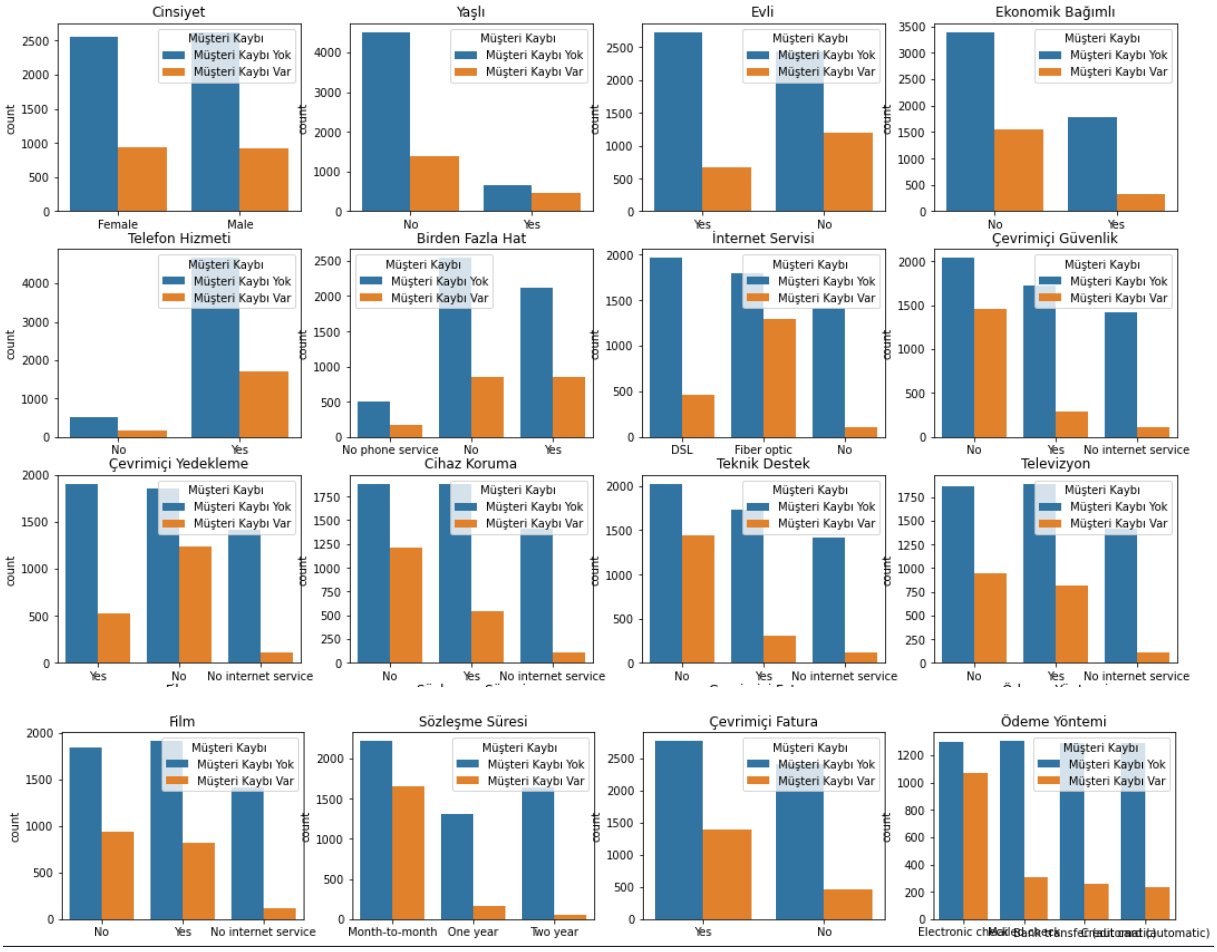
-Fold size değerine göre çıkan foldların confusion matrix grafikleri gözükmektedir.

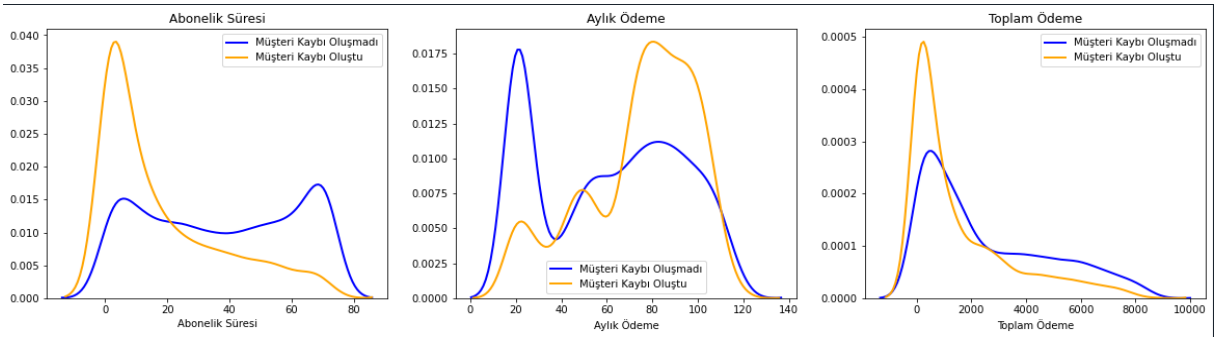
-Son olarak hold-out sonucunda test size değerine göre çekilen veriler gösterilir.

2) Visualization Butonuna Basıldığında Yürtülen İşlem

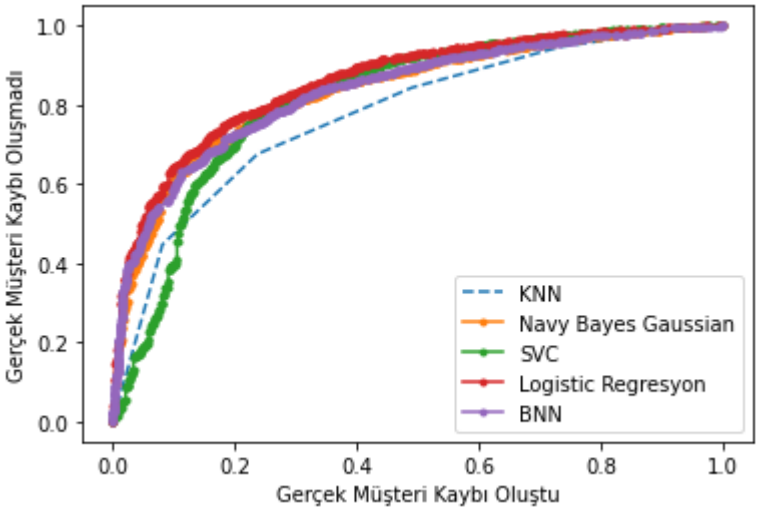


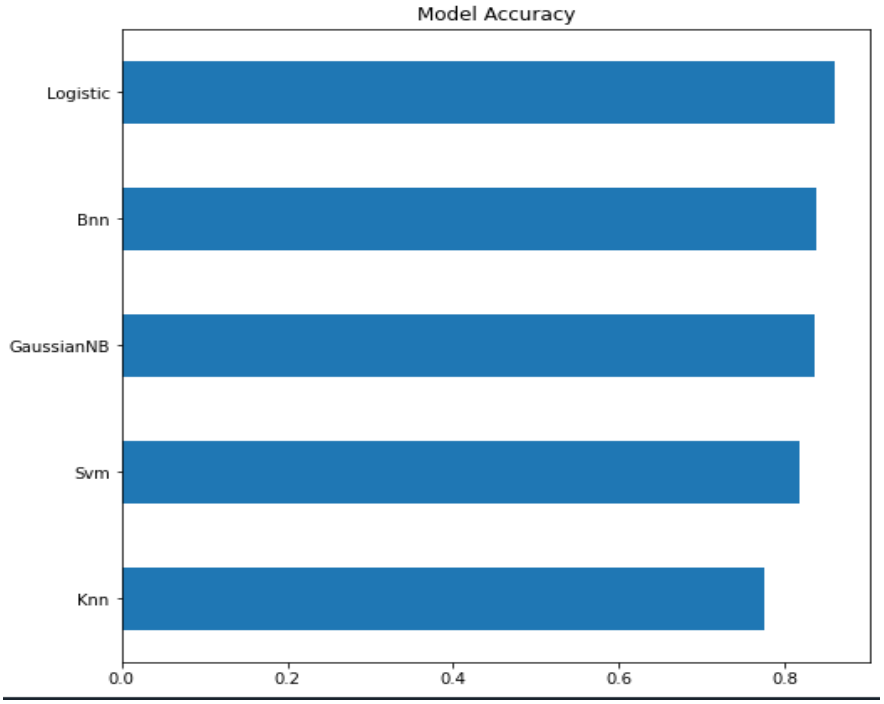
-Müşteri kaybının grafiğe dökümü gösterilmiştir.





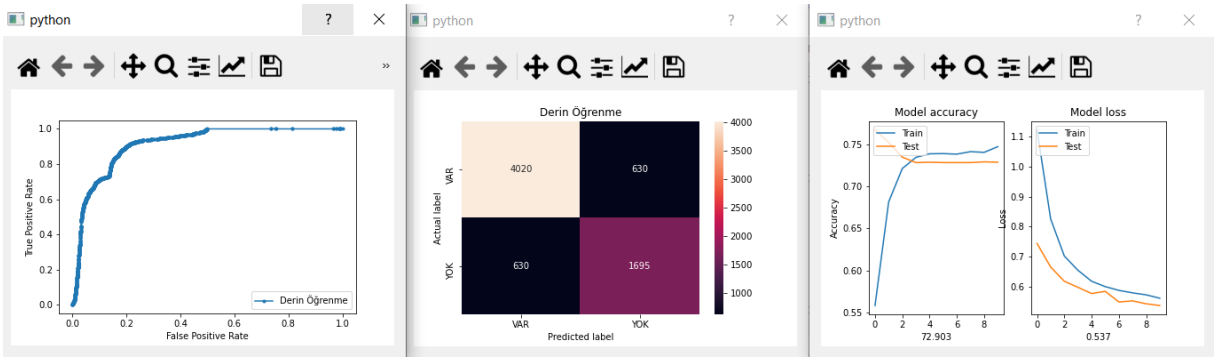
-Bütün özniteliklerin müşteri kaybı ile bağlantısının grafiğe dökümü gösterilmiştir.





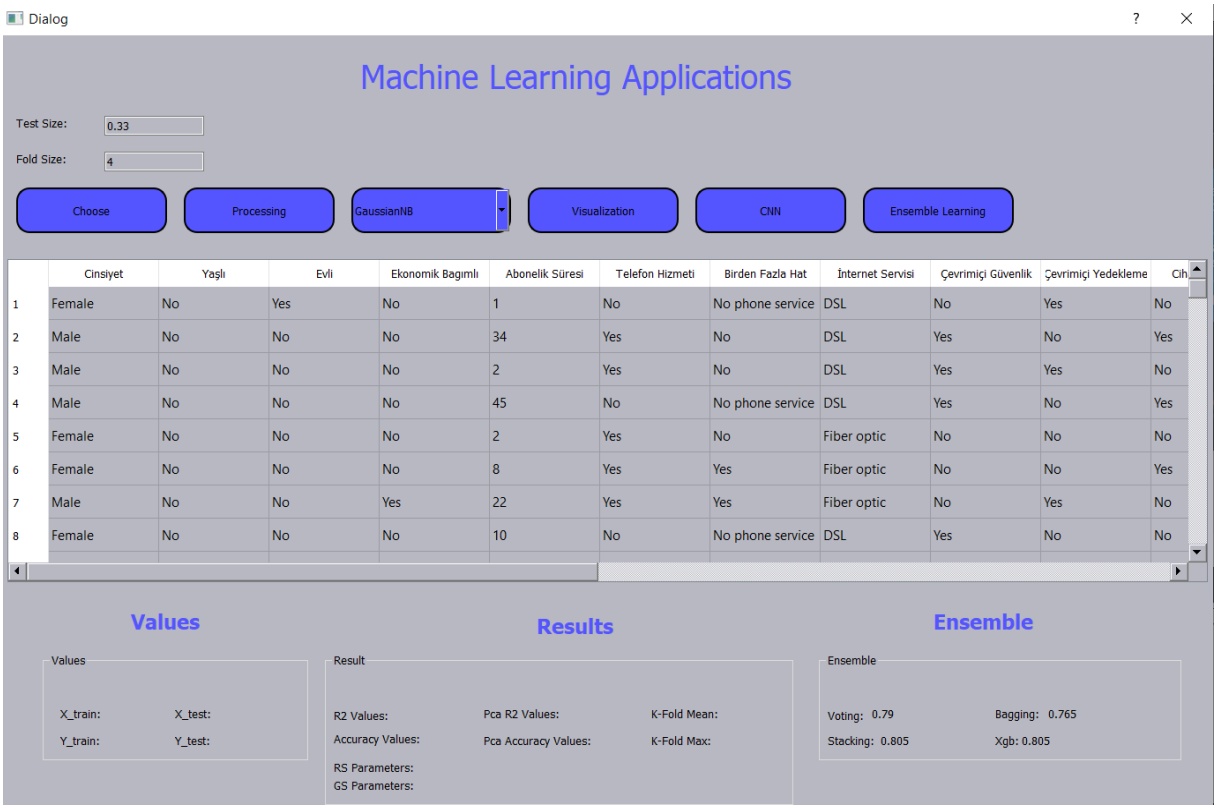
-Bütün algoritmaların roc eğrisi gösterimi ve başarı analizi yapılmıştır.

3) Cnn Butonuna Basıldığında Yürtülen İşlem



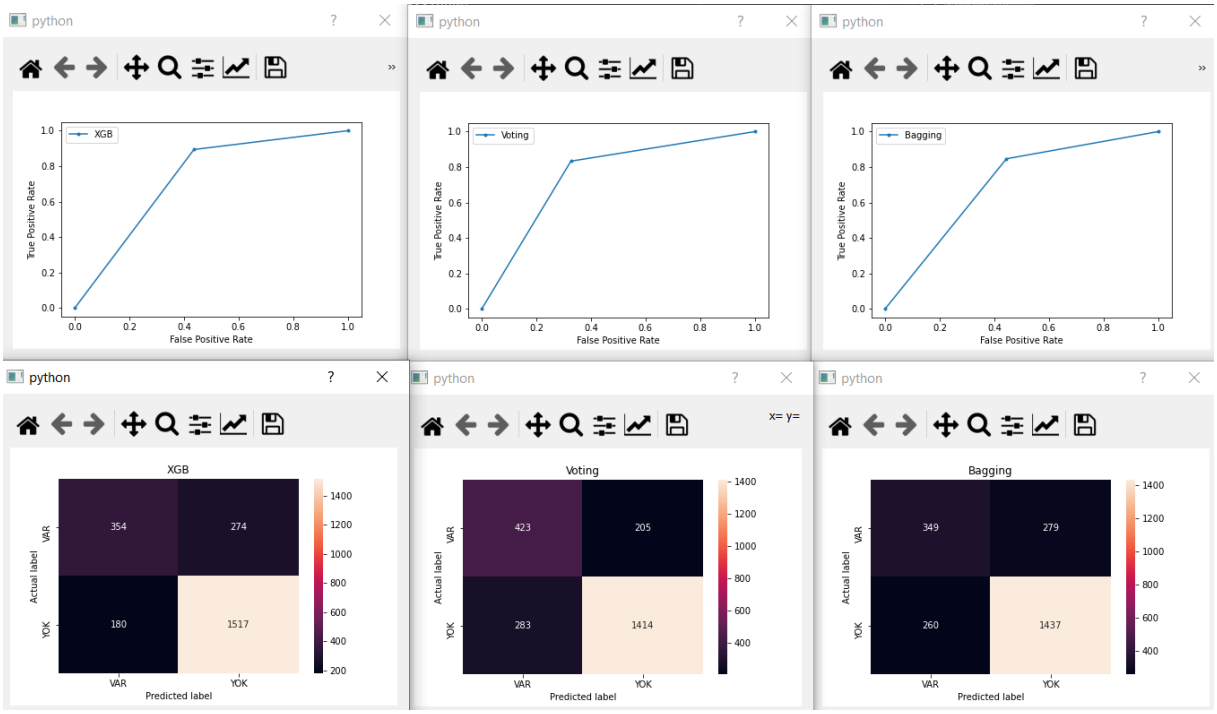
-YSA algoritmasıyla Roc eğrisi, Confusion matrix, accuracy ve loss grafikleri gösterilmiştir.

4) Ensemble Butonuna Basıldığında Yürtülen İşlem

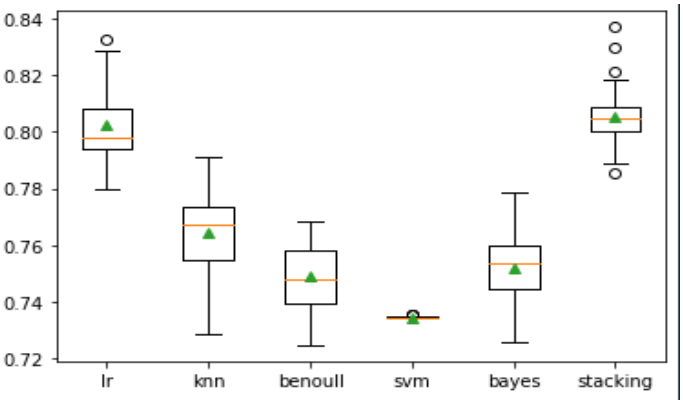


	Cinsiyet	Yaşlı	Evli	Ekonomik Bagımlı	Abonelik Süresi	Telefon Hizmeti	Birden Fazla Hat	İnternet Servisi	Çevrimiçi Güvenlik	Çevrimiçi Yedekleme	Cih
1	Female	No	Yes	No	1	No	No phone service	DSL	No	Yes	No
2	Male	No	No	No	34	Yes	No	DSL	Yes	No	Yes
3	Male	No	No	No	2	Yes	No	DSL	Yes	Yes	No
4	Male	No	No	No	45	No	No phone service	DSL	Yes	No	Yes
5	Female	No	No	No	2	Yes	No	Fiber optic	No	No	No
6	Female	No	No	No	8	Yes	Yes	Fiber optic	No	No	Yes
7	Male	No	No	Yes	22	Yes	Yes	Fiber optic	No	Yes	No
8	Female	No	No	No	10	No	No phone service	DSL	Yes	No	No

-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,B
NN_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.