



KASTAMONU ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
MAKİNE ÖĞRENMESİ DERSİ ÖDEV RAPORU

ÖDEV

MAKİNE ÖĞRENMESİ UYGULAMALARI

ÖDEV TARİHİ

11.01.2021

DERSİN SORUMLUSU

KEMAL AKYOL

RAPORU YAZAN ÖĞRENCİ

174410041 - SEVİLAY BULUT

ÖDEVİMDE KULLANDIĞIM KÜTÜPHANELER

#TABLO

```
from PyQt5.QtWidgets import QApplication, QTableWidgetItem, QMainWindow, QWidget,
QInputDialog, QLineEdit, QFileDialog, QLabel, QTextEditor, QGridLayout
import sys
import tasarimUI
import pandas as pd
import csv
from sklearn.model_selection import train_test_split
```

#VRİSETİ İŞLEMLERİ

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import pyqtSlot
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QInputDialog, QLineEdit,
QFileDialog
from PyQt5.QtWidgets import
QTextEditor, QLabel, QPushButton, QVBoxLayout, QHBoxLayout, QTableWidgetItem,
QTableWidget,
QTableWidgetI
tem, QVBoxLayout
from PyQt5.QtGui import QIcon
import os
import matplotlib.pyplot as plt
import pandas as pd
from pandas.plotting import scatter_matrix
from sklearn import model_selection
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
```

#VOOTING

```
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.ensemble import VotingClassifier
```

#VERİSETİNİ AYIRMA

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold, train_test_split, cross_val_predict,
cross_val_score
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
```

#K-FOLD

```
import numpy as np
from sklearn.model_selection import KFold
```

#ROC

```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
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
```

#GRAFİKLER

```
from PyQt5.QtGui import QPixmap
```

#randomcvc

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform
```

#metrikler

```
import math
```

#OVERSAMPLİNG-UNDERSAMPLİNG

```
from collections import Counter
from sklearn.datasets import make_classification
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler
```

KULLANDIĞIM PUSHBUTTONLARIN CLİCKED OLAYLARINDAKİ FONKSİYONLAR

```
class Pencere(QMainWindow, tasarimUI.Ui_MainWindow):
```

```
    def __init__(self):
```

```
        super().__init__()
```

```
        self.setupUi(self)
```

```
        self.pushButton.clicked.connect(self.yukle)
```

```
        self.pushButton_8.clicked.connect(self.yukle1)
```

```
        self.pushButton_9.clicked.connect(self.yukle2)
```

```
        self.pushButton_10.clicked.connect(self.yukle3)
```

```
        self.pushButton_11.clicked.connect(self.yukle4)
```

```
        self.pushButton_12.clicked.connect(self.yukle5)
```

```
        self.pushButton_13.clicked.connect(self.yukle6)
```

```
        self.pushButton_14.clicked.connect(self.yukle7)
```

```
        self.pushButton_15.clicked.connect(self.yukle8)
```

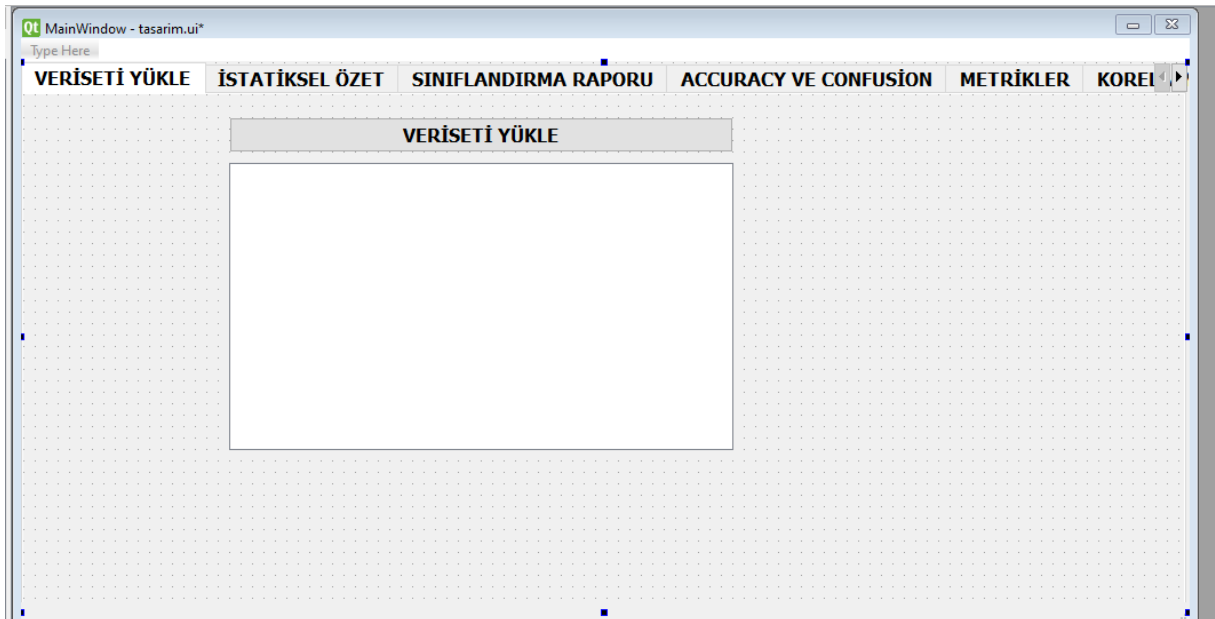
```
        self.pushButton_16.clicked.connect(self.yukle9)
```

```
        self.pushButton_17.clicked.connect(self.yukle10)
```

```
        self.pushButton_18.clicked.connect(self.yukle11)
```

```
self.pushButton_19.clicked.connect(self.yukle12)
self.pushButton_20.clicked.connect(self.yukle13)
self.pushButton_21.clicked.connect(self.yukle14)
self.pushButton_22.clicked.connect(self.yukle15)
self.pushButton_23.clicked.connect(self.yukle16)
self.pushButton_24.clicked.connect(self.yukle17)
self.pushButton_25.clicked.connect(self.yukle18)
self.pushButton_26.clicked.connect(self.yukle19)
self.pushButton_27.clicked.connect(self.yukle20)
self.pushButton_28.clicked.connect(self.yukle21)
self.pushButton_29.clicked.connect(self.yukle22)
self.pushButton_30.clicked.connect(self.yukle23)
self.pushButton_31.clicked.connect(self.yukle24)
self.pushButton_32.clicked.connect(self.yukle25)
self.pushButton_33.clicked.connect(self.yukle26)
self.pushButton_34.clicked.connect(self.yukle27)
self.pushButton_35.clicked.connect(self.yukle28)
self.pushButton_36.clicked.connect(self.yukle29)
self.pushButton_37.clicked.connect(self.yukle30)
self.pushButton_39.clicked.connect(self.yukle31)
self.pushButton_40.clicked.connect(self.yukle32)
self.pushButton_41.clicked.connect(self.yukle33)
self.pushButton_42.clicked.connect(self.yukle34)
self.pushButton_43.clicked.connect(self.yukle35)
self.pushButton_38.clicked.connect(self.yukle36)
self.pushButton_44.clicked.connect(self.yukle37)
self.pushButton_45.clicked.connect(self.yukle38)
self.pushButton_46.clicked.connect(self.yukle39)
self.pushButton_47.clicked.connect(self.yukle40)
```

MAINWINDOW QT DESIGNER TASARIMIM



VERİSETİNİ TABLODA GÖSTERDİM

#VERİSETİ TABLODA GÖSTR

def yukle(self):

file,_ = QFileDialog.getOpenFileName(self, 'Open file', './',"CSV files (*.csv)")

#browse edip dosyayı import etmek için pencere

self.dataset_file_path = file

print(self.dataset_file_path)

self.dataset = pd.read_csv(self.dataset_file_path, engine='python')

self.dataset = self.dataset.values

print("yükleme=",len(self.dataset[0])) # Öz nitelik sayısı

self.tableWidget.clear()

self.tableWidget.setColumnCount(len(self.dataset[0]))

self.tableWidget.setRowCount(len(self.dataset))

for i,row **in** enumerate(self.dataset):

for j,cell **in** enumerate(row):

self.tableWidget.setItem(i,j, QTableWidgetItem(str(cell)))

self.tableWidget.horizontalHeader().setStretchLastSection(**True**)

self.tableWidget.resizeColumnsToContents() # table gösterim

def read_CSV(self,file): #verisetini okuyup listeye atıyoruz

with open(file, 'r') **as** csvFile: # okunabilir dosya

reader = csv.reader(csvFile)

for row **in** reader:

lines=[]

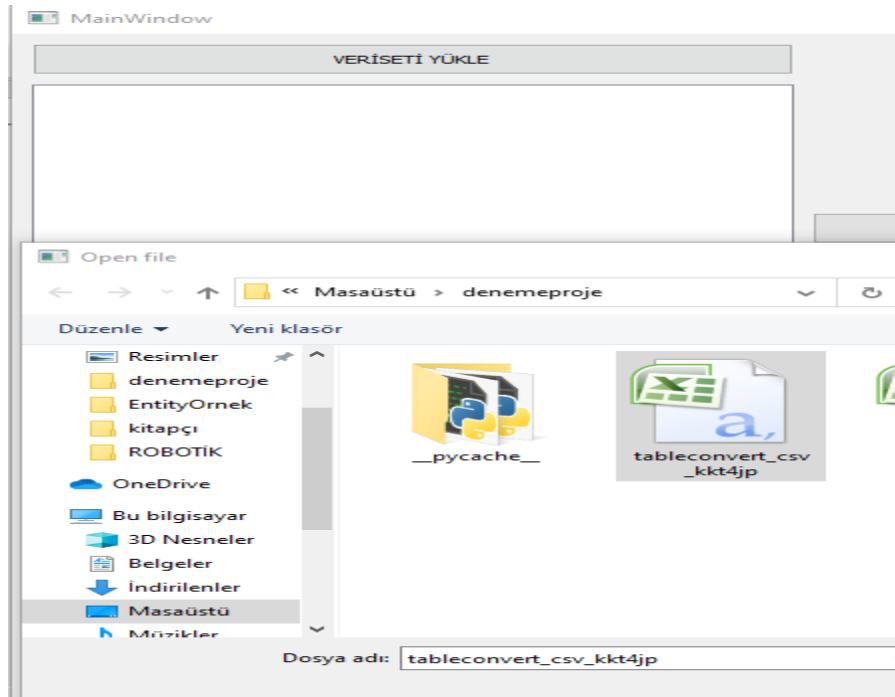
for value **in** row:

lines.append(str(round(float(value),3)))

self.dataset.append(lines)

csvFile.close()

VERİSETİNİ YÜKLEMEK İÇİN DOSYANIN YOLU



YÜKLENEN VERİ SETİNİN TABLODA GÖSTERİLMESİ

VERİSETİ YÜKLE					
	1	2	3	4	5
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa

Burada verisetinin verisetini yükle buton una tıklayınca tablewidget te veriseti gösterilecek. csv Excel dosyasını okutup tablewidget e aktardım

VERİSETİNİN İSTATİKSEL ÖZETİ

```
def yukle3(self):
    iris_dataset = pd.read_csv("iris.data")

    # Bağımlı ve bağımsız değişkenlerin oluşturulması
    X = iris_dataset.values[:, 0:4]
    Y = iris_dataset.values[:, 4]

    # Veri kümesinin eğitim ve test verileri olarak ayrılması
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

    #veri setinin istatistiksel özeti
    self.textEdit.setText(str(iris_dataset.describe()))
```

İSTATİKSEL ÖZET					
	5.1	3.5	1.4	0.2	
count	149.000000	149.000000	149.000000	149.000000	
mean	5.848322	3.051007	3.774497	1.205369	
std	0.828594	0.433499	1.759651	0.761292	
min	4.300000	2.000000	1.000000	0.100000	
25%	5.100000	2.800000	1.600000	0.300000	
50%	5.800000	3.000000	4.400000	1.300000	

Describe() metodu ile veri setindeki her sutunun istatistiksel özet butonuna tıklayınca istatistiksel özeti belirledim.

SINIFLANDIRMA RAPORLARI(f-1 score, precision, recall, support)

SVM İÇİN

```
#svm için
#SINIFLANDIRMA RAPORU
def yukle8(self):
    iris_dataset = pd.read_csv("iris.data")

    # Bağımlı ve bağımsız değişkenlerin oluşturulması
    X = iris_dataset.values[:, 0:4]
    Y = iris_dataset.values[:, 4]

    # Veri kümesinin eğitim ve test verileri olarak ayrılması
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

    # Modellerin listesinin olusturulmasi
    models = [
        ("SVM", SVC())
    ]

    print("MODEL İNŞASI")
    # Modeller için 'cross validation' sonuçlarının yazdırılması
    results = []
    names = []
    for name, model in models:
        kfold = model_selection.KFold(n_splits=10, random_state=7)
        cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv=kfold,
scoring="accuracy")
        results.append(cv_results)
        names.append(name)
        print("%s: %f (%f)" % (name, cv_results.mean(), cv_results.std()))

    svc = SVC()
    svc.fit(X_train, Y_train)
    predictions = svc.predict(X_test)
    #SINIFLANDIRMA RAPORU
    self.textEdit_8.setText(str(classification_report(Y_test, predictions)))
```

LDA İÇİN

```
#LDA İÇİN
#SINIFLANDIRMA RAPORU
def yukle36(self):
    iris_dataset = pd.read_csv("iris.data")

    # Bağımlı ve bağımsız değişkenlerin oluşturulması
    X = iris_dataset.values[:, 0:4]
    Y = iris_dataset.values[:, 4]
```

```

# Veri kümesinin eğitim ve test verileri olarak ayrılması
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

# Modellerin listesinin olusturulmasi
models = [
    ("LR", LogisticRegression()),
    ("LDA", LinearDiscriminantAnalysis()),
    ("KNN", KNeighborsClassifier()),
    ("DT", DecisionTreeClassifier()),
    ("NB", GaussianNB()),
    ("SVM", SVC())
]

print("MODEL İNŞASI")
# Modeller için 'cross validation' sonuçlarının yazdırılması
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=7)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train,
cv=kfold, scoring="accuracy")
    results.append(cv_results)
    names.append(name)
    print("%s: %f (%f)" % (name, cv_results.mean(), cv_results.std()))

lda = LinearDiscriminantAnalysis()
lda.fit(X_train, Y_train)
predictions = lda.predict(X_test)

self.textEdit_46.setText(classification_report(Y_test, predictions))

```

LR İÇİN

```

#LR İÇİN
#SINIFLANDIRMA RAPORU
def yukle37(self):
    iris_dataset = pd.read_csv("iris.data")

    # Bağımlı ve bağımsız değişkenlerin oluşturulması
    X = iris_dataset.values[:, 0:4]
    Y = iris_dataset.values[:, 4]

    # Veri kümesinin eğitim ve test verileri olarak ayrılması
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

    # Modellerin listesinin olusturulmasi
    models = [
        ("LR", LogisticRegression()),

```



```

        ("LDA", LinearDiscriminantAnalysis()),
        ("KNN", KNeighborsClassifier()),
        ("DT", DecisionTreeClassifier()),
        ("NB", GaussianNB()),
        ("SVM", SVC())
    ]

    print("MODEL İNŞASI")
    # Modeller için 'cross validation' sonuçlarının yazdırılması
    results = []
    names = []
    for name, model in models:
        kfold = model_selection.KFold(n_splits=10, random_state=7)
        cv_results = model_selection.cross_val_score(model, X_train, Y_train,
cv=kfold, scoring="accuracy")
        results.append(cv_results)
        names.append(name)
        print("%s: %f (%f)" % (name, cv_results.mean(), cv_results.std()))

    lr = LogisticRegression()
    lr.fit(X_train, Y_train)
    predictions = lr.predict(X_test)

    self.textEdit_47.setText(classification_report(Y_test, predictions))

```

KNN İÇİN

```

#knn İÇİN
#SINIFLANDIRMA RAPORU
def yukle38(self):
    iris_dataset = pd.read_csv("iris.data")

    # Bağımlı ve bağımsız değişkenlerin oluşturulması
    X = iris_dataset.values[:, 0:4]
    Y = iris_dataset.values[:, 4]

    # Veri kümesinin eğitim ve test verileri olarak ayrılması
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

    # Modellerin listesinin oluşturulması
    models = [
        ("LR", LogisticRegression()),
        ("LDA", LinearDiscriminantAnalysis()),
        ("KNN", KNeighborsClassifier()),
        ("DT", DecisionTreeClassifier()),
        ("NB", GaussianNB()),
        ("SVM", SVC())
    ]

```

```

print("MODEL İNŞASI")
# Modeller için 'cross validation' sonuçlarının yazdırılması
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=7)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train,
cv=kfold, scoring="accuracy")
    results.append(cv_results)
    names.append(name)
    print("%s: %f (%f)" % (name, cv_results.mean(), cv_results.std()))

knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)
predictions = knn.predict(X_test)

self.textEdit_48.setText(classification_report(Y_test, predictions))

```

DT İÇİN

```

#DT İÇİN İÇİN
#SINIFLANDIRMA RAPORU
def yukle39(self):
    iris_dataset = pd.read_csv("iris.data")

    # Bağımlı ve bağımsız değişkenlerin oluşturulması
    X = iris_dataset.values[:, 0:4]
    Y = iris_dataset.values[:, 4]

    # Veri kümesinin eğitim ve test verileri olarak ayrılması
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

    # Modellerin listesinin olusturulmasi
    models = [
        ("LR", LogisticRegression()),
        ("LDA", LinearDiscriminantAnalysis()),
        ("KNN", KNeighborsClassifier()),
        ("DT", DecisionTreeClassifier()),
        ("NB", GaussianNB()),
        ("SVM", SVC())
    ]

    print("MODEL İNŞASI")
    # Modeller için 'cross validation' sonuçlarının yazdırılması
    results = []
    names = []
    for name, model in models:
        kfold = model_selection.KFold(n_splits=10, random_state=7)

```

```

        cv_results = model_selection.cross_val_score(model, X_train, Y_train,
cv=kfold, scoring="accuracy")
        results.append(cv_results)
        names.append(name)
        print("%s: %f (%f)" % (name, cv_results.mean(), cv_results.std()))

        dt = DecisionTreeClassifier()
        dt.fit(X_train, Y_train)
        predictions = dt.predict(X_test)

self.textEdit_49.setText(classification_report(Y_test, predictions))

```

NB İÇİN

```

#NB İÇİN İÇİN
#SINIFLANDIRMA RAPORU
def yukle40(self):
    iris_dataset = pd.read_csv("iris.data")

    # Bağımlı ve bağımsız değişkenlerin oluşturulması
    X = iris_dataset.values[:, 0:4]
    Y = iris_dataset.values[:, 4]

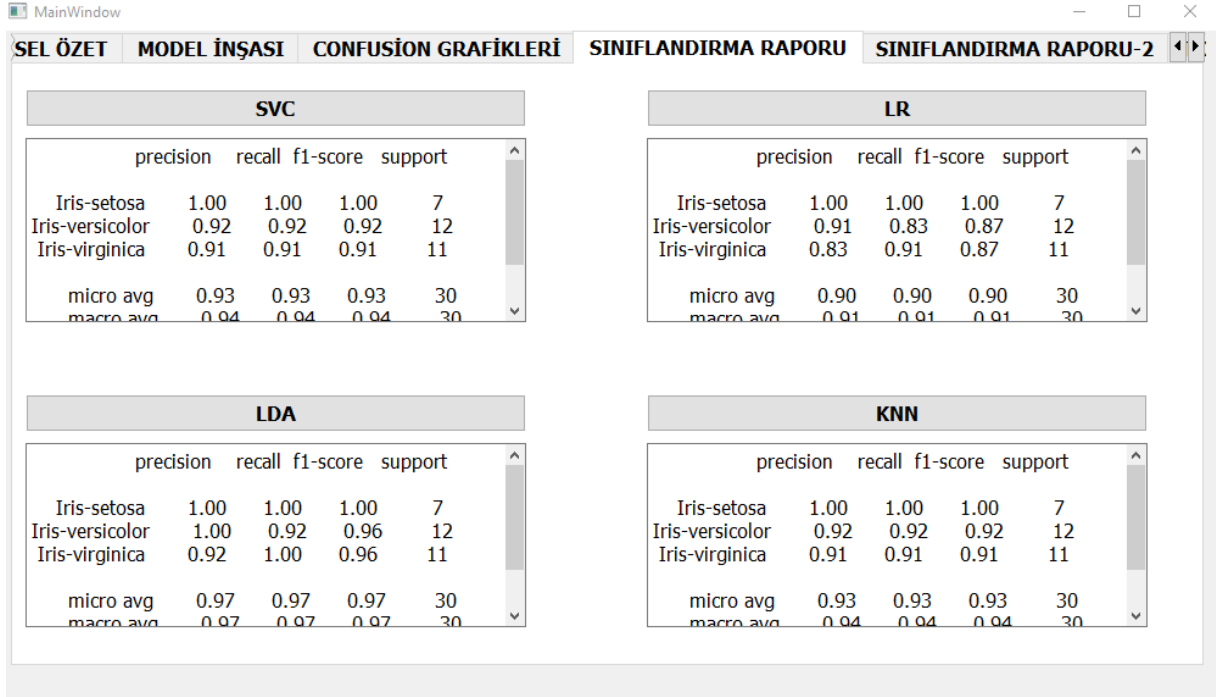
    # Veri kümesinin eğitim ve test verileri olarak ayrılması
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

    # Modellerin listesinin olusturulması
    models = [
        ("LR", LogisticRegression()),
        ("LDA", LinearDiscriminantAnalysis()),
        ("KNN", KNeighborsClassifier()),
        ("DT", DecisionTreeClassifier()),
        ("NB", GaussianNB()),
        ("SVM", SVC())
    ]

    print("MODEL İNŞASI")
    # Modeller için 'cross validation' sonuçlarının yazdırılması
    results = []
    names = []
    for name, model in models:
        kfold = model_selection.KFold(n_splits=10, random_state=7)
        cv_results = model_selection.cross_val_score(model, X_train, Y_train,
cv=kfold, scoring="accuracy")
        results.append(cv_results)
        names.append(name)
        print("%s: %f (%f)" % (name, cv_results.mean(), cv_results.std()))
        nb = GaussianNB()
        nb.fit(X_train, Y_train)

```

```
predictions = nb.predict(X_test)
self.textEdit_50.setText(classification_report(Y_test, predictions))
```



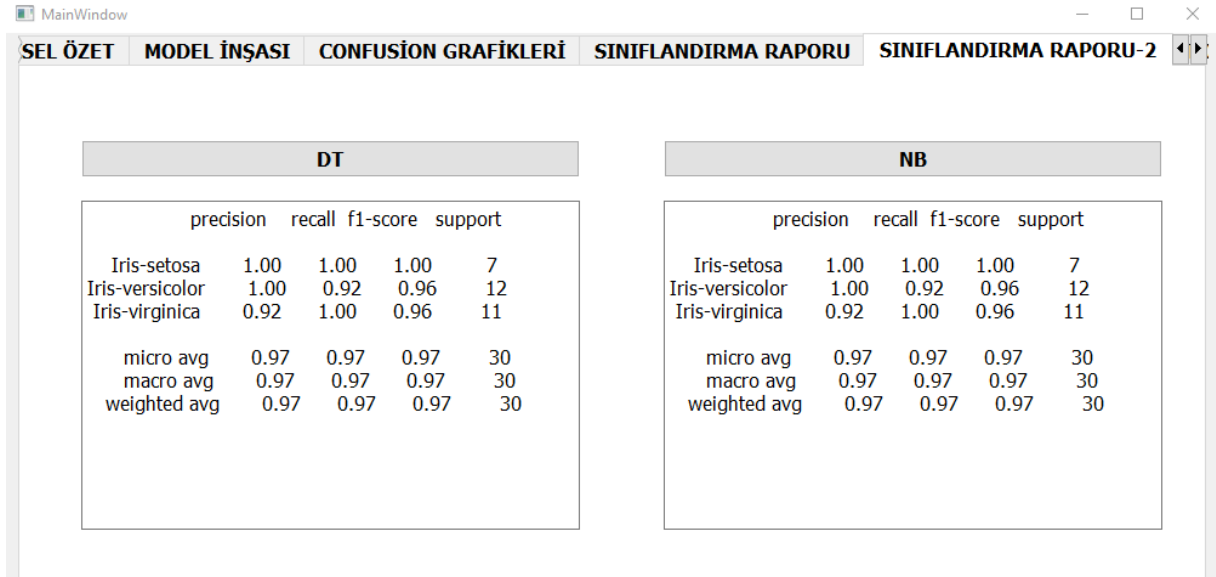
The screenshot shows a software window titled 'MainWindow' with a tabbed interface. The 'SINIFLANDIRMA RAPORU-2' tab is active, displaying four classification reports for different models: SVC, LR, LDA, and KNN. Each report is presented in a table with columns for precision, recall, f1-score, and support for each class (Iris-setosa, Iris-versicolor, Iris-virginica) and overall averages (micro avg, macro avg).

SVC				
	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	7
Iris-versicolor	0.92	0.92	0.92	12
Iris-virginica	0.91	0.91	0.91	11
micro avg	0.93	0.93	0.93	30
macro avg	0.94	0.94	0.94	30

LR				
	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	7
Iris-versicolor	0.91	0.83	0.87	12
Iris-virginica	0.83	0.91	0.87	11
micro avg	0.90	0.90	0.90	30
macro avg	0.91	0.91	0.91	30

LDA				
	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	7
Iris-versicolor	1.00	0.92	0.96	12
Iris-virginica	0.92	1.00	0.96	11
micro avg	0.97	0.97	0.97	30
macro avg	0.97	0.97	0.97	30

KNN				
	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	7
Iris-versicolor	0.92	0.92	0.92	12
Iris-virginica	0.91	0.91	0.91	11
micro avg	0.93	0.93	0.93	30
macro avg	0.94	0.94	0.94	30



The screenshot shows the same software window with the 'SINIFLANDIRMA RAPORU-2' tab active, displaying classification reports for Decision Tree (DT) and Naive Bayes (NB) models. The reports are presented in tables with columns for precision, recall, f1-score, and support for each class and overall averages.

DT				
	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	7
Iris-versicolor	1.00	0.92	0.96	12
Iris-virginica	0.92	1.00	0.96	11
micro avg	0.97	0.97	0.97	30
macro avg	0.97	0.97	0.97	30
weighted avg	0.97	0.97	0.97	30

NB				
	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	7
Iris-versicolor	1.00	0.92	0.96	12
Iris-virginica	0.92	1.00	0.96	11
micro avg	0.97	0.97	0.97	30
macro avg	0.97	0.97	0.97	30
weighted avg	0.97	0.97	0.97	30

Modellerin sınıflandırma raporlarına f-1 score, precision, recall, support, oranlarını gösterdim

İLK 5 SATIR, EKSİK VERİ SAYDIRMA, BOŞ SUTUNLARI SİLDİRME, ÖZNİTELİKLERİN ARTİMETİK ORTALAMASI, İSTENİLEN KOLONU VERİSETİNDEN ÇIKAR, VERİSETİNİN BOTUTU

İLK 5 SATIRI HEAD() METODU İLE LİSTELETTİM

```
def yukle1(self):
    iris_dataset = pd.read_csv("iris.data")

    # Bağımlı ve bağımsız değişkenlerin oluşturulması
    X = iris_dataset.values[:, 0:4]
    Y = iris_dataset.values[:, 4]

    # Veri kümesinin eğitim ve test verileri olarak ayrılması
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

    # İLK 5 SATIR
    self.textEdit_2.setText(str(iris_dataset.head()))
```

EKSİK VERİLERİ ISNULL().SUM() İLE SAYDIRDIM

```
def yukle4(self):
    iris_dataset = pd.read_csv("iris.data")

    # Bağımlı ve bağımsız değişkenlerin oluşturulması
    X = iris_dataset.values[:, 0:4]
    Y = iris_dataset.values[:, 4]

    # Veri kümesinin eğitim ve test verileri olarak ayrılması
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

    # eksik verinin kolon bazında saydırılması
    self.textEdit_3.setText(str(iris_dataset.isnull().sum()))
```

İSTENİLEN KOLONU SİLME

```
def yukle5(self):
    iris_dataset = pd.read_csv("iris.data")

    # Bağımlı ve bağımsız değişkenlerin oluşturulması
    X = iris_dataset.values[:, 0:4]
    Y = iris_dataset.values[:, 4]
```

```
# Veri kümesinin eğitim ve test verileri olarak ayrılması
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

#istenilen satırı silme
self.textEdit_4.setText(str(iris_dataset.drop(columns="Iris-setosa")))
```

ARİTMETİK ORTALAMA

```
def yukle9(self):
    iris_dataset = pd.read_csv("iris.data")

    # Bağımlı ve bağımsız değişkenlerin oluşturulması
    X = iris_dataset.values[:, 0:4]
    Y = iris_dataset.values[:, 4]

    # Veri kümesinin eğitim ve test verileri olarak ayrılması
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

    #ARİTMETİK ORTAMA
    self.textEdit_9.setText(str(iris_dataset.mean(axis = 0, skipna = True)))
```

BOŞ SUTUNLARI SİLDİRDİM

```
def yukle2(self):
    iris_dataset = pd.read_csv("iris.data")

    # Bağımlı ve bağımsız değişkenlerin oluşturulması
    X = iris_dataset.values[:, 0:4]
    Y = iris_dataset.values[:, 4]

    # Veri kümesinin eğitim ve test verileri olarak ayrılması
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

    #♣boş sütunları silme
    self.textEdit_5.setText(str(iris_dataset.dropna()))
```

VERİSETİNİN BOYUTU

```
#VERİSETİNİN BOYUTU
def yukle24(self):
    iris_dataset = pd.read_csv("iris.data")

    # Bağımlı ve bağımsız değişkenlerin oluşturulması
```

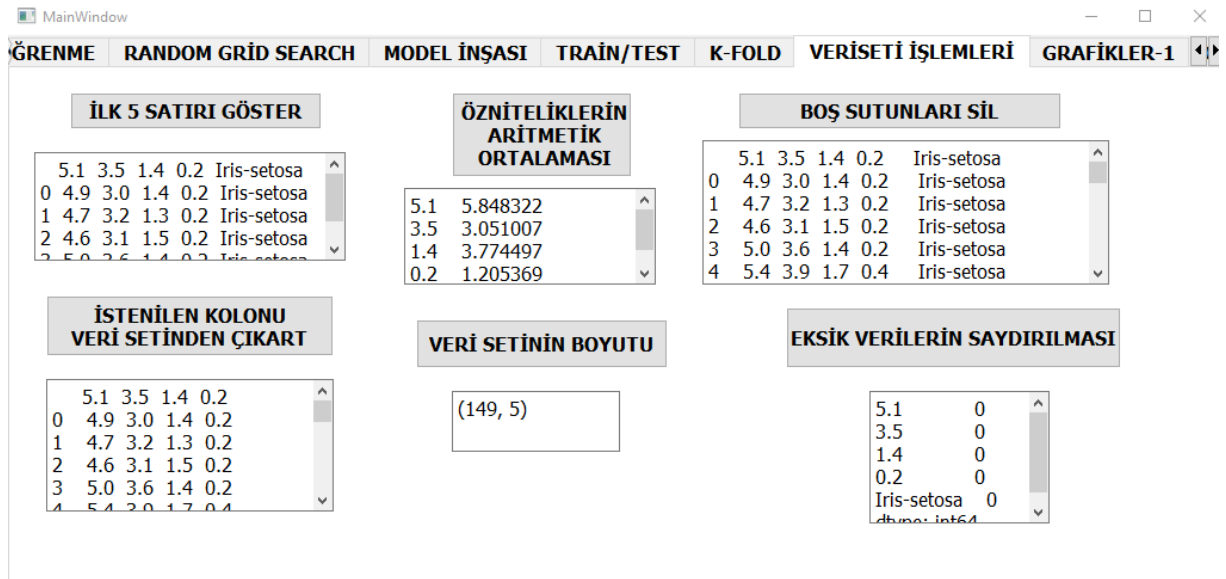
```

X = iris_dataset.values[:, 0:4]
Y = iris_dataset.values[:, 4]

# Veri kümesinin eğitim ve test verileri olarak ayrılması
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

print("VERİ SETİNİN BOYUTU")
print(iris_dataset.shape)
self.textEdit_24.setText(str(iris_dataset.shape))

```



KORELASYON MATRİSİ

```

def ykle10(self):
    iris_dataset = pd.read_csv("iris.data")

    # Bağımlı ve bağımsız değişkenlerin oluşturulması
    X = iris_dataset.values[:, 0:4]
    Y = iris_dataset.values[:, 4]

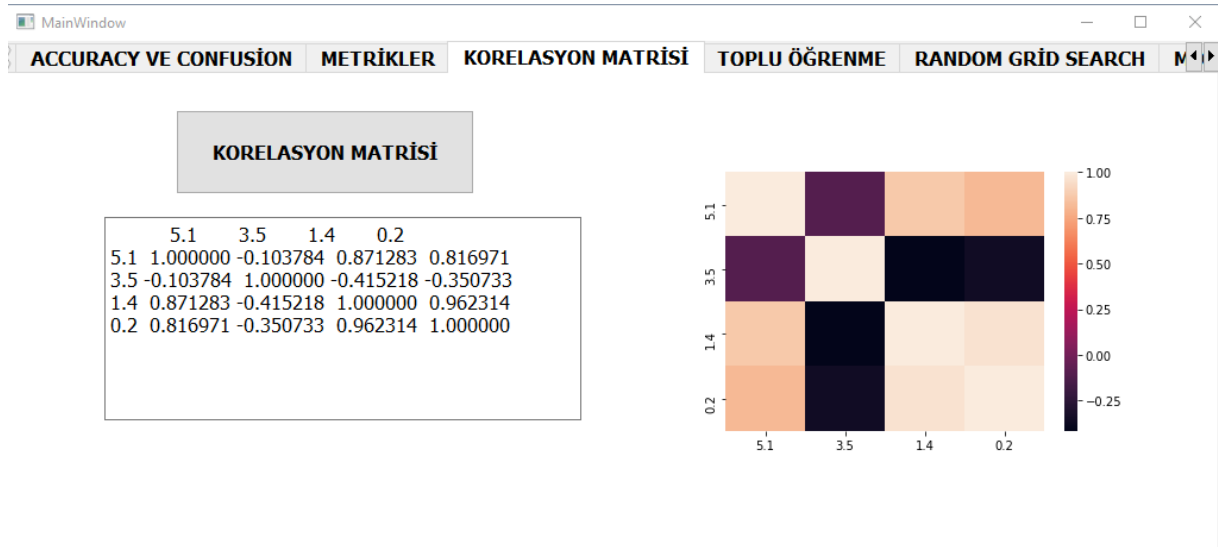
    # Veri kümesinin eğitim ve test verileri olarak ayrılması
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

    # korelasyon hesaplama
    self.textEdit_10.setText(str(iris_dataset.corr()))
    # 5iris_dataset.plot(x='5.1', y='3.5', style='*')

    corr = iris_dataset.corr()

    sns.heatmap(corr,xticklabels=corr.columns.values,yticklabels=corr.columns.values)

```



Korelasyon matrisini hesaplatım grafikte ifade ettim.

MODEL İNŞASI(ACCURACY, CONFUSİON MATRİS, CONFUSİON GRAFİKLERİ STANDART SAPMA VE ORTALAMALARI)

SVM İÇİN

```
#MODEL İNŞASI SVM İÇİN
def yukle25(self):
    iris_dataset = pd.read_csv("iris.data")

    # Bağımlı ve bağımsız değişkenlerin oluşturulması
    X = iris_dataset.values[:, 0:4]
    Y = iris_dataset.values[:, 4]

    # Veri kümesinin eğitim ve test verileri olarak ayrılması
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

    #SVM İÇİN
    # Modellerin listesinin olusturulmasi
    models = [
        ("LR", LogisticRegression()),
        ("LDA", LinearDiscriminantAnalysis()),
        ("KNN", KNeighborsClassifier()),
        ("DT", DecisionTreeClassifier()),
        ("NB", GaussianNB()),
        ("SVM", SVC())
    ]
```



```

print("MODEL İNŞASI")
# Modeller için 'cross validation' sonuçlarının yazdırılması
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=7)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train,
cv=kfold, scoring="accuracy")
    results.append(cv_results)
    names.append(name)

    svc = SVC()
    svc.fit(X_train, Y_train)
    predictions = svc.predict(X_test)

    print("%s: %f (%f)" % (name, cv_results.mean(), cv_results.std()))
    self.textEdit_25.setText(str("%s: %f (%f)" % (name,
cv_results.mean(), cv_results.std()))))
    self.label_24.setText(str(accuracy_score(Y_test, predictions)))
    self.textEdit_38.setText(str(confusion_matrix(Y_test, predictions)))

    # karmaşıklık matrisi
    print(confusion_matrix(Y_test, predictions))
    # grafik
    conf_mat = confusion_matrix(Y_test, predictions)
    labels = ['Class 0', 'Class 1']
    fig = plt.figure()
    ax = fig.add_subplot(111)
    cax = ax.matshow(conf_mat, cmap=plt.cm.Blues)
    fig.colorbar(cax)
    ax.set_xticklabels([""] + labels)
    ax.set_yticklabels([""] + labels)
    plt.xlabel('Tahmin')
    plt.ylabel('Beklenen')
    plt.show()

```

LR İÇİN

```

# MODEL İNŞASI LR İÇİN
def yukle33(self):
    iris_dataset = pd.read_csv("iris.data")

    # Bağımlı ve bağımsız değişkenlerin oluşturulması
    X = iris_dataset.values[:, 0:4]
    Y = iris_dataset.values[:, 4]

    # Veri kümesinin eğitim ve test verileri olarak ayrılması
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

```

```

#LR İÇİN
models = [
    ("LR", LogisticRegression())
]

print("MODEL İNŞASI")
# Modeller için 'cross validation' sonuçlarının yazdırılması
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=7)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train,
cv=kfold, scoring="accuracy")
    results.append(cv_results)
    names.append(name)

    lr = LogisticRegression()
    lr.fit(X_train, Y_train)
    predictions = lr.predict(X_test)

    self.textEdit_30.setText(str("%s: %f (%f)" % (name,
cv_results.mean(), cv_results.std()))))
    self.label_21.setText(str(accuracy_score(Y_test, predictions)))
    self.textEdit_39.setText(str(confusion_matrix(Y_test, predictions)))

# karmaşıklık matrisi
print(confusion_matrix(Y_test, predictions))
# grafik
conf_mat = confusion_matrix(Y_test, predictions)
labels = ['Class 0', 'Class 1']
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(conf_mat, cmap=plt.cm.Blues)
fig.colorbar(cax)
ax.set_xticklabels([""] + labels)
ax.set_yticklabels([""] + labels)
plt.xlabel('Tahmin')
plt.ylabel('Beklenen')
plt.show()

```

LDA İÇİN

```

#MODEL İNŞASI LDA İÇİN
def yuikle34(self):
    iris_dataset = pd.read_csv("iris.data")

    # Bağımlı ve bağımsız değişkenlerin oluşturulması
    X = iris_dataset.values[:, 0:4]
    Y = iris_dataset.values[:, 4]

```

```

# Veri kümesinin eğitim ve test verileri olarak ayrılması
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)
#LDA İÇİN
models = [
    ("LDA", LinearDiscriminantAnalysis())
]

print("MODEL İNŞASI")
# Modeller için 'cross validation' sonuçlarının yazdırılması
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=7)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train,
cv=kfold, scoring="accuracy")
    results.append(cv_results)
    names.append(name)

    lda = LinearDiscriminantAnalysis()
    lda.fit(X_train, Y_train)
    predictions = lda.predict(X_test)

    self.textEdit_31.setText(str("%s: %f (%f)" % (name,
cv_results.mean(), cv_results.std()))))
    self.label_22.setText(str(accuracy_score(Y_test, predictions)))
    self.textEdit_42.setText(str(confusion_matrix(Y_test, predictions)))

# karmaşıklık matrisi
print(confusion_matrix(Y_test, predictions))
# grafik
conf_mat = confusion_matrix(Y_test, predictions)
labels = ['Class 0', 'Class 1']
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(conf_mat, cmap=plt.cm.Blues)
fig.colorbar(cax)
ax.set_xticklabels([""] + labels)
ax.set_yticklabels([""] + labels)
plt.xlabel('Tahmin')
plt.ylabel('Beklenen')
plt.show()

```

DT İÇİN

```

#MODEL İNŞASI DT İÇİN
def yuikle35(self):
    iris_dataset = pd.read_csv("iris.data")

    # Bağımlı ve bağımsız değişkenlerin oluşturulması

```

```

X = iris_dataset.values[:, 0:4]
Y = iris_dataset.values[:, 4]

# Veri kümesinin eğitim ve test verileri olarak ayrılması
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)
#DT İÇİN
models = [
    ("DT", DecisionTreeClassifier())
]

print("MODEL İNŞASI")
# Modeller için 'cross validation' sonuçlarının yazdırılması
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=7)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train,
cv=kfold, scoring="accuracy")
    results.append(cv_results)
    names.append(name)

    dt = DecisionTreeClassifier()
    dt.fit(X_train, Y_train)
    predictions = dt.predict(X_test)

    self.textEdit_32.setText(str("%s: %f (%f)" % (name,
cv_results.mean(), cv_results.std()))))
    self.label_23.setText(str(accuracy_score(Y_test, predictions)))
    self.textEdit_43.setText(str(confusion_matrix(Y_test, predictions)))

# karmaşıklık matrisi
print(confusion_matrix(Y_test, predictions))
# grafik
conf_mat = confusion_matrix(Y_test, predictions)
labels = ['Class 0', 'Class 1']
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(conf_mat, cmap=plt.cm.Blues)
fig.colorbar(cax)
ax.set_xticklabels([""] + labels)
ax.set_yticklabels([""] + labels)
plt.xlabel('Tahmin')
plt.ylabel('Beklenen')
plt.show()

```

NB İÇİN

```

#MODEL İNŞASI NB İÇİN
def yukle32(self):
    iris_dataset = pd.read_csv("iris.data")

```

```

# Bağımlı ve bağımsız değişkenlerin oluşturulması
X = iris_dataset.values[:, 0:4]
Y = iris_dataset.values[:, 4]

# Veri kümesinin eğitim ve test verileri olarak ayrılması
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

#NB İÇİN
models = [
    ("NB", GaussianNB())
]

print("MODEL İNŞASI")
# Modeller için 'cross validation' sonuçlarının yazdırılması
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=7)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train,
cv=kfold, scoring="accuracy")
    results.append(cv_results)
    names.append(name)

    nb = GaussianNB()
    nb.fit(X_train, Y_train)
    predictions = nb.predict(X_test)

    self.textEdit_33.setText(str("%s: %f (%f)" % (name,
cv_results.mean(), cv_results.std()))))
    self.label_28.setText(str(accuracy_score(Y_test, predictions)))
    self.textEdit_41.setText(str(confusion_matrix(Y_test, predictions)))

# karmaşıklık matrisi
print(confusion_matrix(Y_test, predictions))
# grafik
conf_mat = confusion_matrix(Y_test, predictions)
labels = ['Class 0', 'Class 1']
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(conf_mat, cmap=plt.cm.Blues)
fig.colorbar(cax)
ax.set_xticklabels([""] + labels)
ax.set_yticklabels([""] + labels)
plt.xlabel('Tahmin')
plt.ylabel('Beklenen')
plt.show()

```

KNN İÇİN

```
#MODEL İNŞASI DT İÇİN
def yukle31(self):
    iris_dataset = pd.read_csv("iris.data")

    # Bağımlı ve bağımsız değişkenlerin oluşturulması
    X = iris_dataset.values[:, 0:4]
    Y = iris_dataset.values[:, 4]

    # Veri kümesinin eğitim ve test verileri olarak ayrılması
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

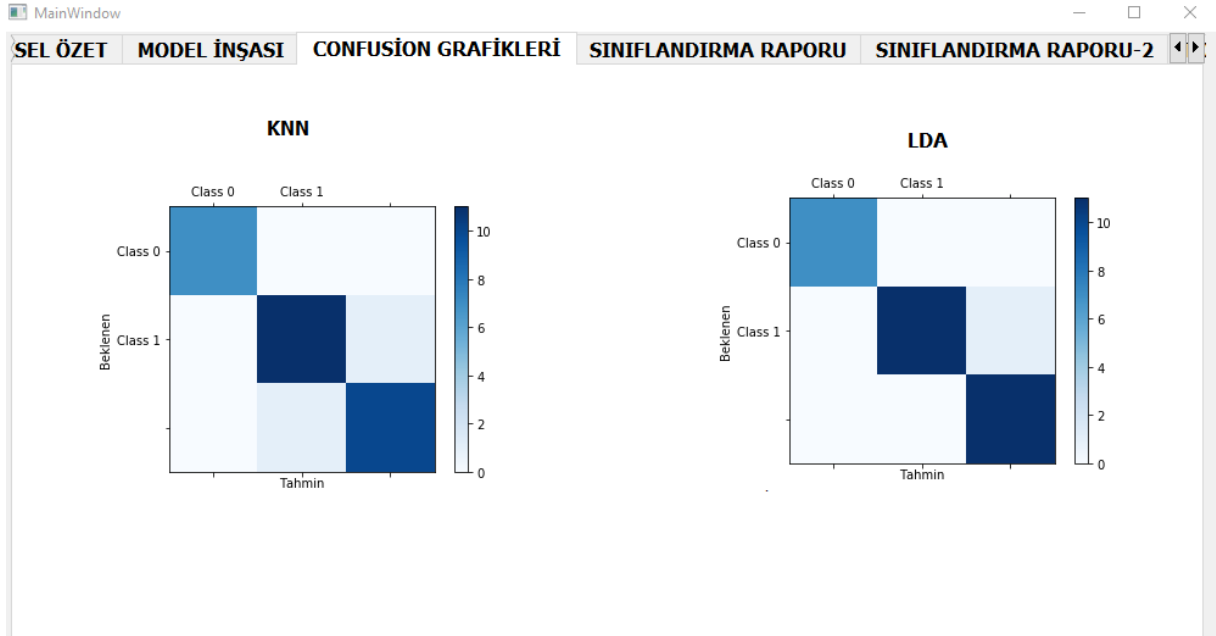
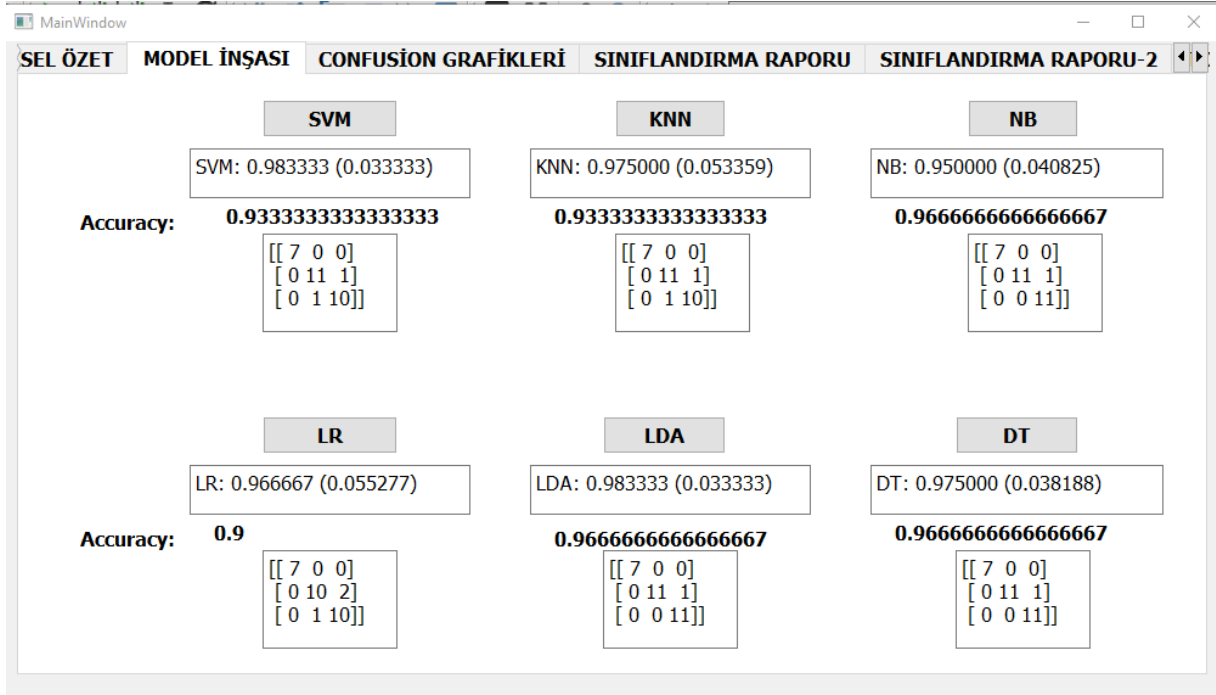
    #KNN İÇİN
    models = [
        ("KNN", KNeighborsClassifier())
    ]
    print("MODEL İNŞASI")
    # Modeller için 'cross validation' sonuçlarının yazdırılması
    results = []
    names = []
    for name, model in models:
        kfold = model_selection.KFold(n_splits=10, random_state=7)
        cv_results = model_selection.cross_val_score(model, X_train, Y_train,
cv=kfold, scoring="accuracy")
        results.append(cv_results)
        names.append(name)

        knn = KNeighborsClassifier()
        knn.fit(X_train, Y_train)
        predictions = knn.predict(X_test)

        self.textEdit_34.setText(str("%s: %f (%f)" % (name,
cv_results.mean(), cv_results.std()))))
        self.label_25.setText(str(accuracy_score(Y_test, predictions)))
        self.textEdit_40.setText(str(confusion_matrix(Y_test, predictions)))

    # karmaşıklık matrisi
    print(confusion_matrix(Y_test, predictions))
    # grafik
    conf_mat = confusion_matrix(Y_test, predictions)
    labels = ['Class 0', 'Class 1']
    fig = plt.figure()
    ax = fig.add_subplot(111)
    cax = ax.matshow(conf_mat, cmap=plt.cm.Blues)
    fig.colorbar(cax)
    ax.set_xticklabels([""] + labels)
    ax.set_yticklabels([""] + labels)
    plt.xlabel('Tahmin')
    plt.ylabel('Beklenen')
```

plt.show()



LDA,DT,NB,LR,KNN,SVM algoritmalarının accuracy değerlerini, confusion matrislerini ve grafiklerini ,ortalamalarını,standar sapmalarını hesaplattım

Accuracy ;bir modelin başarısını ölçmek için çok kullanılan ancak tek başına yeterli olmadığı görülen bir metriktir.

Karışıklık(confusion matrix); gerçek değerlerin bilinmekte olduğu bir dizi test verisi üzerinde, bir sınıflandırma modelinin performansını tanımlamak için sıklıkla kullanılan bir tablo modeli.(TP,NF,TN,TF) VE grafiğini gösterdim.

VERİSETLERİNİ HEM TRAIN/TEST OLARAK HEMDE K-FOLD İLE AYIRDIM

X-TRAIN

```
#VERİSETİ AYIRMA
def yukle15(self):

    iris_dataset=pd.read_csv('iris.data')
    iris_dataset

    x=iris_dataset.iloc[:, [2,3]].values
    y=iris_dataset.iloc[:, 4].values

    from sklearn.model_selection import train_test_split
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,
    random_state=0)

    #veri setini train ve test olarak ikiye ayırıyoruz
    x_train, x_test, y_train, y_test = train_test_split(x, y
    ,test_size=0.2
    ,shuffle = False
    ,random_state = 0)

    veri_kesitleri = {"x_train" : x_train
    , "x_test" : x_test
    , "y_train" : y_train
    , "y_test" : y_test}

    for i in veri_kesitleri:
        veri_kesitleri = {"x_train" : x_train}
        self.label_4.setText(str(f"{i}: satır sayısı {veri_kesitleri.get(i).shape[0]}"))
```


Y-TEST

```
def yukle16(self):
    iris_dataset=pd.read_csv('iris.data')
    iris_dataset

    x=iris_dataset.iloc[:, [2,3]].values
    y=iris_dataset.iloc[:, 4].values

    from sklearn.model_selection import train_test_split
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,
    random_state=0)

    #veri setini train ve test olarak ikiye ayırıyoruz
    x_train, x_test, y_train, y_test = train_test_split(x, y
    ,test_size=0.2
    ,shuffle = False
    ,random_state = 0)

    veri_kesitleri = {"x_train" : x_train
    , "x_test" : x_test
    , "y_train" : y_train
    , "y_test" : y_test}

    for i in veri_kesitleri:
        self.label.setText(str(f"{i}: satır sayısı {veri_kesitleri.get(i).shape[0]}"))
```

Y-TRAIN

```
def yukle18(self):
    iris_dataset=pd.read_csv('iris.data')
    iris_dataset

    x=iris_dataset.iloc[:, [2,3]].values
    y=iris_dataset.iloc[:, 4].values

    from sklearn.model_selection import train_test_split
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,
    random_state=0)

    #veri setini train ve test olarak ikiye ayırıyoruz
    x_train, x_test, y_train, y_test = train_test_split(x, y
    ,test_size=0.2
    ,shuffle = False
    ,random_state = 0)

    veri_kesitleri = {"y_train" : y_train}

    for i in veri_kesitleri:
        self.label_3.setText(str(f"{i}: satır sayısı {veri_kesitleri.get(i).shape[0]}"))
```

X-TEST

```
def yukle17(self):
    iris_dataset=pd.read_csv('iris.data')
    iris_dataset

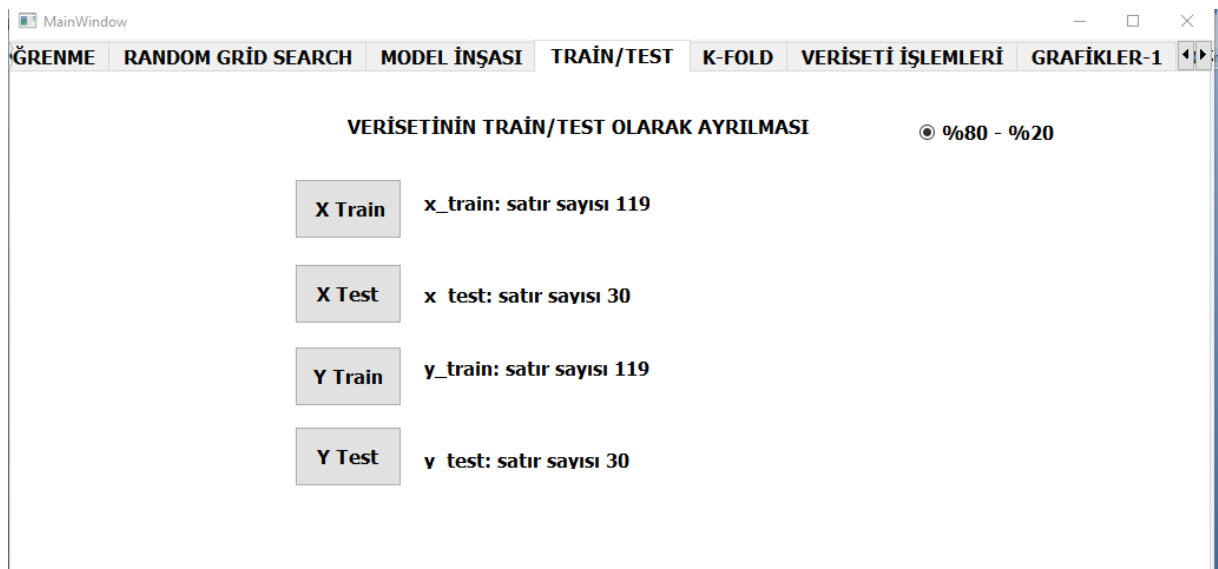
    x=iris_dataset.iloc[:, [2,3]].values
    y=iris_dataset.iloc[:, 4].values

    from sklearn.model_selection import train_test_split
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,
    random_state=0)

    #veri setini train ve test olarak ikiye ayırıyoruz
    x_train, x_test, y_train, y_test = train_test_split(x, y
    ,test_size=0.2
    ,shuffle = False
    ,random_state = 0)

    veri_kesitleri = {"x_test" : x_test}

    for i in veri_kesitleri:
        #veri_kesitleri = {"x_test" : x_test}
        self.label_2.setText(str(f"{i}: satır sayısı {veri_kesitleri.get(i).shape[0]}"))
```



K-FOLD İLE TRAIN/TEST OLARAK AYIRDIM

```
def yukle19(self):
    iris_dataset = pd.read_csv("iris.data")
```

```

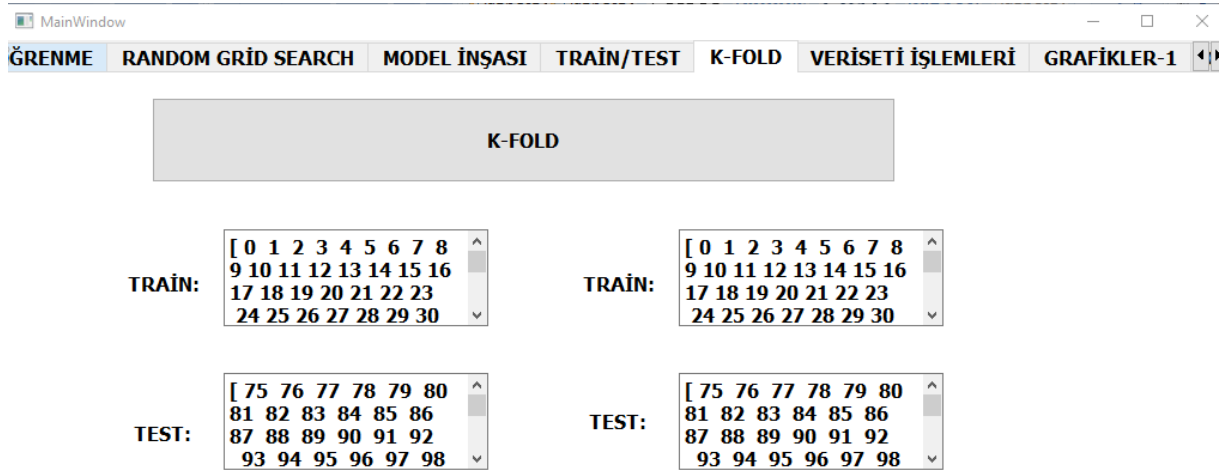
# Bağımlı ve bağımsız değişkenlerin oluşturulması
X = iris_dataset.values[:, 0:4]
Y = iris_dataset.values[:, 4]

# Veri kümesinin eğitim ve test verileri olarak ayrılması
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

kf = KFold(n_splits=2)
kf.get_n_splits(X)

print(kf)
KFold(n_splits=2, random_state=None, shuffle=False)
for train_index, test_index in kf.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    self.textEdit_15.setText(str(train_index))
    self.textEdit_16.setText(str(test_index))
    self.textEdit_17.setText(str(train_index))
    self.textEdit_18.setText(str(test_index))

```



K-fold; makine öğrenme modellerinin başarılarının değerlendirilmesi için kullanılan bir yöntemdir. Bu yöntemde veri seti eğitim ve test seti olarak ayrılmaktadır.(K-FOLD butonuna tıklayarak k-fold ile verisetlerini ayırdım)

Verisetlerini X/Y TRAIN/TEST butonlarına tıklayarak ayırdım.

RANDOM SEARCH VE GRID SEARCH (HİPERPARAMETRELER)

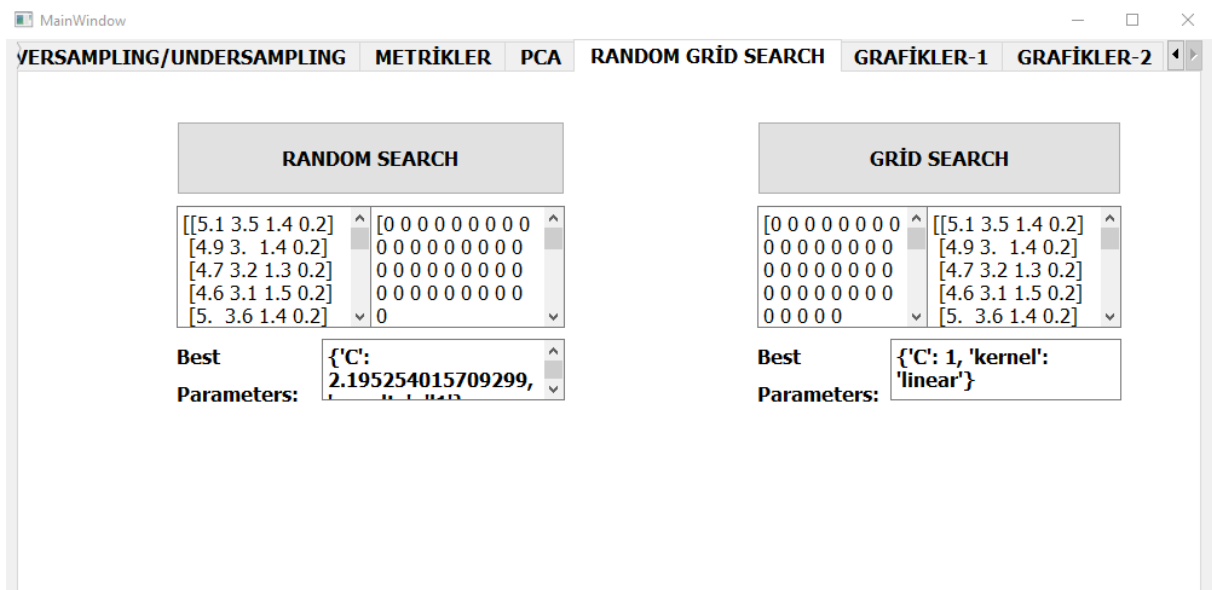
RANDOM SEARCH

```
#RANDOM SEARCH
def yukle21(self):
    iris = load_iris()
    logistic = LogisticRegression(solver='saga', tol=1e-2,
max_iter=200,random_state=0)
    distributions = dict(C=uniform(loc=0, scale=4),penalty=['l2', 'l1'])
    clf = RandomizedSearchCV(logistic, distributions, random_state=0)
    search = clf.fit(iris.data, iris.target)
    search.best_params_
    #print(iris.target)
    #print(iris.data)
    print(search)
    print(clf)

self.textEdit_44.setText(str(search.best_params_))
self.textEdit_19.setText(str(iris.target))
self.textEdit_20.setText(str(iris.data))
```

GRID SEARCH

```
from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV
iris = datasets.load_iris()
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
svc = svm.SVC()
clf = GridSearchCV(svc, parameters)
clf.fit(iris.data, iris.target)
sorted(clf.cv_results_.keys())
#print(iris.data)
clf.best_params_
print("sevilay",clf.best_params_)
self.textEdit_45.setText(str(clf.best_params_))
self.textEdit_21.setText(str(iris.target))
self.textEdit_22.setText(str(iris.data))
```



Burada random ve grid search için best parameters lerini ve target ile data ifadelerini gösterdim.

Grid search; Modelde denenmesi istenen hiperparametreler ve değerleri için bütün kombinasyonlar ile ayrı ayrı model kurulur ve belirtilen metriğe göre en başarılı hiperparametre seti belirlenir.(Gridsearch butonuna tıklayarak gösterdim)

Random search; Rastgele olarak bir hiperparametre seti seçilir ve cross-validation ile model kurularak test edilir.(randomsearch butonuna tıklayarak gösterdim)

TOPLU ÖĞRENME

KARAR AĞACINA AİT ACCURACY VE KARISIKLIK MATRİSİ

```
def yukle11(self):
    iris_dataset=pd.read_csv('iris.data')
    iris_dataset

    x=iris_dataset.iloc[:, [2,3]].values
    y=iris_dataset.iloc[:, 4].values

    from sklearn.model_selection import train_test_split
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,
    random_state=0)

    #karar ağacı
```

```

from sklearn.tree import DecisionTreeClassifier
karar_agaci=DecisionTreeClassifier()
karar_agaci.fit(x_train,y_train)
y_pred = karar_agaci.predict(x_test)
from sklearn.metrics import confusion_matrix,accuracy_score
print("KARAR AĞACINA AİT SINIFLANDIRMA")
print(confusion_matrix(y_test,y_pred))
print("doğruluk oranı:",accuracy_score(y_test,y_pred))
self.label_12.setText(str(accuracy_score(y_test,y_pred)))
self.textEdit_11.setText(str(confusion_matrix(y_test,y_pred)))

```

RANDOM FORESTE AİT KARIŞIKLIK MATRİSİ VE ACCURACY

```

def yukle12(self):
    iris_dataset=pd.read_csv('iris.data')
    iris_dataset

    x=iris_dataset.iloc[:, [2,3]].values
    y=iris_dataset.iloc[:, 4].values

    from sklearn.model_selection import train_test_split
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,
    random_state=0)

    #random forest
    from sklearn.ensemble import RandomForestClassifier
    randomForest=RandomForestClassifier(n_estimators=15,random_state=0)
    randomForest.fit(x_train,y_train)
    y_pred = randomForest.predict(x_test)
    print("RANDOM FOREST")
    from sklearn.metrics import confusion_matrix,accuracy_score
    self.label_11.setText(str(accuracy_score(y_test,y_pred)))
    self.textEdit_12.setText(str(confusion_matrix(y_test,y_pred)))

```

BAGGINGE AİT KARIŞIKLIK MATRİSİ VE ACCURACY

```

def yukle13(self):
    iris_dataset=pd.read_csv('iris.data')
    iris_dataset

    x=iris_dataset.iloc[:, [2,3]].values
    y=iris_dataset.iloc[:, 4].values

    from sklearn.model_selection import train_test_split
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,
    random_state=0)
    from sklearn.ensemble import BaggingClassifier

    bagging=BaggingClassifier(DecisionTreeClassifier(),max_samples=0.5,max_features=1.
    0,n_estimators=20)
    bagging.fit(x_train,y_train)
    y_pred = bagging.predict(x_test)
    print("BAGGING")
    from sklearn.metrics import accuracy_score,confusion_matrix
    self.label_10.setText(str(accuracy_score(y_test,y_pred)))

```

```
self.textEdit_13.setText(str(confusion_matrix(y_test,y_pred)))
```

VOOTING E AİT KARIŞIKLIK MATRİSİ VE ACCURACY

```
#VOOTING
def yukle14(self):
    iris_dataset=pd.read_csv('iris.data')
    iris_dataset

    X=iris_dataset.iloc[:, [2,3]].values
    y=iris_dataset.iloc[:, 4].values

    from sklearn.model_selection import train_test_split
    X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,
    random_state=0)

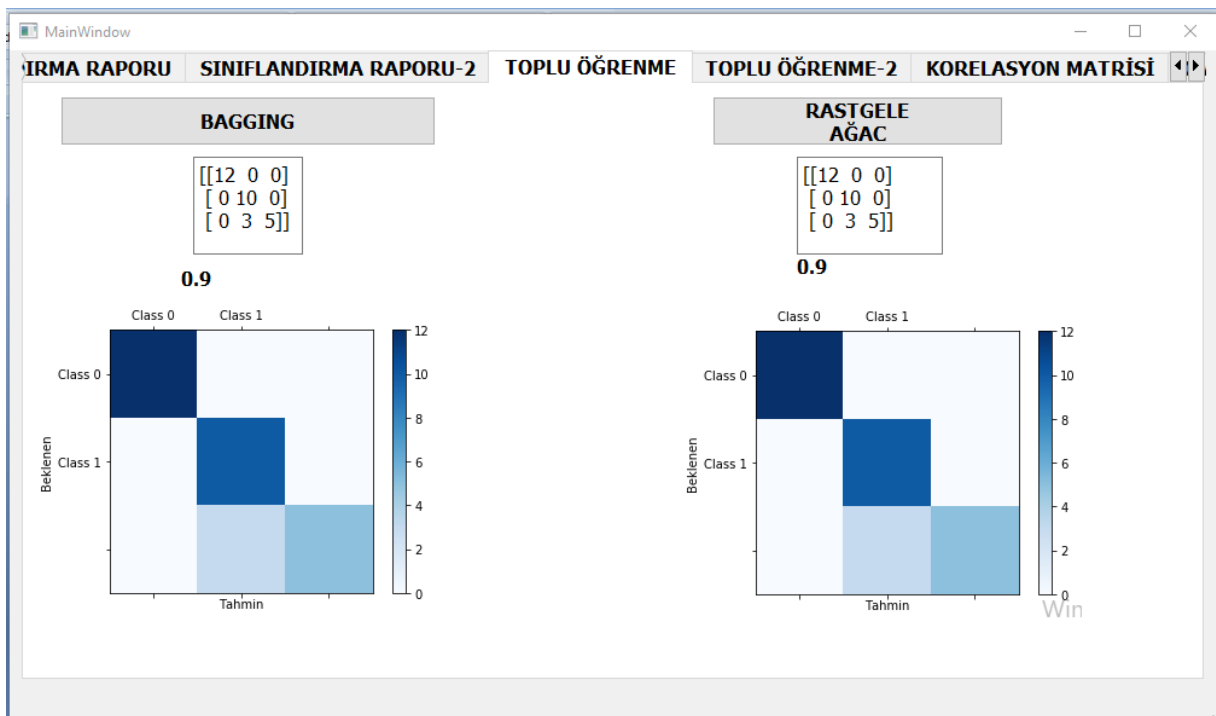
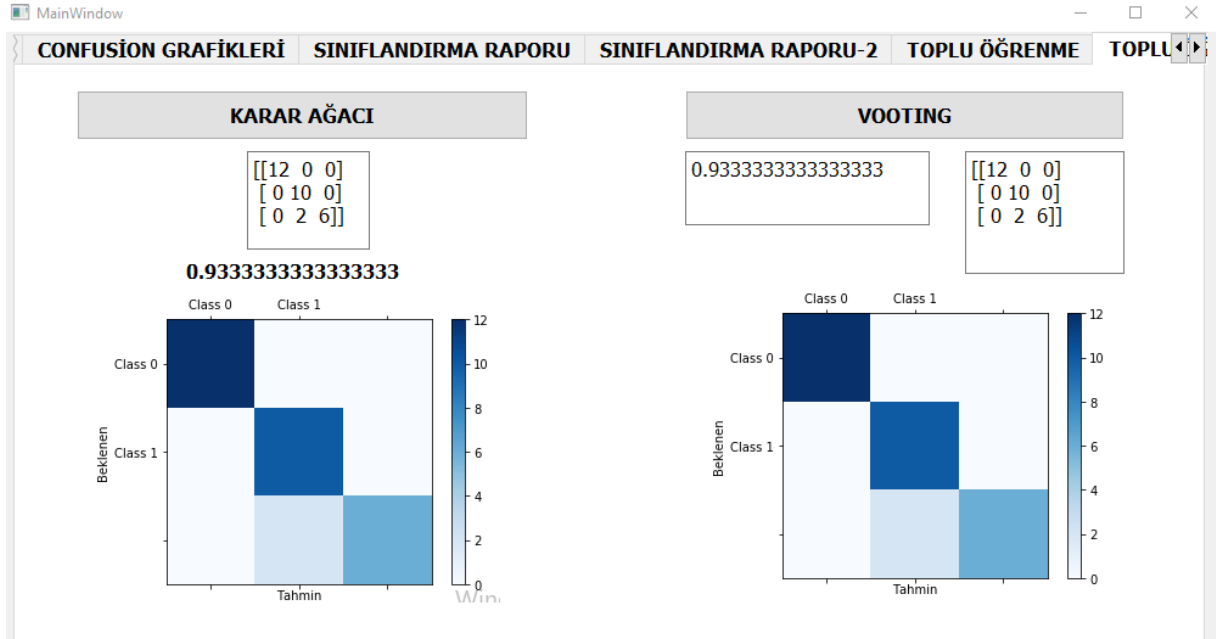
    LogReg = LogisticRegression()
    KNN=KNeighborsClassifier(n_neighbors = 4)
    dt = DecisionTreeClassifier(criterion='gini', max_depth= 6, min_samples_leaf = 1)

    classifiers = [ ('Logistic Regression', LogReg),
    ('KNeighbors', KNN),
    ('Decision Tree', dt)
    ]

    for name, model in classifiers:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        print(name, acc)

    vc = VotingClassifier(estimators = classifiers, voting = 'hard')
    vc.fit(X_train, y_train)
    y_pred = vc.predict(X_test)
    vc_acc = accuracy_score(y_test, y_pred)

    self.textEdit_14.setText(str(accuracy_score(y_test, y_pred)))
    #self.textEdit_23.setText(str(vc.predict(X_test)))
    self.textEdit_23.setText(str(confusion_matrix(y_test,y_pred)))
```



Bagging;torbalama tamamen random veriler orta çıkması(bagging butonuna tıklayarak gösterdim)

Random forest; Random forest regresyon birden fazla karar ağacını kullanarak daha uyumlu modeller üreterek isabetli tahminlerde bulunmaya yarayan bir regresyon modelidir.(random forest butonuna tıklayarak gösterdim)

Karar ağacı; Bir karar ağacı, çok sayıda kayıt içeren bir veri kümesini, bir dizi karar kuralları uygulayarak daha küçük kümelere bölmek için kullanılan bir yapıdır hangi sınıfa ait olduğu belli olmayan verilerin sınıflarının belirlenmesini sağlar.(karar ağacı butonuna tıklayarak gösterdim)

Vooting; en son kararı veren yapıdır.Vooting butununa tıklayarak Accuracy değerini ve confusion matrisinin gösterdim

LR İÇİN ROC EĞRİSİ

```
def yukle23(self):
    # 2 sınıflı veri seti oluşturdum
    x, y = make_classification(n_samples=1000, n_classes=2, random_state=6)

    # Train/test
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.5,
random_state=2)

    probs = [0 for _ in range(len(y_test))]# rastgele tahmin

    # Regression model
    model = LogisticRegression(solver='lbfgs')
    model.fit(x_train, y_train)

    lr_probs = model.predict_proba(x_test)# Olasılık tahmini

    lr_probs = lr_probs[:, 1]# Olasılıkların sadece pozitif olması için

    ns_auc = roc_auc_score(y_test, probs)
    lr_auc = roc_auc_score(y_test, lr_probs)

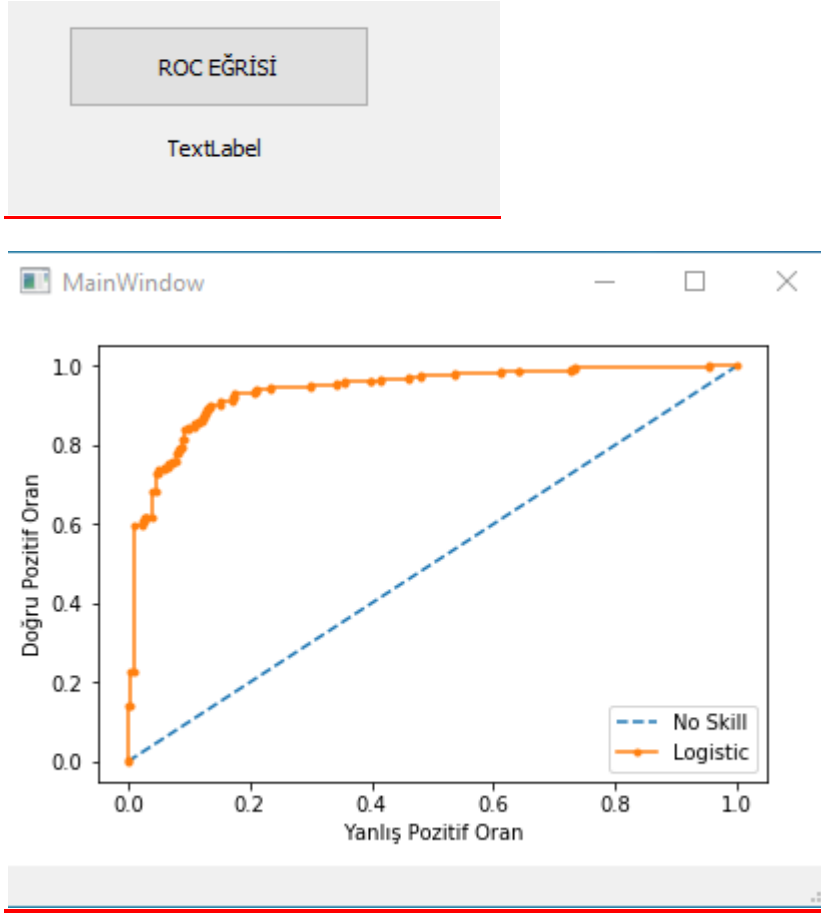
    print('No Skill: ROC AUC=%.3f' % (ns_auc))# Özet
    print('Logistic: ROC AUC=%.3f' % (lr_auc))
    self.label_10.setText(str('No Skill: ROC AUC=%.3f' % (ns_auc)))
    self.label_11.setText(str('LOjistik: ROC AUC=%.3f' % (lr_auc)))

    # ROC hesabı
    ns_fpr, ns_tpr, _ = roc_curve(y_test, probs)
    lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)

    # ROC EĞRİSİ ÇİZİMİ
    pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
    pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
    pyplot.xlabel('Yanlış Pozitif Oran')
    pyplot.ylabel('Doğru Pozitif Oran')
    pyplot.legend()
    pyplot.show()

    label_13 = QLabel(self)
    pixmap = QPixmap('roc.png')
    label_13.setPixmap(pixmap)
    self.setCentralWidget(label_13)
```

```
self.resize(pixmap.width(), pixmap.height())
```



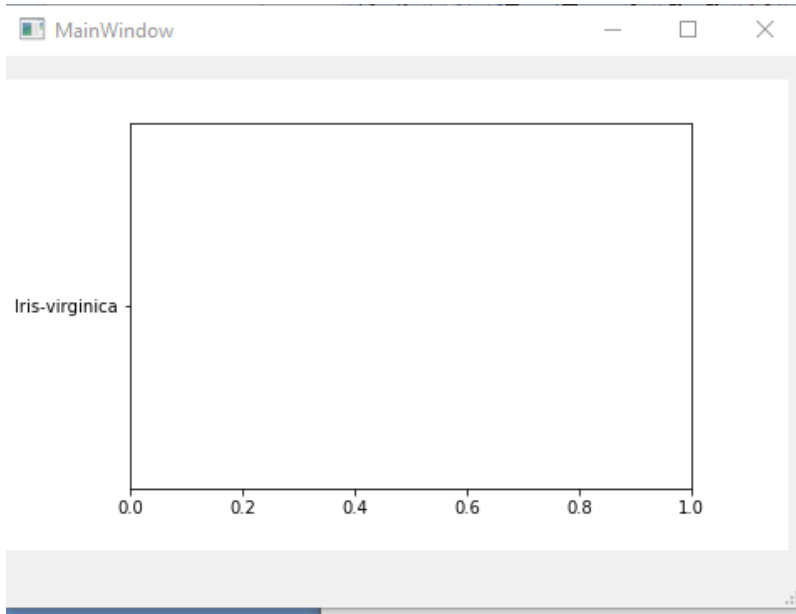
Roc eğrisi;tüm sınıflandırma eşiklerinden bir sınıflandırma modelinin performansını gösteren bir grafikdir. Y ekseninde gerçek pozitif oranı ve X ekseninde yanlış pozitif oranını gösterir.

ROC eğrisi buttouna basınca txtlabelde grafiği çizdirdim.

VERİSETLERİNİ TRAIN VE TEST OLARAK AYIRMA GRAFIĞI

```
def yukle20(self):
    label_12 = QLabel(self)
    pixmap = QPixmap('ayirma1.png')
    label_12.setPixmap(pixmap)
    self.setCentralWidget(label_12)
    self.resize(pixmap.width(), pixmap.height())
```

VERİSETNİ AYIRMA BUTTONUNA TIKLAYINCA AYRI BİR MAIN WINDOW SAYFASI AÇIYOR.



Veri setimde string ifade olduğundan tablom böyle gözüktü.

HİSTOGRAM VE DAĞILIM GRAFİĞİ:

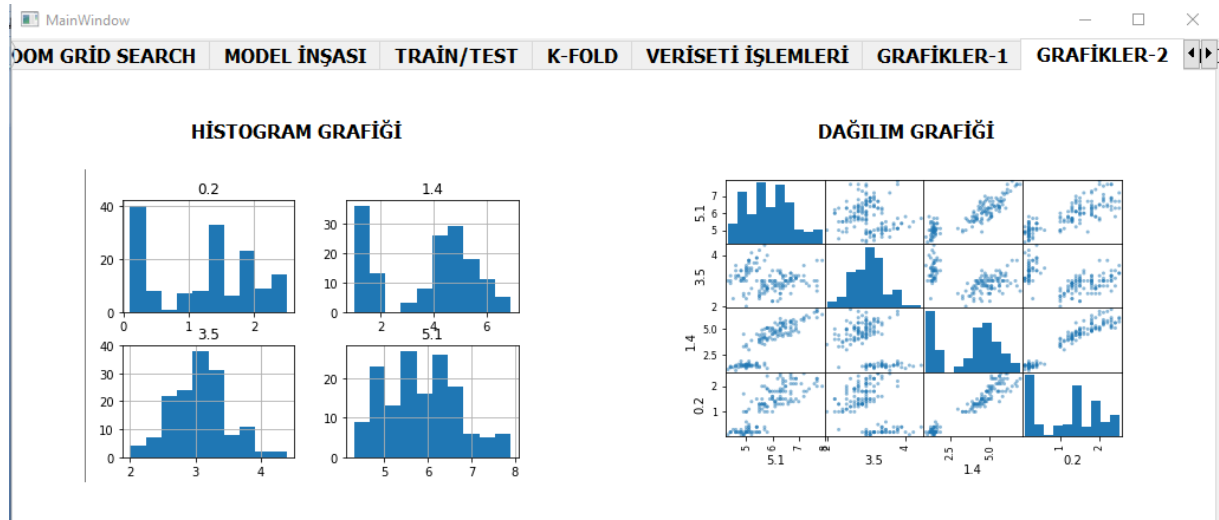
```
#GRAFİKLER
iris_dataset = pd.read_csv("iris.data")

# Bağımlı ve bağımsız değişkenlerin oluşturulması
X = iris_dataset.values[:, 0:4]
Y = iris_dataset.values[:, 4]

# Veri kümesinin eğitim ve test verileri olarak ayrılması
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

print("HİSTOGRAM")
# histogram
iris_dataset.hist()
plt.show()

print("DAĞILIM GRAFİĞİ MATRİX")
# scatter plot matrix
scatter_matrix(iris_dataset)
plt.show()
```



PCA

```
#PCA
def yukle26(self):
    iris_dataset = pd.read_csv("iris.data")

    # Bağımlı ve bağımsız değişkenlerin oluşturulması
    X = iris_dataset.values[:, 0:4]
    Y = iris_dataset.values[:, 4]

    # Veri kümesinin eğitim ve test verileri olarak ayrılması
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=0.20, random_state=7)

    from sklearn.preprocessing import StandardScaler
    X = StandardScaler().fit_transform(X)
    X = pd.DataFrame(X)

    from sklearn.decomposition import PCA
    pca = PCA()
    X_pca = pca.fit_transform(X)
    X_pca = pd.DataFrame(X_pca)
    X_pca.head()

    explained_variance = pca.explained_variance_ratio_
    explained_variance

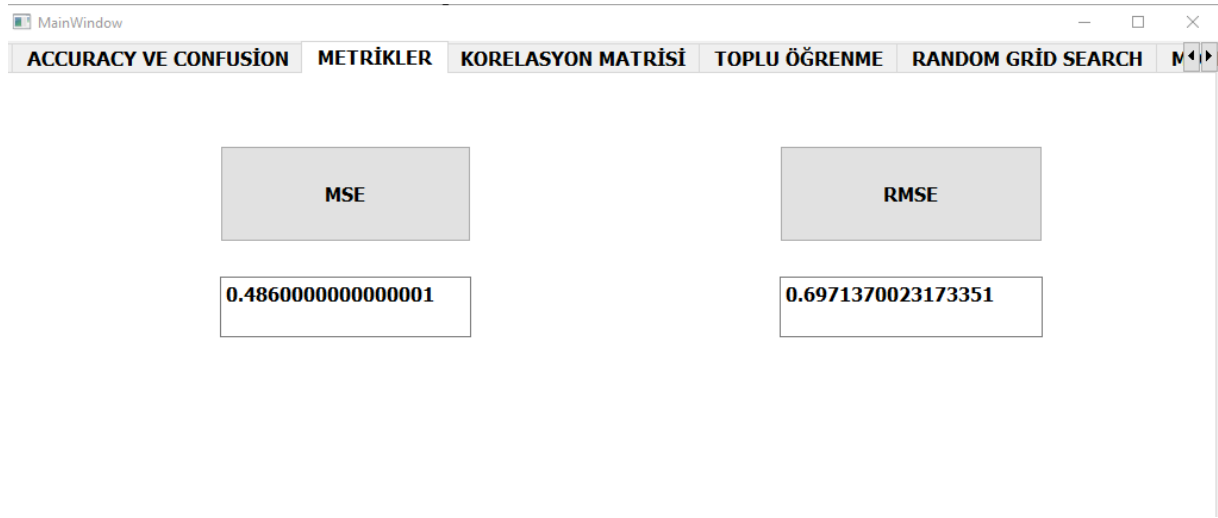
    X_pca['target']=Y
    X_pca.columns = ['PC1','PC2','PC3','PC4','target']
    X_pca.head()
    print(X_pca.head())
    self.textEdit_26.setText(str(X_pca))
```

PCA					
	PC1	PC2	PC3	PC4	target
0	-2.107950	-0.644276	-0.228768	-0.103873	Iris-setosa
1	-2.387971	-0.305833	0.049672	-0.028274	Iris-setosa
2	-2.324879	-0.562923	0.097142	0.066136	Iris-setosa
3	-2.405086	0.687591	0.018819	0.037002	Iris-setosa
4	-2.083204	1.530252	0.027757	-0.004931	Iris-setosa
5	-2.463685	0.087954	0.340195	0.037864	Iris-setosa
6	-2.251750	0.259644	-0.084873	0.025101	Iris-setosa
7	-2.364581	-1.082557	0.152275	0.026586	Iris-setosa

METRİKLER(MSE VE RMSE):

```
def yukle27(self):
    y_actual = [1,2,3,4,5]
    y_predicted = [1.6,2.5,2.9,3,4.1]
    MSE = np.square(np.subtract(y_actual,y_predicted)).mean()
    RMSE = math.sqrt(MSE)
    self.textEdit_27.setText(str(MSE))
    #print("RMS:",RMSE)
    print("MSE:",MSE)
```

```
def yukle28(self):
    y_actual = [1,2,3,4,5]
    y_predicted = [1.6,2.5,2.9,3,4.1]
    MSE = np.square(np.subtract(y_actual,y_predicted)).mean()
    RMSE = math.sqrt(MSE)
    self.textEdit_28.setText(str(RMSE))
    print("RMS:",RMSE)
    #print("MSE:",MSE)
```



OVERSAMPLING UNDERSAMPLING

OVERSAMPLING GÖSTERDİM

```
def yukle29(self):
    #OVERSAMPLİNG
    X, y = make_classification(n_classes=2, class_sep=2,
        weights=[0.1, 0.9], n_informative=3, n_redundant=1, flip_y=0,
        n_features=20, n_clusters_per_class=1, n_samples=1000, random_state=10)
    print('Orjinal veri kümesi: %s' % Counter(y))
    self.textEdit_29.setText(str('Orjinal veri kümesi: %s' % Counter(y)))

    ros = RandomOverSampler(random_state=42)
    X_res, y_res = ros.fit_resample(X, y)
    print('Yeniden örneklenmiş veri kümesi %s' % Counter(y_res))
    self.textEdit_35.setText(str('Yeniden örneklenmiş veri kümesi %s' %
        Counter(y_res)))
```

UNDERSAMPLING GÖSTERDİM

```
def yukle30(self):
    #UNDERSAMPLİNG
    X, y = make_classification(n_classes=2, class_sep=2, weights=[0.1, 0.9],
        n_informative=3, n_redundant=1, flip_y=0, n_features=20, n_clusters_per_class=1,
        n_samples=1000, random_state=10)
    print('orjinal veri kümesi: %s' % Counter(y))
    self.textEdit_36.setText(str('orjinal veri kümesi: %s' % Counter(y)))

    rus = RandomUnderSampler(random_state=42)
    X_res, y_res = rus.fit_resample(X, y)
    print('Yeniden örneklenmiş veri kümesi: %s \n' % Counter(y_res))
```

```
self.textEdit_37.setText(str('Yeniden örneklenmiş veri kümesi: %s \n' %  
Counter(y_res)))
```

