

**TC  
KASTAMONU ÜNİVERSİTESİ  
MÜHENDİSLİK VE MİMARLIK FAKÜLTESİ  
BİLGİSAYAR MÜHENDİSLİĞİ PROGRAMI**

**BLM409  
MAKİNE ÖĞRENMESİ**

**Metin ORAL  
134410013**

Kastamonu, 2018

## 1. RUS(Random Undersampling) - ROS (Random Oversampling)

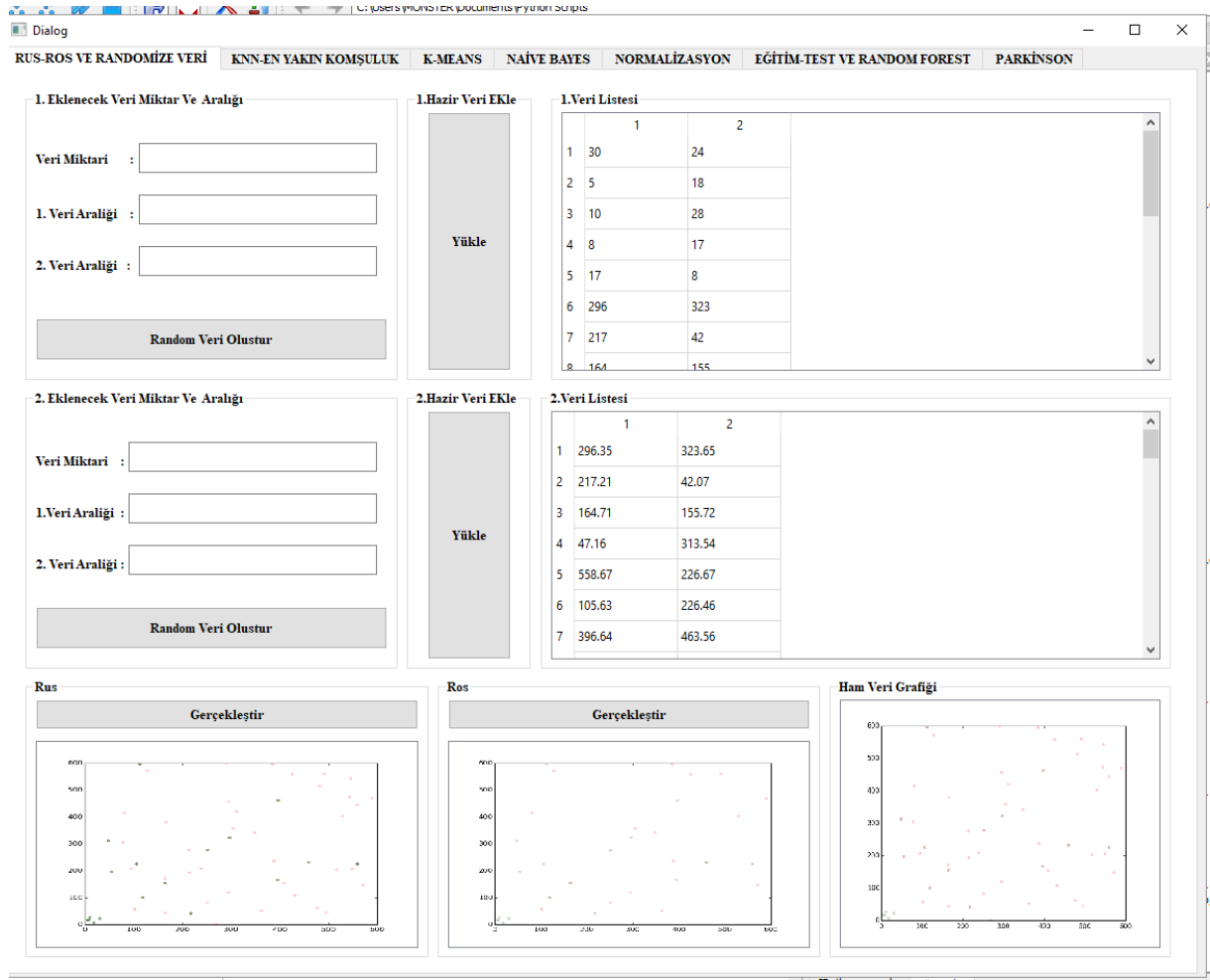
Line editlerden alınan veri aralık ve miktarı ile random olarak veri oluşturma.

Rus Veya Ros işlemlerinin uygulamak için daha önceden hazırlanmış anlamlı veri setlerini yükleme .

Rus: Veri Setini dengelemek İçin random Olarak veri miktarını azaltma

Ros: Veri Setini dengelemek İçin random Olarak veri miktarını çoğaltma

Her iki durumda da amaç veriyi dengeleyerek daha sağlıklı sonuç alma



	1	2
1	30	24
2	5	18
3	10	28
4	8	17
5	17	8
6	296	323
7	217	42
8	164	155

	1	2
1	296.35	323.65
2	217.21	42.07
3	164.71	155.72
4	47.16	313.54
5	558.67	226.67
6	105.63	226.46
7	396.64	463.56

```
self.btnRusRosVeriYukle1.clicked.connect(self.RusRosVeriYukle1)
self.btnRusRosVeriYukle2.clicked.connect(self.RusRosVeriYukle2)
self.btnRus.clicked.connect(self.Rus)
self.btnRos.clicked.connect(self.Ros)
```

```
Y=[]
X=[]
def RusRosRandomVeri1(self):
self.X=[]
for i in range(int(self.leRusRosVeriMiktari1.text())):
```

```

x="{:.2f}".format(random.uniform(float(self.leRusRosVeriAralk1.text()), float(self.leRusRosVeriAralk11.text())))
y="{:.2f}".format(random.uniform(float(self.leRusRosVeriAralk1.text()), float(self.leRusRosVeriAralk11.text())))
self.X.append([x,y])
self.tw1WeriDoldur()
def RusRosRandomVeri2(self):
self.Y=[]
for i in range(int(self.leRusRosVeriMiktari2.text())):
x="{:.2f}".format(random.uniform(float(self.leRusRosVeriAralk2.text()), float(self.leRusRosVeriAralk22.text())))
y="{:.2f}".format(random.uniform(float(self.leRusRosVeriAralk2.text()), float(self.leRusRosVeriAralk22.text())))
self.Y.append([x,y])
self.tw2WeriDoldur()
def tw1WeriDoldur(self):
if(len(self.Y)>0):
for i in range(len(self.X)):
plt.plot(self.X[i][0], self.X[i][1], "g",markersize = 5,marker = "o",alpha=0.2)
for i in range(len(self.Y)):
plt.plot(self.Y[i][0], self.Y[i][1], "r",markersize = 9,marker = ".",alpha=0.2)
else:
for i in range(len(self.X)):
plt.plot(self.X[i][0], self.X[i][1], "g",markersize = 5,marker = "o",alpha=0.2)
plt.savefig('./sonuclar/tumveriler.png')
plt.show()
w,h=self.gvRusRos.width()-5,self.gvRusRos.height()-5
self.gvRusRos.setScene(self.show_image('./sonuclar/tumveriler.png',w,h))
self.twRusRosVeri1.setColumnCount(2)
self.twRusRosVeri1.setRowCount(len(self.X)) ##set number of rows
for rowNumber,row in enumerate(self.X):
self.twRusRosVeri1.setItem(rowNumber, 0, QtGui.QTableWidgetItem(str(row[0])))
self.twRusRosVeri1.setItem(rowNumber, 1, QtGui.QTableWidgetItem(str(row[1])))
def tw2WeriDoldur(self):
if(len(self.X)>0):
for i in range(len(self.X)):
plt.plot(self.X[i][0], self.X[i][1], "g",markersize = 5,marker = "o",alpha=0.2)
for i in range(len(self.Y)):
plt.plot(self.Y[i][0], self.Y[i][1], "r",markersize = 9,marker = ".",alpha=0.2)
else:
for i in range(len(self.Y)):
plt.plot(self.Y[i][0], self.Y[i][1], "g",markersize = 9,marker = ".",alpha=0.2)
plt.savefig('./sonuclar/tumveriler.png')
# plt.show()
w,h=self.gvRusRos.width()-5,self.gvRusRos.height()-5
self.gvRusRos.setScene(self.show_image('./sonuclar/tumveriler.png',w,h))
self.twRusRosVeri2.setColumnCount(2)
self.twRusRosVeri2.setRowCount(len(self.Y)) ##set number of rows
for rowNumber,row in enumerate(self.Y):
self.twRusRosVeri2.setItem(rowNumber, 0, QtGui.QTableWidgetItem(str(row[0])))
self.twRusRosVeri2.setItem(rowNumber, 1, QtGui.QTableWidgetItem(str(row[1])))
def Rus(self):
adet=0
if(len(self.X)>len(self.Y)):
adet=len(self.X)
else:
adet=len(self.Y)
for i in range(adet):
plt.plot(self.Y[i%len(self.Y)][0], self.Y[i%len(self.Y)][1], "r", markersize = 9,marker = ".",alpha=0.2)
plt.plot(self.X[i%len(self.X)][0], self.X[i%len(self.X)][1], "g", markersize = 5,marker = "o",alpha=0.2)
plt.savefig('./sonuclar/Rus.png')
#plt.show()
w,h=self.gvRus.width()-5,self.gvRus.height()-5
self.gvRus.setScene(self.show_image('./sonuclar/Rus.png',w,h))
def Ros(self):
ros=[]
if(len(self.X)>len(self.Y)):

```

```

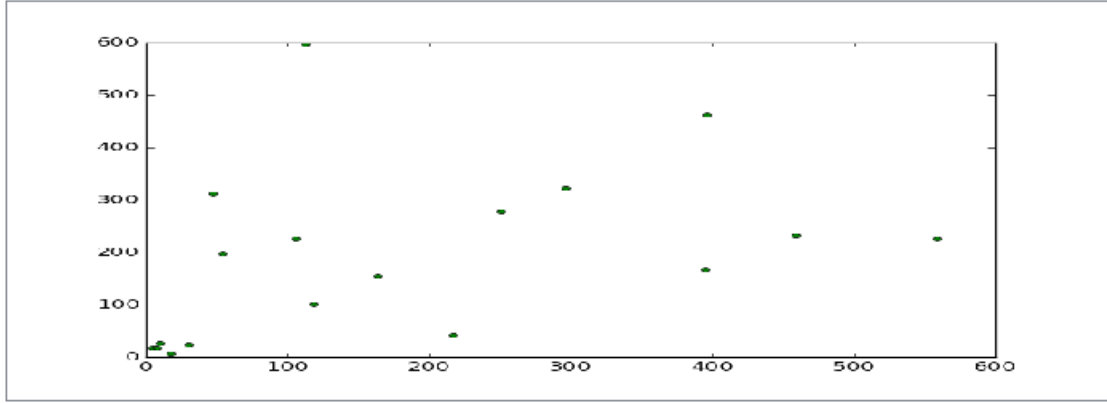
ros=random.sample(self.X, len(self.Y))
for i in range(len(ros)):
plt.plot(self.Y[i][0], self.Y[i][1], "r", markersize = 9,marker = ".",alpha=0.2)
plt.plot(ros[i][0], ros[i][1], "g", markersize = 5,marker = "o",alpha=0.2)
else:
ros=random.sample(self.Y, len(self.X))
for i in range(len(ros)):
plt.plot(ros[i][0], ros[i][1], "r", markersize = 9,marker = ".",alpha=0.2)
plt.plot(self.X[i][0], self.X[i][1], "g", markersize = 5,marker = "o",alpha=0.2)
plt.savefig('./sonuclar/Ros.png')
#plt.show()
w,h=self.gvRos.width()-5,self.gvRos.height()-5
self.gvRos.setScene(self.show_image('./sonuclar/Ros.png',w,h))
def show_image(self, img_name,width,height):
pixMap = QtGui.QPixmap(img_name)
pixMap=pixMap.scaled(width,height)
pixItem = QtGui.QGraphicsPixmapItem(pixMap)
scene2 = QGraphicsScene()
scene2.addItem(pixItem)
return scene2
def RusRosVeriYukle1(self):
self.veriyolu1 = unicode(QtGui.QFileDialog.getOpenFileName(self, "Duzenlenecek dosyayi secin", ".", "Resim dosyolari (*.*)"))
f = open(self.veriyolu1)
self.X=[]
for i,row in enumerate(f.readlines()):
currentline = row.split(",")
temp=[]
for column_value in currentline:
temp.append(column_value)
self.X.append(temp)
self.tw1WeriDoldur()
def RusRosVeriYukle2(self):
self.veriyolu2 = unicode(QtGui.QFileDialog.getOpenFileName(self,"Duzenlenecek dosyayi secin", ".", "Resim dosyolari (*.*)"))
f = open(self.veriyolu2)
self.Y=[]
for i,row in enumerate(f.readlines()):
currentline = row.split(",")
temp=[]
for column_value in currentline:
temp.append(column_value)
self.Y.append(temp)
self.tw2WeriDoldur()

```

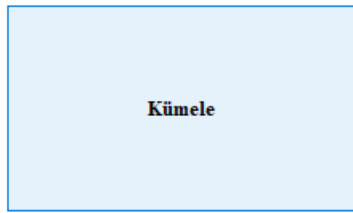
## 2.Knn(En Yakın Komşuluk Algoritması)

Sınıflandırmada (classification) kullanılan bu algoritmaya göre sınıflandırma sırasında çıkarılan özelliklerden (feature extraction), sınıflandırılmak istenen yeni bireyin daha önceki bireylerden k tanesine yakınlığına bakılmasıdır.

Ham Verinin Grafiği



Kümele İşlemini Yap



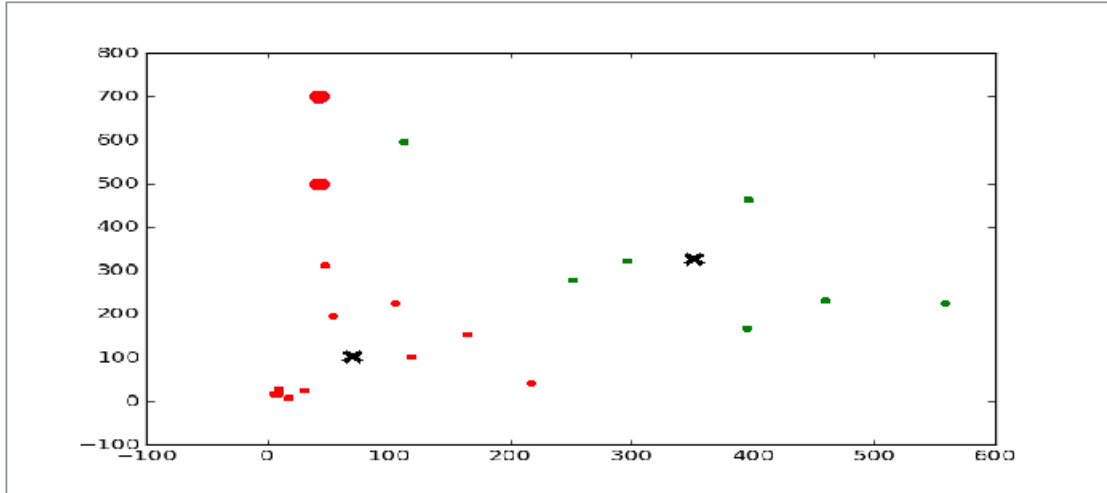
Veri Ekleme Ekranı

1. Veri :   
2. Veri :

Yeni Veri Ekle



GroupBox



```

self.btnKnnVeriYukle.clicked.connect(self.KnnVeriYukle)
self.btnKnnKumele.clicked.connect(self.KnnKumele)
self.btnKnnYeniVeriEkleme.clicked.connect(self.KnnYeniVeriEkleme)

Knn = []
def KnnVeriYukle(self):
    self.fileNameKNN = unicode(
    QtGui.QFileDialog.getOpenFileName(self,"Duzenlenecek dosyayi secin", ".", "Resim dosyaları (*.*)"))
    f = open(self.fileNameKNN)
    self.Knn = []
    for i, row in enumerate(f.readlines()):
        currentline = row.split(",")
        temp = []
        for column_value in currentline:
            temp.append(column_value)
        self.Knn.append(temp)
    self.twKnnVeriDoldur()

def twKnnVeriDoldur(self):
    if (len(self.Knn) > 0):
        for i in range(len(self.Knn)):
            plt.plot(self.Knn[i][0], self.Knn[i][1], "g", markersize=5, marker="o")
            plt.savefig('./sonuclar/knn/tumveriler.png')
            plt.show()
        w, h = self.gvKnnVeriGrafigi.width() - 5, self.gvKnnVeriGrafigi.height() - 5
        self.gvKnnVeriGrafigi.setScene(self.show_image('./sonuclar/knn/tumveriler.png', w, h))

self.twKnnVeri.setColumnCount(2)
self.twKnnVeri.setRowCount(len(self.Knn)) ##set number of rows
for rowNumber, row in enumerate(self.Knn):
    self.twKnnVeri.setItem(rowNumber, 0, QtGui.QTableWidgetItem(str(row[0])))
    self.twKnnVeri.setItem(rowNumber, 1, QtGui.QTableWidgetItem(str(row[1])))

def KnnKumele(self):
    X = self.Knn
    kumeyeri = []
    kumesayisi = 2
    for i in range(len(X)):
        kumeyeri.append(int(X[i][0]) % kumesayisi)
        devammi = True
        while (devammi):
            merkezler = []
            for i in range(kumesayisi):
                merkezler.append([0, 0, 0])
            for i in range(len(X)):
                merkezler[kumeyeri[i]] = [float(merkezler[kumeyeri[i]][0]) + float(X[i][0]),
                float(merkezler[kumeyeri[i]][1]) + float(X[i][1]),
                int(merkezler[kumeyeri[i]][2]) + 1]
            for i in range(len(merkezler)):
                merkezler[i] = [float(merkezler[i][0]) / float(merkezler[i][2]), (merkezler[i][1] / merkezler[i][2]),
                int(merkezler[i][2])]
            kumeyeriyeni = []
            for i in range(len(X)):
                kumeyeriyeni.append(0)
            for i in range(len(X)):
                deger = [0, 0]
                deger = [math.sqrt(math.pow(abs(float(X[i][0]) - float(merkezler[0][0])), 2) + math.pow(
                abs(float(X[i][1]) - float(merkezler[0][1])), 2)), 0]
                if (deger[0] > math.sqrt(math.pow(abs(float(X[i][0]) - float(merkezler[1][0])), 2) + math.pow(
                abs(float(X[i][1]) - float(merkezler[1][1])), 2))):
                    deger = [math.sqrt(math.pow(abs(float(X[i][0]) - float(merkezler[1][0])), 2) + math.pow(
                    abs(float(X[i][1]) - float(merkezler[1][1])), 2)), 1]
                kumeyeriyeni[i] = deger[1]

```

```

if (kumeyeriye == kumeyeri):
devammi = False
else:
kumeyeri = kumeyeriye
colors = ["g.", "r.", "b.", "y.", "c.", "m."]
for i in range(len(X)):
plt.plot(X[i][0], X[i][1], colors[kumeyeri[i]], markersize=10)
for i in range(len(merkezler)):
plt.scatter(int(merkezler[i][0]), int(merkezler[i][1]), marker="x", s=70, linewidths=3, zorder=10,
c="black")
plt.savefig('./sonuclar/knn/Knn.png')
plt.show()
w, h = self.gvKnnSonucGrafigi.width() - 5, self.gvKnnSonucGrafigi.height() - 5
self.gvKnnSonucGrafigi.setScene(self.show_image('./sonuclar/knn/Knn.png', w, h))
self.kumeye = kumeyeri

```

#### **def KnnYeniVeriEkleme(self):**

```

ekle = [int(self.leKnnVeri1.text()), int(self.leKnnVeri2.text())]
X = self.Knn
merkezler = []
kumeyeri = self.kumeye
colors = ["g.", "r.", "b.", "y.", "c.", "m."]
plt.plot(ekle[0], ekle[1], colors[5], markersize=20)
for i in range(len(X)):
plt.plot(X[i][0], X[i][1], colors[kumeyeri[i]], markersize=10)
for i in range(len(merkezler)):
plt.scatter(int(merkezler[i][0]), int(merkezler[i][1]), marker="x", s=70, linewidths=3, zorder=10,
c="black")
plt.show()
degerler = []
for i in range(len(X)):
degerler.append(
math.sqrt(math.pow(abs(float(X[i][0]) - ekle[0]), 2) + math.pow(abs(float(X[i][1]) - ekle[1]), 2)))
bak = [[max(degerler), 0], [max(degerler), 0], [max(degerler), 0], [max(degerler), 0], [max(degerler), 0]]
for i in range(len(degerler)):
for j in range(len(bak)):
if (degerler[i] < bak[j][0]):
if (degerler[i] != bak[j][0]):
bak[j] = [degerler[i], i]
break
sifir = 0
bir = 0
for a in range(len(bak)):
if (kumeyeri[bak[a][1]] == 1):
bir = bir + 1
else:
sifir = sifir + 1
renk = 0
if (bir > sifir):
renk = 1
plt.plot(ekle[0], ekle[1], colors[renk], markersize=20)
for i in range(len(X)):
plt.plot(X[i][0], X[i][1], colors[kumeyeri[i]], markersize=10)
for i in range(len(merkezler)):
plt.scatter(int(merkezler[i][0]), int(merkezler[i][1]), marker="x", s=70, linewidths=3, zorder=10,
c="black")
plt.savefig('./sonuclar/knn/Knn-1.png')
plt.show()
w, h = self.gvKnnSonucGrafigi.width() - 5, self.gvKnnSonucGrafigi.height() - 5
self.gvKnnSonucGrafigi.setScene(self.show_image('./sonuclar/knn/Knn-1.png', w, h))

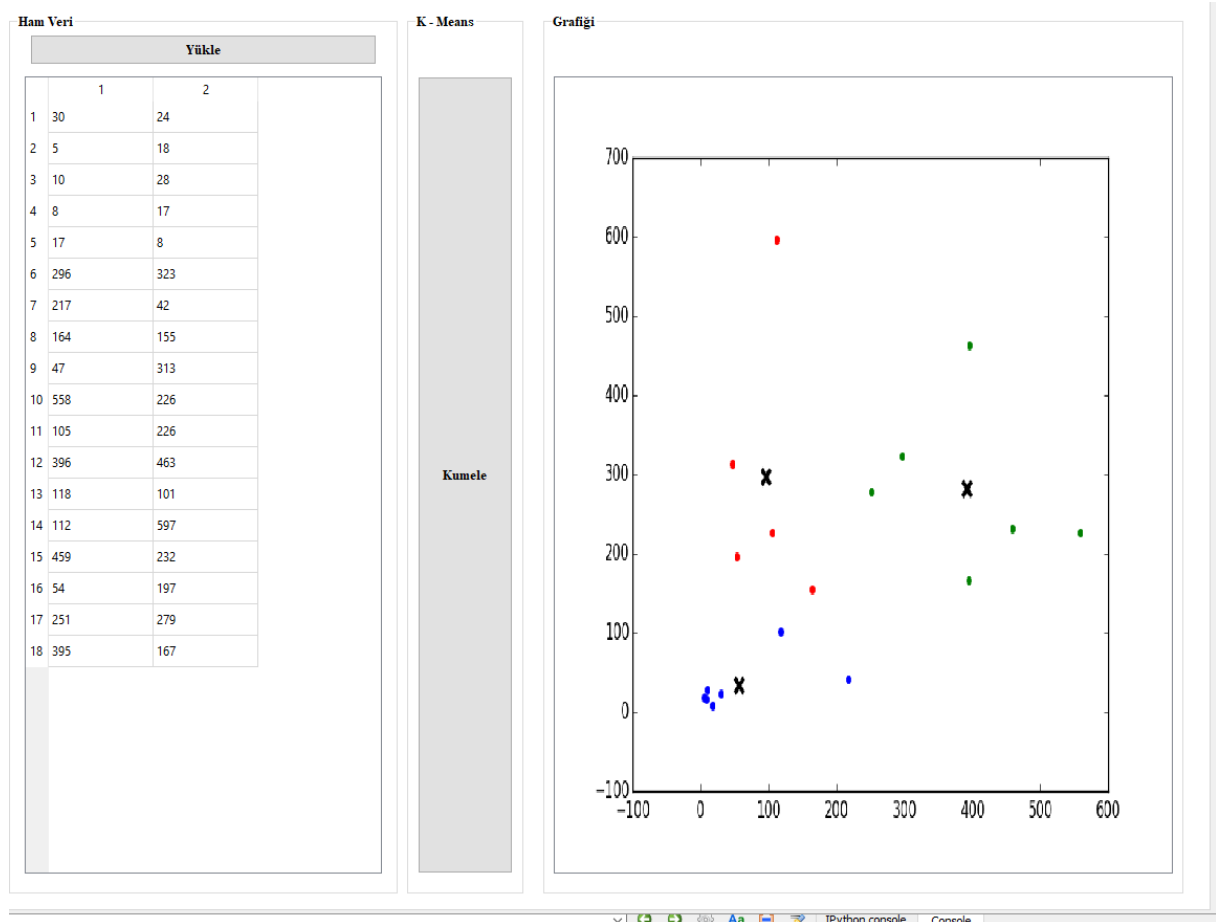
```

### 3.K-Means(K-Ortalama Algoritması)

Kümeleme (clustering) kullanılan algoritmalarından birisidir. Amaç özellik çıkarımı (Feature extraction) yapılmış bir grup verinin birden fazla küme özelliğine göre hangi kümeye ait olduğunu bulunmasıdır. Kullanılan matematiksel yöntem her sınıf için merkez belirlenen noktaya uzaklığa (aynı zamanda bu hata miktarıdır) göre yeni kümelerin yerleştirilmesidir.

Algoritma temel olarak 4 aşamadan oluşur:

1. Küme merkezlerinin belirlenmesi
2. Merkez dışındaki örneklerin mesafelerine göre sınıflandırılması
3. Yapılan sınıflandırmaya göre yeni merkezlerin belirlenmesi (veya eski merkezlerin yeni merkeze kaydırılması)
4. Kararlı hale (stable state) gelene kadar 2. ve 3. adımların tekrarlanması





```
self.btnKmeansVeriYukle.clicked.connect(self.KmeansVeriYukle)  
self.btnKmeansVeriKumele.clicked.connect(self.KmeansVeriKumele)
```

```
colors = ["g.", "r.", "b.", "y.", " c."]  
kumesayisi=3  
kumeyeri=[]  
XKmeans=[]
```

```
def KmeansVeriYukle(self):  
f = open('./veriler/veri.txt')  
X=[]  
for i,row in enumerate(f.readlines()):  
    currentline = row.split(",")  
    temp=[]  
    for column_value in currentline:  
        temp.append(column_value)  
    X.append(temp)  
    for i in range(len(X)):  
        self.kumeyeri.append(int(X[i][0])%self.kumesayisi)  
    self.XKmeans=X  
    self.twKmeansVeriYukle.setColumnCount(2)  
    self.twKmeansVeriYukle.setRowCount(len(self.XKmeans)) ##set number of rows  
    for rowNumber, row in enumerate(self.XKmeans):  
        self.twKmeansVeriYukle.setItem(rowNumber, 0, QtGui.QTableWidgetItem(str(row[0])))  
        self.twKmeansVeriYukle.setItem(rowNumber, 1, QtGui.QTableWidgetItem(str(row[1])))
```

```
def KmeansVeriKumele(self):  
X=self.XKmeans  
devammi=True  
while(devammi):  
    merkezler=[]  
    for i in range(self.kumesayisi):  
        merkezler.append([0,0,0])  
    for i in range(len(X)):  
        merkezler[self.kumeyeri[i]]=[float(merkzler[self.kumeyeri[i]][0])+float(X[i][0]),float(merkzler[self.kumeyeri[i]][1])+float(X[i][1]),int(merkzler[self.kumeyeri[i]][2])+1]  
    for i in range(len(merkzler)):  
        merkezler[i]=[float(merkzler[i][0])/float(merkzler[i][2]),(merkezler[i][1]/merkezler[i][2]),int(merkzler[i][2])]  
    kumeyeriyeni=[]  
    print merkezler  
    for i in range(len(X)):  
        kumeyeriyeni.append(0)  
    for i in range(len(X)):  
        deger=[0,0]  
        deger=[ math.sqrt(math.pow(abs(float(X[i][0]) - float(merkzler[0][0])),2)+math.pow(abs(float(X[i][1]) - float(merkzler[0][1])),2)) , 0]  
        if(deger[0]> math.sqrt(math.pow(abs(float(X[i][0]) - float(merkzler[1][0])),2)+math.pow(abs(float(X[i][1]) - float(merkzler[1][1])),2))):  
            deger=[math.sqrt(math.pow(abs(float(X[i][0]) - float(merkzler[1][0])),2)+math.pow(abs(float(X[i][1]) - float(merkzler[1][1])),2)),1]  
        if(deger[0]> math.sqrt(math.pow(abs(float(X[i][0]) - float(merkzler[2][0])),2)+math.pow(abs(float(X[i][1]) - float(merkzler[2][1])),2))):  
            deger=[math.sqrt(math.pow(abs(float(X[i][0]) - float(merkzler[2][0])),2)+math.pow(abs(float(X[i][1]) - float(merkzler[2][1])),2)),2]  
        kumeyeriyeni[i]=deger[1]  
    if(kumeyeriyeni==self.kumeyeri):  
        devammi=False  
    else:  
        self.kumeyeri=kumeyeriyeni  
    for i in range(len(X)):  
        plt.plot(X[i][0], X[i][1], self.colors[self.kumeyeri[i]], markersize = 10)  
    for i in range(len(merkzler)):  
        plt.scatter(int(merkzler[i][0]),int(merkzler[i][1]), marker = "x", s=70, linewidths = 3, zorder = 10,c="Black")
```

```
plt.savefig('./sonuclar/kmeans/Kmeans.png')
plt.show()
w, h = self.gvKmeansVeriKumele.width() - 5, self.gvKmeansVeriKumele.height() - 5
self.gvKmeansVeriKumele.setScene(self.show_image('./sonuclar/kmeans/Kmeans.png', w, h))
```

#### 4.Naive Bayes

Herhangi bir sınıflandırma probleminde olduğu gibi, amacımız birden fazla özelliği taşıyan bir yöney (vektör) kullanarak verilen bilgilerden bir eğitim oluşturmak ve bu eğitim neticesinde gelen yeni verileri doğru bir şekilde sınıflandırmaktır.

**Tahmini Yapılacak Kelimeyi Gir**

para

**Tahmin Sonucu**

Ekonomi

**Naive Bayes**

**Tahmin Yap**

```
self.btnNiveBayesAra.clicked.connect(self.NiveBayesAra)
```

```
def NiveBayesAra(self):
    kelime=str(self.leNaiveBayes.text())
    def aranacak(kume, kelime, index):
        counter=0
        for i in range(len(kume)):
            if kume[i][index]==kelime:
                counter+=1
        return counter
    def aranacak_2(kume, kelime):
        counter=0
        for i in range(len(kume)):
            if kume[i]==kelime:
                counter+=1
        return counter
    def kelimeler(kume):
        kelimeler=[]
        silinecek="!@#$%&?,"
        for i in range(len(kume)):
            cumle=kume[i][0]
            for char in silinecek:
                cumle=cumle.replace(char,"")
            parca=cumle.split(' ')
            for c in parca:
                if aranacak_2(kelimeler, c)==0:
                    kelimeler.append(c)
        return kelimeler
    def arama(kume,kumeci,kelime):
        counter=0
```

```

for i in range(len(kume)):
if kume[i][1]==kumeci and kume[i][0].count(kelime)>0:
counter+=kume[i][0].count(kelime)
return counter
data=[["top, futbol, quaresma, atiba.", "Besiktas"],
["saha futbol fitness voleybol basketbol.", "Besiktas"],
["pepe, ofsayt,sut,tac, masa tenisi", "Besiktas"],
["ceza sahasi ,kale, top.", "Besiktas"],
["enflasyon, deflasyon, komisyon, sermaye, indeks", "ekonomi"],
["lira, kar, zarar, altin, faiz, hisse", "ekonomi"],
["bonus , piyasa, euro, tl, para, hesap", "ekonomi"],
["finans, dolar, gelir", "ekonomi"]]

countspor=aranacak(data,"Besiktas",1)
countekonom=aranacak(data,"ekonomi",1)
print("ekonomi adet:"+str(countekonom)+" Besiktas Adet:"+str(countspor))
sporagirlik=float(countspor)/(float(countspor)+float(countekonom))
ekonomagirlik=float(countekonom)/(float(countspor)+float(countekonom))
print("Besiktas :"+str(sporagirlik)+" ekonomi :"+str(ekonomagirlik))
kelimeci=kelimeler(data)
print(kelimeler(data))
sportoplamlam=0
spordeger=[]
for i in kelimeci:
sportoplamlam+=(arama(data,"Besiktas",i)+1)

for i in range(len(kelimeci)):
deger=float(arama(data,"Besiktas",kelimeci[i])+1)/float(sportoplamlam)
spordeger.append(deger)
print(str(kelimeci[i])+" icin "+str(deger))
ekonomtoplamlam=0
ekonomdeger=[]
for i in kelimeci:
ekonomtoplamlam+=(arama(data,"ekonomi",i)+1)
for i in range(len(kelimeci)):
deger=float(arama(data,"ekonomi",kelimeci[i])+1)/float(ekonomtoplamlam)
ekonomdeger.append(deger)
print(str(kelimeci[i])+" icin "+str(deger))
c_kelime=kelime.split(" ")
print(c_kelime)
sporcarpim=1
for i in c_kelime:
for x in range(len(kelimeci)):
if kelimeci[x]==i:
sporcarpim*=spordeger[x]
ekonomcarpim=1
for i in c_kelime:
for x in range(len(kelimeci)):
if kelimeci[x]==i:
ekonomcarpim*=ekonomdeger[x]
sporsonuc=sporcarpim*sporagirlik
ekonomsonuc=ekonomcarpim*ekonomagirlik
print("Besiktas cumle oran:"+str(sporcarpim)+" Oran:"+str(sporsonuc))
print("ekonomi cumle oran:"+str(ekonomcarpim)+" Oran:"+str(ekonomsonuc))
if sporsonuc<ekonomsonuc:
print(" ekonomi")
self.lblNaiveBayesSonuc.setText(" Ekonomi")
if sporsonuc>ekonomsonuc:
print(" Besiktas")
self.lblNaiveBayesSonuc.setText(" Besiktas")
if sporcarpim==1 and ekonomcarpim==1:
self.lblNaiveBayesSonuc.setText(" yok")

```

## 5.Normalizasyon

Veritabanlarında çok fazla sütun ve satırdan oluşan bir tabloyu tekrarlardan arındırmak için daha az satır ve sütun içeren alt kümelerine ayırıştırma işlemidir.

Ham Veri

		1	2	3	4	
1	6	148	72	35	0	
2	1	85	66	29	0	
3	0	183	64	0	0	
4	1	89	66	23	94	
5	0	137	40	35	168	
6	1	116	74	0	0	
7	0	78	50	32	88	
8	1	115	0	0	0	
9	0	197	70	45	543	
10	1	125	96	0	0	
11	1	110	92	0	0	
12	0	168	74	0	0	
13	1	139	80	0	0	
14	0	189	60	23	846	
15	1	166	72	19	175	
16	1	100	0	0	0	
17	1	118	84	47	230	
18	1	107	74	0	0	
19	1	103	30	38	83	
20	0	115	70	30	96	
21	1	126	88	41	235	
22	0	99	84	0	0	
23	0	196	90	0	0	
24	1	119	80	35	0	

Min-Max

Gerçekleştir

	1	2	3	4
1	0.67	1.49	0.73	0.35
2	0.11	0.86	0.67	0.29
3	0.89	1.85	0.65	0.0
4	0.11	0.9	0.67	0.23
5	0.0	1.38	0.41	0.35
6	0.56	1.17	0.76	0.0
7	0.33	0.79	0.51	0.32

Median

Gerçekleştir

	1	2	3	4
1	2.03	1.263	1.0	1.522
2	0.333	0.729	0.918	1.261
3	2.697	1.568	0.89	0.0
4	0.333	0.763	0.918	1.0
5	0.0	1.169	0.562	1.522
6	1.697	0.992	1.041	0.0
7	1.0	0.669	0.699	1.391

Z-Score

Gerçekleştir

	1	2	3	4
1	-0.469	-2.292	-2.827	-0.936
2	-1.029	-2.922	-2.887	-0.996
3	-0.249	-1.932	-2.907	-1.286
4	-1.029	-2.882	-2.887	-1.056
5	-1.139	-2.402	-3.147	-0.936
6	-0.579	-2.612	-2.797	-1.286
7	-0.809	-2.992	-3.047	-0.966

9.10.11.12.13.14.15.16.17.18.19.20.21.22.23.24.

IPython console Console

```
self.btnNormalizasyonVeriYukle.clicked.connect(self.NormalizasyonVeriYukle)
self.btnNormizasyonMinMax.clicked.connect(self.NormizasyonMinMax)
self.btnNormalizasyonZScore.clicked.connect(self.NormalizasyonZScore)
self.btnNormalizasyonMedian.clicked.connect(self.NormalizasyonMedian)
```

```
def NormalizasyonVeriYukle(self):
f = open('./veriler/diabetes.data')
X=[]
for i,row in enumerate(f.readlines()):
currentline = row.split(",")
temp=[]
for column_value in currentline:
temp.append(column_value)
X.append(temp)
X=np.array(X)
print "Array:",X.shape
self.X=X[:,8]
self.y=X[:,8]
self.veriYukle(self.X,self.y,self.twNormalizasyonVeriYukle)
def veriYukle(self,X,y,tablonormalize):
num_rows=len(X)
```

```

tablonormalize.clear()
tablonormalize.setColumnCount(8)
tablonormalize.setRowCount(num_rows) ##set number of rows
for rowNumber,row in enumerate(X):
    #row[1].encode("utf-8")
    tablonormalize.setItem(rowNumber, 0, QtGui.QTableWidgetItem(str(row[0])))
    tablonormalize.setItem(rowNumber, 1, QtGui.QTableWidgetItem(str(row[1])))
    tablonormalize.setItem(rowNumber, 2, QtGui.QTableWidgetItem(str(row[2])))
    tablonormalize.setItem(rowNumber, 3, QtGui.QTableWidgetItem(str(row[3])))
    tablonormalize.setItem(rowNumber, 4, QtGui.QTableWidgetItem(str(row[4])))
    tablonormalize.setItem(rowNumber, 5, QtGui.QTableWidgetItem(str(row[5])))
    tablonormalize.setItem(rowNumber, 6, QtGui.QTableWidgetItem(str(row[6])))
    tablonormalize.setItem(rowNumber, 7, QtGui.QTableWidgetItem(str(row[7])))
for rowNumber,row in enumerate(y):
    tablonormalize.setItem(rowNumber, 8, QtGui.QTableWidgetItem(str(row)))
def NormizasyonMinMax(self):
for s in range(0,8):
    first_column=self.X[:,s]
    max_value=float(max(first_column))
    min_value=float(min(first_column))
    print "max value:",max_value," min value:",min_value
    num_rows=len(self.X)
    for i,value in enumerate(first_column):
        normalize_value=((float(value)-min_value)/(max_value-min_value))
        first_column[i]=round(normalize_value,2)
    self.twNormizasyonMinMax.setColumnCount(8)
    self.twNormizasyonMinMax.setRowCount(num_rows)
    for rowNumber,row in enumerate(first_column):
        self.twNormizasyonMinMax.setItem(rowNumber, s, QtGui.QTableWidgetItem(str(row)))
def NormalizasyonZScore(self):
for s in range(0,8):
    colm= np.array(self.X[:,s]).astype(np.float)
    ui=np.mean(colm)
    ai=np.std(colm)
    print"Aritmetik ortalama:",ui," standart sapma:",ai
    num_rows=len(self.X)
    for i,value in enumerate(colm):
        normalize_zscor=float(value)-ui/ai
        colm[i]=float(round(normalize_zscor,3))
    self.twNormalizasyonZScore.setColumnCount(8)
    self.twNormalizasyonZScore.setRowCount(num_rows)
    for rowNumber,row in enumerate(colm):
        self.twNormalizasyonZScore.setItem(rowNumber, s, QtGui.QTableWidgetItem(str(row)))
def NormalizasyonMedian(self):
for s in range(0,8):
    column=np.array(self.X[:,s]).astype(np.float)
    med=np.median(column)
    print "Medyan: ",med
    num_rows=len(self.X)
    for i,value in enumerate(column):
        normalize_medyan=float(value)/med
        column[i]=float(round(normalize_medyan,3))
    self.twNormalizasyonMedian.setColumnCount(8)
    self.twNormalizasyonMedian.setRowCount(num_rows)
    for rowNumber,row in enumerate(column):
        self.twNormalizasyonMedian.setItem(rowNumber, s, QtGui.QTableWidgetItem(str(row)

```

## 6.Eğitim – Test Verisi Ve Random Forest

Eğitim ve test verisi: eldeki verilerin ne kadarı ile öğrenme işleminin yapılacağını. Karalaştırılır eğitim verisinin çok kullanılması ezberleme işlemine neden olabileceği gibi az kullanılması da öğrenme işleminin tam olarak yerine getirilmemesine sebep olabilir.

Random Forest: Karar ağacı algoritması olarak da kullanılmaktadır. veri madenciliği (data mining) ve makine öğrenmesi (machine learning) konularında geçen, karar ağacı öğrenmesi ile ilgilidir. Karar ağacı öğrenmesi sırasında, öğrenilen bilgi bir ağaç üzerinde modellenir. Bu ağacın bütün iç düğümleri (interior nodes) birer girdiyi ifade eder.

Ham Veriler

Yükle

	1	2	3	4
1	6	148	72	35
2	1	85	66	29
3	8	183	64	0
4	1	89	66	23
5	0	137	40	35
6	5	116	74	0
7	3	78	50	32
8	10	115	0	0
9	2	197	70	45
10	8	125	96	0
11	4	110	92	0
12	10	168	74	0
13	10	139	80	0
14	1	189	60	23
15	5	166	72	19
16	7	100	0	0
17	0	118	84	47
18	7	107	74	0
19	1	103	30	38
20	1	115	70	30
21	3	126	88	41
22	8	99	84	0
23	7	196	90	0
24	9	119	80	35

Test Veri Miktarı %

20

10

30

40

50

60

70

80

90

Uygula

Eğitim Verileri

	1	2	3	4	5
1	2	84	0	0	0
2	9	112	82	24	0
3	1	139	46	19	83
4	0	161	50	0	0
5	6	134	80	37	370
6	1	130	70	13	105
7	4	132	0	0	0
8	10	161	68	23	132
9	1	108	60	46	178

Test Verileri

	1	2	3	4	5
1	6	98	58	33	190
2	2	112	75	32	0
3	2	108	64	0	0
4	8	107	80	0	0
5	7	136	90	0	0
6	6	103	72	32	190
7	1	71	48	18	76
8	0	117	0	0	0
9	4	154	72	29	126

Random Forest ile Tahmin Yapma

Random Forest

Tahmin Oranı : 72.08

```
self.btnRandomForest.clicked.connect(self.RandomForest)
self.btnEgitimTestYukle.clicked.connect(self.EgitimTestYukle)
self.btnEgitimTestUygula.clicked.connect(self.EgitimTestUygula)
```

```
def EgitimTestYukle(self):
f = open('./veriler/veri.data')
X = []
self.Xveri = []
self.Yveri = []
for i, row in enumerate(f.readlines()):
currentline = row.split(",")
temp = []
for column_value in currentline:
temp.append(column_value)
X.append(temp)
self.Xveri.append(temp[:8])
self.Yveri.append(temp[8])
self.twEgitimTestYukle.clear()
self.twEgitimTestYukle.setColumnCount(9)
self.twEgitimTestYukle.setRowCount(len(X)) ##set number of rows
for rowNumber, row in enumerate(X):
```

```

self.twEgitimTestYukle.setItem(rowNumber, 0, QtGui.QTableWidgetItem(str(row[0])))
self.twEgitimTestYukle.setItem(rowNumber, 1, QtGui.QTableWidgetItem(str(row[1])))
self.twEgitimTestYukle.setItem(rowNumber, 2, QtGui.QTableWidgetItem(str(row[2])))
self.twEgitimTestYukle.setItem(rowNumber, 3, QtGui.QTableWidgetItem(str(row[3])))
self.twEgitimTestYukle.setItem(rowNumber, 4, QtGui.QTableWidgetItem(str(row[4])))
self.twEgitimTestYukle.setItem(rowNumber, 5, QtGui.QTableWidgetItem(str(row[5])))
self.twEgitimTestYukle.setItem(rowNumber, 6, QtGui.QTableWidgetItem(str(row[6])))
self.twEgitimTestYukle.setItem(rowNumber, 7, QtGui.QTableWidgetItem(str(row[7])))
self.twEgitimTestYukle.setItem(rowNumber, 8, QtGui.QTableWidgetItem(str(row[8])))

```

#### **def EgitimTestUygula(self):**

```
yuzde=float(float(self.cbEgitim.currentText())/100)
```

```
self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(self.Xveri, self.Yveri, test_size=yuzde, random_state=42)
```

```
self.twEgitim.clear()
```

```
self.twEgitim.setColumnCount(8)
```

```
self.twEgitim.setRowCount(len(self.X_train)) ##set number of rows TT_train_Table
```

```
for rowNumber, row in enumerate(self.X_train):
```

```
self.twEgitim.setItem(rowNumber, 0, QtGui.QTableWidgetItem(str(row[0])))
```

```
self.twEgitim.setItem(rowNumber, 1, QtGui.QTableWidgetItem(str(row[1])))
```

```
self.twEgitim.setItem(rowNumber, 2, QtGui.QTableWidgetItem(str(row[2])))
```

```
self.twEgitim.setItem(rowNumber, 3, QtGui.QTableWidgetItem(str(row[3])))
```

```
self.twEgitim.setItem(rowNumber, 4, QtGui.QTableWidgetItem(str(row[4])))
```

```
self.twEgitim.setItem(rowNumber, 5, QtGui.QTableWidgetItem(str(row[5])))
```

```
self.twEgitim.setItem(rowNumber, 6, QtGui.QTableWidgetItem(str(row[6])))
```

```
self.twEgitim.setItem(rowNumber, 7, QtGui.QTableWidgetItem(str(row[7])))
```

```
self.twTest.clear()
```

```
self.twTest.setColumnCount(8)
```

```
self.twTest.setRowCount(len(self.X_test)) ##set number of rows
```

```
for rowNumber, row in enumerate(self.X_test):
```

```
self.twTest.setItem(rowNumber, 0, QtGui.QTableWidgetItem(str(row[0])))
```

```
self.twTest.setItem(rowNumber, 1, QtGui.QTableWidgetItem(str(row[1])))
```

```
self.twTest.setItem(rowNumber, 2, QtGui.QTableWidgetItem(str(row[2])))
```

```
self.twTest.setItem(rowNumber, 3, QtGui.QTableWidgetItem(str(row[3])))
```

```
self.twTest.setItem(rowNumber, 4, QtGui.QTableWidgetItem(str(row[4])))
```

```
self.twTest.setItem(rowNumber, 5, QtGui.QTableWidgetItem(str(row[5])))
```

```
self.twTest.setItem(rowNumber, 6, QtGui.QTableWidgetItem(str(row[6])))
```

```
self.twTest.setItem(rowNumber, 7, QtGui.QTableWidgetItem(str(row[7])))
```

#### **def RandomForest(self):**

```
clf = RandomForestClassifier(max_depth=2, random_state=0)
```

```
clf.fit(self.X_train, self.y_train)
```

```
results=clf.predict(self.X_test)
```

```
self.lblRandomForest.setText(str(round(accuracy_score(self.y_test, results)*100,2)))
```

## 7.Parkinson

Parinson hastalığı ile ilgili olarak toplanmış veriler ile makine öğrenmesini gerçekleştirip. Hastalık tahmini yapma.

RUS-ROS VE RANDOMİZE VERİ

KNN-EN YAKIN KOMŞULUK

K-MEANS

NAİVE BAYES

NORMALİZASYON

EĞİTİM-TEST

PARKİNSON

Veri Yükle

DST

Eğitim

	1	2	3	4	
140627	399	206	0	513	1390
140628	399	206	0	481	1390
140629	399	206	0	419	1390
140630	399	206	0	290	1390

< >

Test

	1	2	3	4	
51989	413	191	0	750	960
51990	415	185	0	666	940
51991	417	179	0	565	940
51992	419	172	0	405	950

< >

DST

SST

Eğitim

	1	2	3	4	
134256	397	202	0	504	1450
134257	397	203	0	468	1450
134258	396	203	0	401	1440
134259	396	204	0	228	1440

< >

Test

	1	2	3	4	
87452	409	180	0	762	930
87453	410	180	0	675	920
87454	412	180	0	546	920
87455	413	181	0	379	900

< >

SST

STCP

Eğitim

	1	2	3	4	
48077	242	242	272	0	1460
48078	242	242	272	0	1460
48079	242	242	272	0	1460
48080	242	242	260	0	1350

< >

Test

	1	2	3	4	
62324	381	298	754	0	1290
62325	381	299	778	0	1380
62326	387	300	818	0	1430
62327	387	300	818	0	1430

< >

STCP

Tahminler

Tümünü Ver

Random Forest

Tahmin : :99.99

```
self.btnParkinsonVeriYukle.clicked.connect(self.ParkinsonVeriYukle)
self.btnParkinsonSST.clicked.connect(self.ParkinsonSST)
self.btnParkinsonRandomForest.clicked.connect(self.ParkinsonRandomForest)
self.btnParkinsonDST.clicked.connect(self.ParkinsonDST)
self.btnParkinsonSTCP.clicked.connect(self.ParkinsonSTCP)
self.btnParkinsonTumu.clicked.connect(self.ParkinsonTumu)
```

```
def ParkinsonTumu(self):
self.X_trainp, self.X_testptp, self.y_trainp, self.y_testp =train_test_split(self.Topalamx,self.Topalamy , test_size=0.30,
random_state=42)
def ParkinsonDST(self):
self.X_trainp, self.X_testptp, self.y_trainp, self.y_testp =self.DSTx, self.DSTxn,self.DSTy,self.DSTyn
def ParkinsonSTCP(self):
self.X_trainp, self.X_testptp, self.y_trainp, self.y_testp =self.STCPx, self.STCPxn,self.STCPy,self.STCPyn
def ParkinsonRandomForest(self):
clf = RandomForestClassifier(max_depth=None, random_state=0)
clf.fit(self.X_trainp,self.y_trainp)
results=clf.predict(self.X_testptp)
self.lblTahmin.setText("."+str(float("{0:.2f}".format(accuracy_score(self.y_testp,results)*100))))
def ParkinsonSST(self):
self.X_trainp, self.X_testptp, self.y_trainp, self.y_testp=self.SSTx, self.SSTxn,self.SSTy,self.SSTyn
def ParkinsonVeriYukle(self):
parkinson="./hw_dataset/parkinson/"
```



```

pathcontrol="./hw_dataset/control/"
parkinsonnew="./new_dataset/parkinson/"
SST=[] #0
DST=[] #1
STCP=[] #2
SST_train=[] #0
DST_train=[] #1
STCP_train=[] #2
SST_test=[] #0
DST_test=[] #1
STCP_test=[] #2
dosyalar=os.listdir(parkinson)
for dosya in dosyalar:
    f = open(parkinson+dosya)
    for i,row in enumerate(f.readlines()):
        currentline = row.split(";")
        temp=[]
        for column_value in currentline:
            temp.append(column_value)
            if(int(temp[len(temp)-1])==0):
                temp.remove(temp[len(temp)-1])
            SST_train.append(temp[:6])
            SST_test.append(1)
            temp.append(1)
            SST.append(temp)
            elif(int(temp[len(temp)-1])==1):
                temp.remove(temp[len(temp)-1])
                DST_train.append(temp[:6])
                DST_test.append(1)
                temp.append(1)
                DST.append(temp)
            elif(int(temp[len(temp)-1])==2):
                temp.remove(temp[len(temp)-1])
                STCP_train.append(temp[:6])
                STCP_test.append(1)
                temp.append(1)
                STCP.append(temp)
        dosyalar=os.listdir(pathcontrol)
        for dosya in dosyalar:
            f = open(pathcontrol+dosya)
            for i,row in enumerate(f.readlines()):
                currentline = row.split(";")
                temp=[]
                for column_value in currentline:
                    temp.append(column_value)
                    if(int(temp[len(temp)-1])==0):
                        temp.remove(temp[len(temp)-1])
                    SST_train.append(temp[:6])
                    SST_test.append(0)
                    temp.append(0)
                    SST.append(temp)
                    elif(int(temp[len(temp)-1])==1):
                        temp.remove(temp[len(temp)-1])
                        DST_train.append(temp[:6])
                        DST_test.append(0)
                        temp.append(0)
                        DST.append(temp)
                    elif(int(temp[len(temp)-1])==2):
                        temp.remove(temp[len(temp)-1])
                        STCP_train.append(temp[:6])
                        STCP_test.append(0)
                        temp.append(0)
                        STCP.append(temp)

```

```

SST_new=[] #0
DST_new=[] #1
STCP_new=[] #2
SST_train_new=[] #0
DST_train_new=[] #1
STCP_train_new=[] #2
SST_test_new=[] #0
DST_test_new=[] #1
STCP_test_new=[] #2
dosyalar=os.listdir(parkinsonnew)
for dosya in dosyalar:
f = open(parkinsonnew+dosya)
for i,row in enumerate(f.readlines()):
currentline = row.split(";")
temp=[]
for column_value in currentline:
temp.append(column_value)
if(int(temp[len(temp)-1])==0):
temp.remove(temp[len(temp)-1])
SST_train_new.append(temp[:6])
SST_test_new.append(1)
temp.append(1)
SST_new.append(temp)
elif(int(temp[len(temp)-1])==1):
temp.remove(temp[len(temp)-1])
DST_train_new.append(temp[:6])
DST_test_new.append(1)
temp.append(1)
DST_new.append(temp)
elif(int(temp[len(temp)-1])==2):
temp.remove(temp[len(temp)-1])
STCP_train_new.append(temp[:6])
STCP_test_new.append(1)
temp.append(1)
STCP_new.append(temp)
self.SSTx=SST_train
self.SSTy=SST_test
self.SSTxn=SST_train_new
self.SSTyn=SST_test_new
self.DSTx=DST_train
self.DSTy=DST_test
self.DSTxn=DST_train_new
self.DSTyn=DST_test_new
self.STCPx=STCP_train
self.STCPy=STCP_test
self.STCPxn=STCP_train_new
self.STCPyn=STCP_test_new
toplam_train=[]
toplam_test=[]
toplam_train.extend(SST_train)
toplam_test.extend(STCP_test)
toplam_train.extend(STCP_train_new)
toplam_test.extend(SST_test_new)
toplam_train.extend(DST_train)
toplam_test.extend(DST_test)
toplam_train.extend(DST_train_new)
toplam_test.extend(DST_test_new)
toplam_train.extend(STCP_train)
toplam_test.extend(SST_test)
toplam_train.extend(SST_train_new)
toplam_test.extend(STCP_test_new)
self.Topalamx=toplam_train
self.Topalamy=toplam_test

```

```

self.twParkinsonSSTegitim.clear()
self.twParkinsonSSTegitim.setColumnCount(7)
self.twParkinsonSSTegitim.setRowCount(len(SST_train)) ##set number of rows
for rowNumber,row in enumerate(SST_train):
self.twParkinsonSSTegitim.setItem(rowNumber, 0, QtGui.QTableWidgetItem(str(row[0])))
self.twParkinsonSSTegitim.setItem(rowNumber, 1, QtGui.QTableWidgetItem(str(row[1])))
self.twParkinsonSSTegitim.setItem(rowNumber, 2, QtGui.QTableWidgetItem(str(row[2])))
self.twParkinsonSSTegitim.setItem(rowNumber, 3, QtGui.QTableWidgetItem(str(row[3])))
self.twParkinsonSSTegitim.setItem(rowNumber, 4, QtGui.QTableWidgetItem(str(row[4])))
self.twParkinsonSSTegitim.setItem(rowNumber, 5, QtGui.QTableWidgetItem(str(row[5])))
self.twParkinsonSSTegitim.setItem(rowNumber, 6, QtGui.QTableWidgetItem(str(SST_test[rowNumber])))
self.twParkinsonDSTegitim.clear()
self.twParkinsonDSTegitim.setColumnCount(7)
self.twParkinsonDSTegitim.setRowCount(len(DST_train)) ##set number of rows
for rowNumber,row in enumerate(DST_train):
self.twParkinsonDSTegitim.setItem(rowNumber, 0, QtGui.QTableWidgetItem(str(row[0])))
self.twParkinsonDSTegitim.setItem(rowNumber, 1, QtGui.QTableWidgetItem(str(row[1])))
self.twParkinsonDSTegitim.setItem(rowNumber, 2, QtGui.QTableWidgetItem(str(row[2])))
self.twParkinsonDSTegitim.setItem(rowNumber, 3, QtGui.QTableWidgetItem(str(row[3])))
self.twParkinsonDSTegitim.setItem(rowNumber, 4, QtGui.QTableWidgetItem(str(row[4])))
self.twParkinsonDSTegitim.setItem(rowNumber, 5, QtGui.QTableWidgetItem(str(row[5])))
self.twParkinsonDSTegitim.setItem(rowNumber, 6, QtGui.QTableWidgetItem(str(DST_test[rowNumber])))
self.twParkinsonSTCPTTest.clear()
self.twParkinsonSTCPTTest.setColumnCount(7)
self.twParkinsonSTCPTTest.setRowCount(len(STCP_train_new)) ##set number of rows
for rowNumber,row in enumerate(STCP_train_new):
self.twParkinsonSTCPTTest.setItem(rowNumber, 0, QtGui.QTableWidgetItem(str(row[0])))
self.twParkinsonSTCPTTest.setItem(rowNumber, 1, QtGui.QTableWidgetItem(str(row[1])))
self.twParkinsonSTCPTTest.setItem(rowNumber, 2, QtGui.QTableWidgetItem(str(row[2])))
self.twParkinsonSTCPTTest.setItem(rowNumber, 3, QtGui.QTableWidgetItem(str(row[3])))
self.twParkinsonSTCPTTest.setItem(rowNumber, 4, QtGui.QTableWidgetItem(str(row[4])))
self.twParkinsonSTCPTTest.setItem(rowNumber, 5, QtGui.QTableWidgetItem(str(row[5])))
self.twParkinsonSTCPTTest.setItem(rowNumber, 6, QtGui.QTableWidgetItem(str(STCP_test_new[rowNumber])))
self.twParkinsonSSTTest.clear()
self.twParkinsonSSTTest.setColumnCount(7)
self.twParkinsonSSTTest.setRowCount(len(SST_train_new)) ##set number of rows
for rowNumber,row in enumerate(SST_train_new):
self.twParkinsonSSTTest.setItem(rowNumber, 0, QtGui.QTableWidgetItem(str(row[0])))
self.twParkinsonSSTTest.setItem(rowNumber, 1, QtGui.QTableWidgetItem(str(row[1])))
self.twParkinsonSSTTest.setItem(rowNumber, 2, QtGui.QTableWidgetItem(str(row[2])))
self.twParkinsonSSTTest.setItem(rowNumber, 3, QtGui.QTableWidgetItem(str(row[3])))
self.twParkinsonSSTTest.setItem(rowNumber, 4, QtGui.QTableWidgetItem(str(row[4])))
self.twParkinsonSSTTest.setItem(rowNumber, 5, QtGui.QTableWidgetItem(str(row[5])))
self.twParkinsonSSTTest.setItem(rowNumber, 6, QtGui.QTableWidgetItem(str(SST_test_new[rowNumber])))
self.twParkinsonDSTTest.clear()
self.twParkinsonDSTTest.setColumnCount(7)
self.twParkinsonDSTTest.setRowCount(len(DST_train_new)) ##set number of rows
for rowNumber,row in enumerate(DST_train_new):
self.twParkinsonDSTTest.setItem(rowNumber, 0, QtGui.QTableWidgetItem(str(row[0])))
self.twParkinsonDSTTest.setItem(rowNumber, 1, QtGui.QTableWidgetItem(str(row[1])))
self.twParkinsonDSTTest.setItem(rowNumber, 2, QtGui.QTableWidgetItem(str(row[2])))
self.twParkinsonDSTTest.setItem(rowNumber, 3, QtGui.QTableWidgetItem(str(row[3])))
self.twParkinsonDSTTest.setItem(rowNumber, 4, QtGui.QTableWidgetItem(str(row[4])))
self.twParkinsonDSTTest.setItem(rowNumber, 5, QtGui.QTableWidgetItem(str(row[5])))
self.twParkinsonDSTTest.setItem(rowNumber, 6, QtGui.QTableWidgetItem(str(DST_test_new[rowNumber])))
self.twParkinsonSTCPEgitim.clear()
self.twParkinsonSTCPEgitim.setColumnCount(7)
self.twParkinsonSTCPEgitim.setRowCount(len(STCP_train)) ##set number of rows
for rowNumber,row in enumerate(STCP_train):
self.twParkinsonSTCPEgitim.setItem(rowNumber, 0, QtGui.QTableWidgetItem(str(row[0])))
self.twParkinsonSTCPEgitim.setItem(rowNumber, 1, QtGui.QTableWidgetItem(str(row[1])))
self.twParkinsonSTCPEgitim.setItem(rowNumber, 2, QtGui.QTableWidgetItem(str(row[2])))
self.twParkinsonSTCPEgitim.setItem(rowNumber, 3, QtGui.QTableWidgetItem(str(row[3])))
self.twParkinsonSTCPEgitim.setItem(rowNumber, 4, QtGui.QTableWidgetItem(str(row[4])))
self.twParkinsonSTCPEgitim.setItem(rowNumber, 5, QtGui.QTableWidgetItem(str(row[5])))
self.twParkinsonSTCPEgitim.setItem(rowNumber, 6, QtGui.QTableWidgetItem(str(STCP_test[rowNumber])))
print "Kodlar basarili"

```

## KÜTÜPHANELER

```
from PyQt4 import QtGui
from PyQt4.uic.properties import QtCore
from PyQt4.QtCore import *
from PyQt4.QtGui import *
from PyQt4.QtGui import *
from PyQt4.QtGui import *
from PyQt4 import QtGui
from PyQt4 import QtCore
from PyQt4 import QtCore, QtGui
from PyQt4.QtGui import *
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import numpy as np
import matplotlib.pyplot as plt
import random
import os
import sys
import math
from sklearn.cross_validation import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

from makine import Ui_Dialog
```