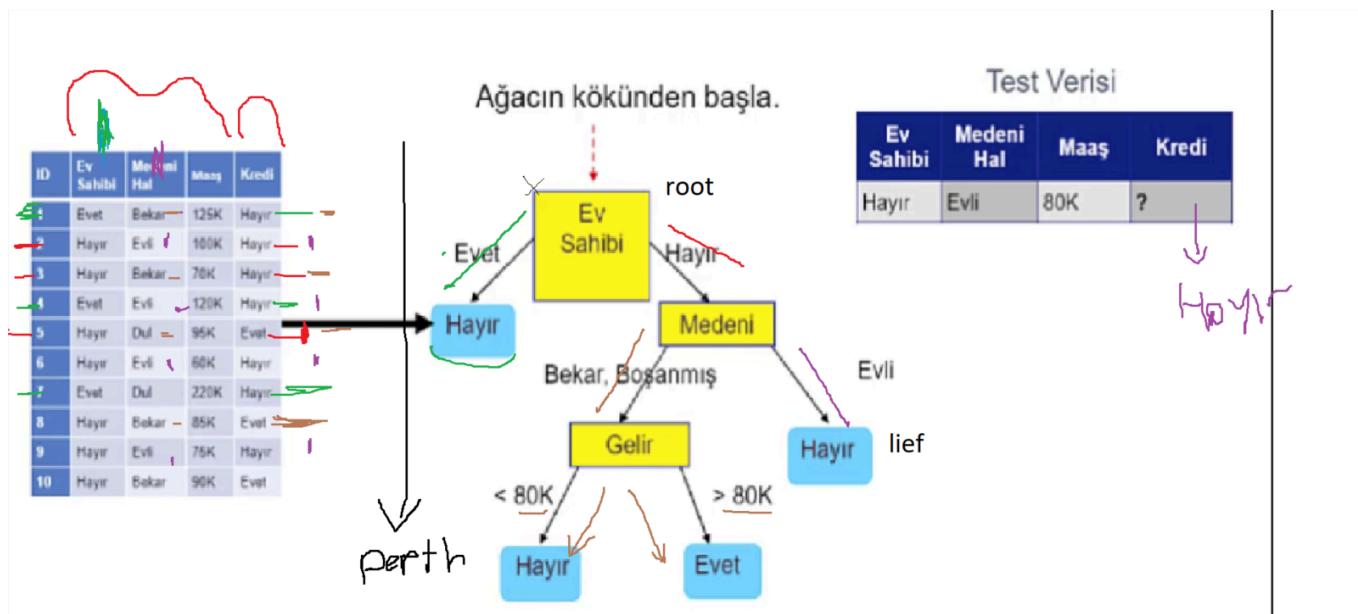
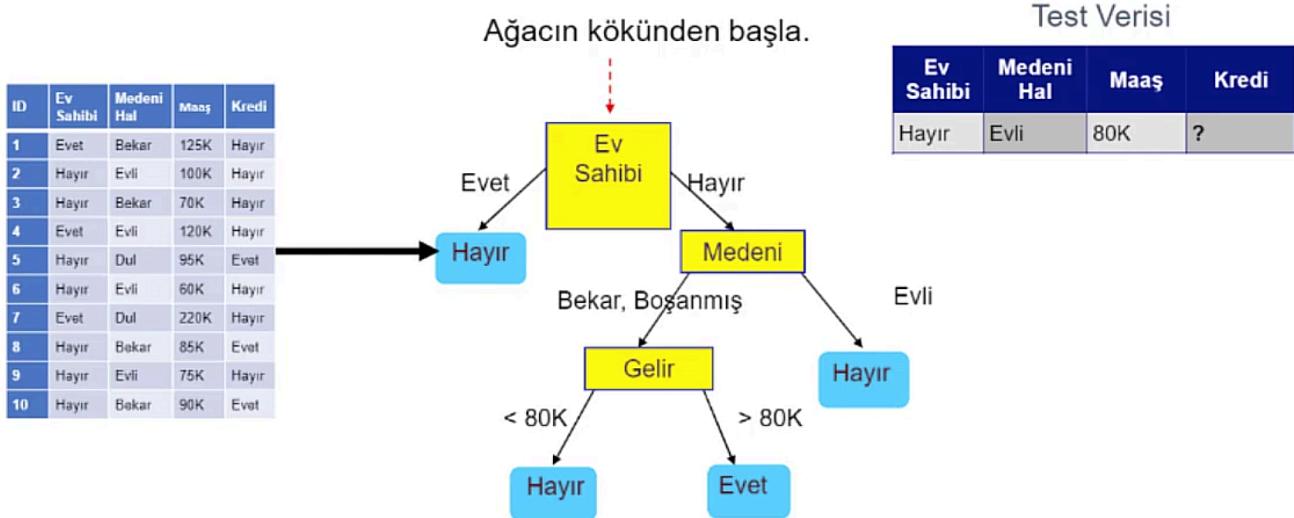
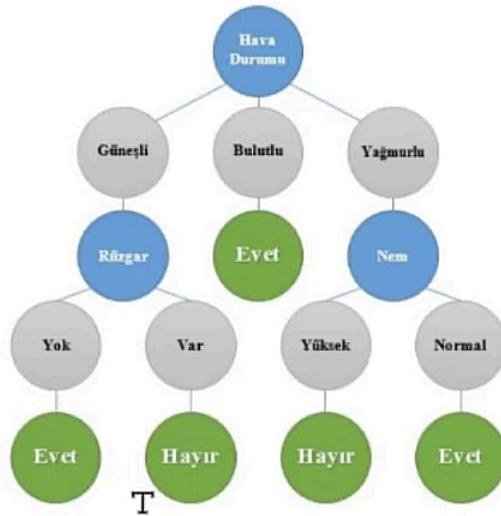


Karar Ağaçları



Özellikler				Hedef
Hava Durumu	Sıcaklık	Nem	Rüzgar	Futbol Oyna
Yağmurlu	Sıcak	Yüksek	Yok	Hayır
Yağmurlu	Sıcak	Yüksek	Var	Hayır
Bulutlu	Sıcak	Yüksek	Yok	Evet
Güneşli	İllik	Yüksek	Yok	Evet
Güneşli	Soğuk	Normal	Yok	Evet
Güneşli	Soğuk	Normal	Var	Hayır
Bulutlu	Soğuk	Normal	Var	Evet
Yağmurlu	İllik	Yüksek	Yok	Hayır
Yağmurlu	Soğuk	Normal	Yok	Evet
Güneşli	İllik	Normal	Yok	Evet
Yağmurlu	İllik	Normal	Yok	Evet
Bulutlu	İllik	Yüksek	Var	Evet
Bulutlu	Sıcak	Normal	Yok	Evet
Güneşli	İllik	Yüksek	Var	Hayır



$$E(T, X) = \sum_{c \in X} P(c)E(c)$$

		Futbol Oyna		Toplam
		Evet	Hayır	
Hava Durumu	Güneşli	3	2	5
	Bulutlu	4	0	4
	Yağmurlu	2	3	5
				14

$$\begin{aligned}
 & E(\text{FutbolOyna}, \text{HavaDurumu}) \\
 &= P(\text{Güneşli}) * E(3,2) + P(\text{Bulutlu}) * E(4,0) + P(\text{Yağmurlu}) * E(2,3) \\
 &\quad) \\
 & P(\text{Güneşli}) = 5/14 = 0.3571, E(3,2) = 0.971 \\
 & P(\text{Bulutlu}) = 4 / 14 = 0.286, E(4,0) = 0 \\
 & P(\text{Yağmurlu}) = 5 / 14 = 0.357, E(2,3) = 0.971 \\
 & E(\text{FutbolOyna}, \text{HavaDurumu}) = 0.694
 \end{aligned}$$

		Futbol Oyna		Toplam
		Evet	Hayır	
Nem	Yüksek	3	4	7
	Normal	6	1	7
				14

$$\begin{aligned}
 & E(\text{FutbolOyna}, \text{Nem}) = \\
 & P(\text{Yüksek}) * E(3,4) + P(\text{Normal}) * E(6,1) \\
 & E(3,4) = 0.985, P(\text{Yüksek}) = 0.5 \\
 & E(6,1) = 0.592, P(\text{Normal}) = 0.5 \\
 & E(\text{FutbolOyna}, \text{Nem}) = 0.788
 \end{aligned}$$

		Futbol Oyna		Topla
		Evet	Hayır	
Sıcaklık	Sıcak	2	2	4
	İllik	4	2	6
	Soğuk	3	1	4
				14

$$\begin{aligned}
 & E(\text{FutbolOyna}, \text{Sıcaklık}) = P(\text{Sıcak}) * E(2,2) + P(\text{İllik}) * E(4,2) \\
 & + P(\text{Soğuk}) * E(3,1) \\
 & E(3,2) = 1.000, P(\text{Güneşli}) = 0.286 \\
 & E(4,0) = 0.918, P(\text{Bulutlu}) = 0.429 \\
 & E(2,3) = 0.811, P(\text{Yağmurlu}) = 0.286
 \end{aligned}$$

$$E(\text{FutbolOyna}, \text{Sıcaklık}) = 0.911$$

		Futbol Oyna		Toplam
		Evet	Hayır	
Rüzgar	Yok	6	2	8
	Var	3	3	6
				14

$$\begin{aligned}
 & E(\text{FutbolOyna}, \text{Rüzgar}) = P(\text{Yok}) * E(6,2) + P(\text{Var}) * E(3,3) \\
 & E(3,4) = 0.811, P(\text{Yüksek}) = 0.571 \\
 & E(6,1) = 1.000, P(\text{Normal}) = 0.429 \\
 & E(\text{FutbolOyna}, \text{Rüzgar}) = 0.892
 \end{aligned}$$

Karar Ağacı teknikleri

ID3 Karar Ağacı Algoritması

$$P = \{p_1, p_2, p_n\}$$

$$H(S) = - \sum_{i=1}^n p_i \log_2 p_i$$

ID3 karar ağaç algoritmasının C4.5 ve C5.0 isminde iki tane versiyonu sıkılıkla kullanılmaktadır.
ID3 karar ağacı algoritmasında her düğümden çıkan dallar ile karar ağacı oluşturmaktadır.
Ağaçtaki dalların sayısı algoritmada tahmin edilecek sınıf sayısına eşittir. Karar ağacı algoritmasında yapraktaki hata (error) oranına göre budama işlemi yapılır.

Deney Sonuçları (S)	a_1	a_2	a_3
P_i	1/2	1/3	1/6

$S = \{a_1, a_2, a_3\}$ deney kümesini ifade etsin. a_1, a_2, a_3 olaylarının belirsizlikleri şöyle hesaplanır:

$$\frac{1}{2} \log_2 \frac{1}{2}, \quad \frac{1}{3} \log_2 \frac{1}{3}, \quad \frac{1}{6} \log_2 \frac{1}{6}$$

Aşağıda sekiz elemanlı S kümесini göz önüne alalım.

$$S=\{ \text{evet}, \text{evet}, \text{hayır}, \text{hayır}, \text{hayır}, \text{hayır}, \text{hayır}, \text{hayır} \}$$

Olasılıklar, iki adet "evet" değeri için,

$$P_1 = \frac{2}{8} = 0.25$$

Diğer altı adet "hayır" değeri için,

$$P_2 = \frac{6}{8} = 0.75$$

S için toplam entropi şu şekilde elde edilir;

$$\begin{aligned} H(S) &= -\{P_1 \log_2(P_1) + P_2 \log_2(P_2)\} \\ &= -(0.25\log_2(0.25)+0.75\log_2(0.75)) = 0.81128 \end{aligned}$$

C&RT Karar Ağacı Algoritması

Gini indeksi (dizini) veya Gini katsayısı, İtalyan istatistikçi Corrado Gini tarafından 1912'de geliştirilen istatistiksel bir ölçütür. Gini'ye dayalı ikili bölme işlemine göre çalışan bir karar ağacı algoritmasıdır. Bu algoritmada en son veya ucta olmayan her bir düğümde iki adet dal vardır. Hem Sınıflandırma hem de regresyon (sayısal sonuç) uygulamalarında kullanılır. Budama işlemi oluşturulan karar ağacı yapısına göre değişiklik gösterir.

CHAID Karar Ağacı Algoritması

Karar ağacı CHAID algoritması, istatistik tabanlı olarak G. V. Kass tarafından 1980'de geliştirilmiştir. Sınıflandırma ve regresyon uygulamalarında tercih edilir. CHAID algoritması, bağımsız değişkenlerin, birbirleriyle olan etkileşimini bulan bir tekniktir. CHAID algoritması dallanma kriterinde bağımlı değişken kategorik ise iki ya da daha çok grup arasında fark olup olmadığını tespit eden Ki-kare testine göre bölme işlemini gerçekleştirir.

SPRINT Karar Ağacı Algoritması

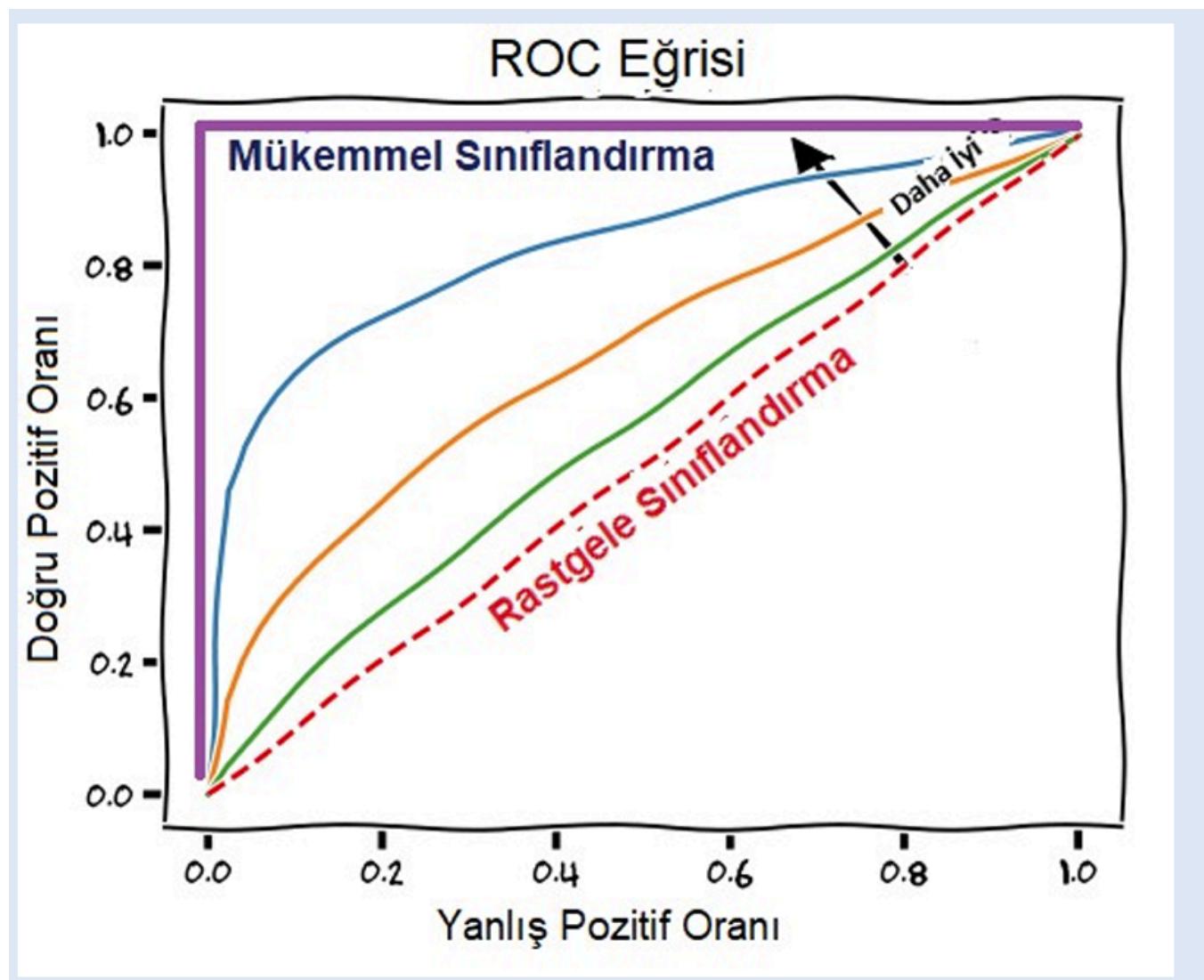
SPRINT algoritması 1996 yılında Shafer, Agrawal ve Mehta tarafından geliştirilip entropiye dayanmaktadır. SPRINT karar ağacı algoritması büyük veri kümeleri için ideal bir algoritmadır. Ağaç yapısında en iyi dallanma için her bir değişkene ait özellikler bir kez sıraya

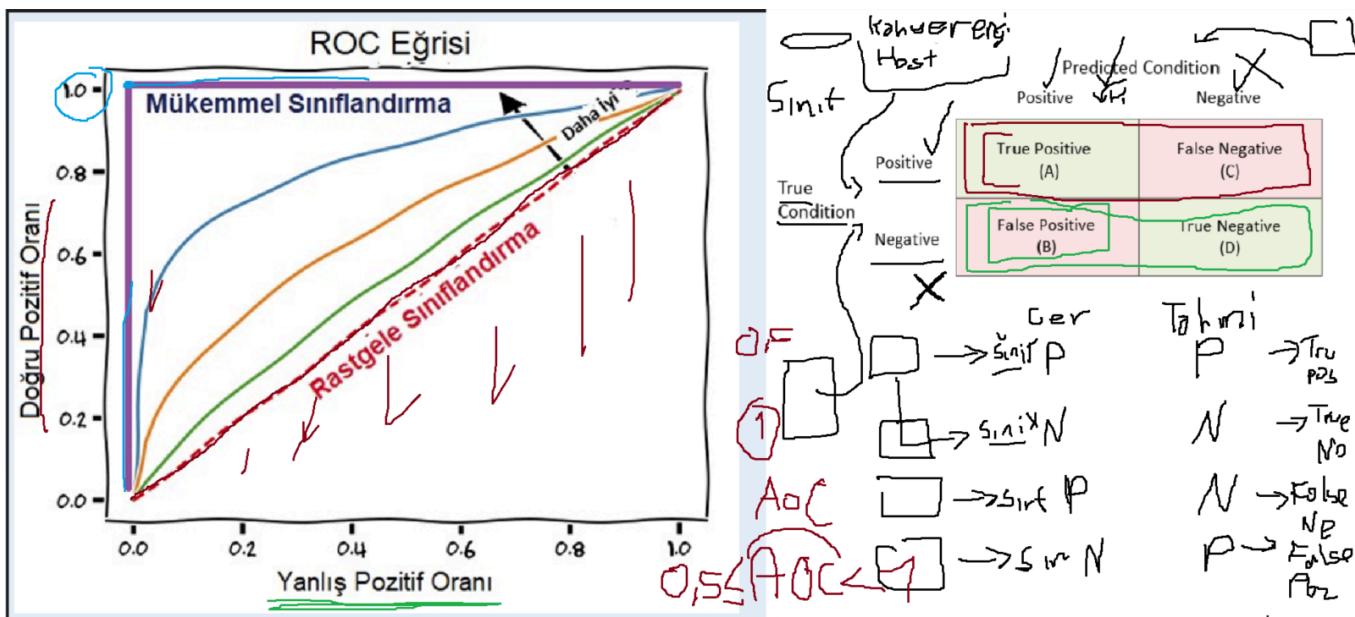
dizer ve karar ağaçları yapısı bu şekilde oluşur. Bu algoritmada her bir değişken için ayrı bir değişken listesi hazırlanır. Bölme işlemi tek bir özelliğin değerine göre saptanır.

SLIQ Karar Ağacı Algoritması

SLIQ karar ağacı algoritması 1996 yılında Agrawal, Mehta ve Rissanen tarafından geliştirilmiştir. Bu algoritma Gini teknigi ile nicel ve nitel veri tipleri kullanılabilmektedir. Ayrıca verilerin sıralanması aşamasında en iyi dallara ayırma tekniğini uygulamaktadır. Bu algoritma hızlı ölçüm yapan bir sınıflandırıcıya ve hızlı ağaç budama algoritmasına sahiptir.

Receiver Operating Characteristic-ROC

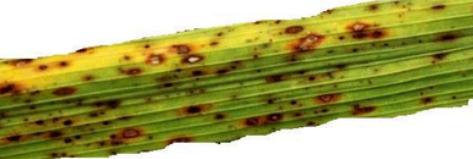


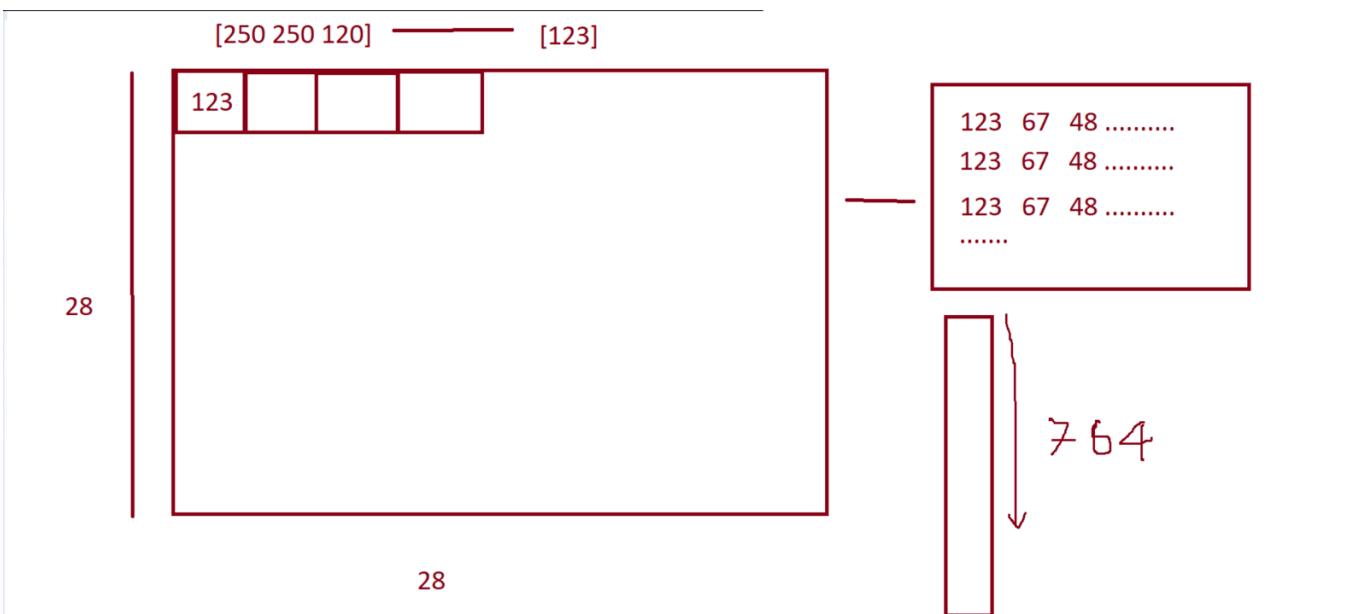


Doğru Pozitif Oranı (tpr): Doğru olarak tahmin edilen pozitif sınıfların sayısı,
 Doğru Negatif Oranı (tnr): Doğru olarak tahmin edilen negatif sınıfların sayısı,
 Yanlış Pozitif Oranı (fpr): Yanlış olarak tahmin edilen pozitif sınıfların sayısı,
 Yanlış Negatif Oranı (fnr): Yanlış olarak tahmin edilen negatif sınıfların sayısı.

aaaaaaaa

Kod ve Modelin testi

Veri Sayısı	GİRİŞ PARAMETRESİ	ÇIKIŞ PARAMETRESİ
	Pirinç yaprak hastalıkları görüntüsü	Hastalık Türü
1		0 (Bakteriyel yaprak yanıklığı)
2		0 (Bakteriyel yaprak yanıklığı)
...
40		0 (Bakteriyel yaprak yanıklığı)
41		1 (Kahverengi nokta)
42		1 (Kahverengi nokta)
...
80		



Veri setini indirme linki:

<https://www.kaggle.com/vbookshelf/rice-leaf-diseases/download>

```
def veri_donusturme(klasor_adi,sinif_adi):
```

```

goruntuler=dosya(klasor_adi)
goruntu_sinif=[]

for goruntu in goruntuler:

    goruntu_oku= img.open(goruntu).convert('L')
    gorunu_boyutlandirma=goruntu_oku.resize((28,28))
    goruntu_donusturme=np.array(gorunu_boyutlandirma).flatten()

    if sinif_adi=="bakteri_yaprak_yanik":
        veriler=np.append (goruntu_donusturme, [0])

    elif sinif_adi=="kahve_nokta":
        veriler=np.append (goruntu_donusturme, [1])

    elif sinif_adi=="yaprak_isi":
        veriler=np.append (goruntu_donusturme, [2])
    else:
        continue
    goruntu_sinif.append(veriler)

return goruntu_sinif

```

Bu fonksiyon belirtilen klasördeki görüntü dosyalarını okur, standart bir formata dönüştürür ve her görüntüyü ilgili sınıf etiketiyle (label) birleştirir.

Amacı: Görüntüleri Sayısal Veriye Dönüştürmek

Argüman : `klasor_adi` (görüntülerin bulunduğu klasör yolu) ve `sinif_adi` (görüntülerin ait olduğu sınıfın adı)

```
goruntuler = dosya(klasor_adi)
```

- Daha önce tanımlanan `dosya(yol)` fonksiyonunu kullanarak, belirtilen `klasor_adi` içindeki tüm görüntü dosyalarının **tam yollarını** içeren bir liste (`goruntuler`) elde edilir.

```
for goruntu in goruntuler:
    # ... işlemler ...
```

- `goruntuler` listesindeki her bir dosya yolu üzerinde bir döngü başlatılır.

```
goruntu_oku= img.open(goruntu).convert('L')
gorunu_boyutlandirma=goruntu_oku.resize((28, 28))
goruntu_donusturme=np.array(gorunu_boyutlandirma).flatten()
```

Döngü içindeki bu kısım, bir görüntüyü makine öğrenimi modelinin anlayabileceği sayısal bir vektöre dönüştürür

Gri tonlamalı bir `Image` nesnesi.

Görüntüyü **28x28 piksel** boyutuna küçültür (yeniden boyutlandırır).

Görüntüyü bir `numpy` dizisine dönüştürür. Ardından `flatten()` metodu ile bu 28x28'lük matrisi **tek boyutlu bir vektöre** dönüştürür. ($28 \times 28 = 784$ elemanlı bir vektör olur).

Veri Dönüşümü ve DataFrame Oluşturma

```
yanik_veri=veri_donusturme(bakteri_yaprak_yanik, "bakteri_yaprak_yanik")
yanik_veri_df=pd.DataFrame(yanik_veri)

kahve_nokta_veri=veri_donusturme(kahve_nokta, "kahve_nokta")
kahve_nokta_veri_df=pd.DataFrame(kahve_nokta_veri)

yaprak_isi_veri=veri_donusturme(yaprak_isi, "yaprak_isi")
yaprak_isi_veri_df=pd.DataFrame(yaprak_isi_veri)

tum_veri= pd.concat([yanik_veri_df, kahve_nokta_veri_df,yaprak_isi_veri_df ])
```

Kod, her hastalık sınıfı için aynı iki adımı tekrarlar:

Daha önce tanımladığınız `veri_donusturme` fonksiyonunu çağırır. `bakteri_yaprak_yanik` klasöründeki tüm görüntüleri okur, 28x28 gri tonlamalı vektörlere dönüştürür ve her vektörün sonuna 0 etiketini (sınıf numarasını) ekler.

Elde edilen bu vektör listesini kullanarak bir **Pandas DataFrame** oluşturur. Bu DataFrame'in her satırı tek bir görüntüyü temsil eder (784 piksel sütunu + 1 etiket sütunu).

Bu işlem, sırasıyla **yaprak yanıtı**, **kahve noktası** ve **yaprak ısı** sınıfları için tekrarlanır ve sonuçta üç ayrı DataFrame elde edilir:

- `yanik_veri_df` (Sınıf Etiketi: 0)
- `kahve_nokta_veri_df` (Sınıf Etiketi: 1)
- `yaprak_isi_veri_df` (Sınıf Etiketi: 2)

```
tum_veri = pd.concat([yanik_veri_df, kahve_nokta_veri_df, yaprak_isi_veri_df])
```

pd.concat() fonksiyonu, Pandas'ta birden fazla DataFrame'i birleştirmek için kullanılır. Bu satır, yukarıda oluşturulan üç ayrı DataFrame'i (**alt alta**) ekleyerek tek bir büyük DataFrame (`tum_veri`) oluşturur.

Sonuç

`tum_veri` DataFrame'i, makine öğrenimi modelinizi eğitmek için hazır olan nihai veri setinizdir. Bu veri seti şunları içerir:

Görüntü Verisi: Tüm sınıflara ait, 28x28 boyutuna indirgenmiş, gri tonlamalı ve düzleştirilmiş (flattened) görüntü piksel değerleri (toplam 784 sütun).

Sınıf Etiketi: Her satırın (görüntünün) son sütununda, o görüntünün hangi hastalığa ait olduğunu belirten sayısal etiket (0, 1 veya 2).

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.model_selection import GridSearchCV

Giris=np.array(tum_veri)[:, :784]
Cikis=np.array(tum_veri)[:, 784]

Giris_train, Giris_test, Cikis_train, Cikis_test = train_test_split(Giris,
Cikis,
test_size=0.2, random_state=109)

model= DecisionTreeClassifier()
clf = model.fit(Giris_train,Cikis_train)
Cikis_pred = clf.predict(Giris_test)

print("Doğruluk:",metrics.accuracy_score(Cikis_test, Cikis_pred))
```

```
Giris_train, Giris_test, Cikis_train, Cikis_test = train_test_split(Giris,
Cikis,
test_size=0.2, random_state=109)
```

- **Giris (X):** Tüm görüntülerin piksel değerlerini içeren veri setiniz (Bağımsız değişkenler).

- **Cikis (y)**: Tüm görüntülerin sınıf etiketlerini (0, 1, 2) içeren veri setiniz (Bağımlı değişkenler).
 - **test_size=0.2** : Veri setinin %20'sinin test seti olarak ayrılacağını belirtir. Geriye kalan %80'i ise eğitim seti olarak kullanılır.
 - **random_state=109** : Bu parametre, verinin her çalıştırışta **aynı şekilde** bölünmesini sağlayan bir tohum (seed) görevi görür. Bu, deneylerinizin **tekrarlanabilir** olmasını sağlar. (Eğer bu belirtilmezse, her çalıştırışta farklı bir rastgele bölme gerçekleşir.)
-

```
import matplotlib.pyplot as plt
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
from itertools import cycle

Cikis_test = label_binarize(Cikis_test, classes=[0, 1, 2])
Cikis_pred = label_binarize(Cikis_pred, classes=[0, 1, 2])

plt.figure(figsize=(60, 40), dpi=150)
n_classes=3
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):

    fpr[i], tpr[i], _ = roc_curve(Cikis_test[:, i], Cikis_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

colors = cycle(['blue', 'red', 'green'])

for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color,
              label=' {} Sinifina ait ROC eğrisi (AUC = {:.2f})'.format(i,
roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--')
```

```

plt.xlim([-0.05, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")

plt.show()

```

**Çok Sınıflı (Multiclass) Sınıflandırma için ROC Eğrileri (Receiver Operating Characteristic) çizmek

Kod, modelinizin her bir sınıfı ne kadar iyi ayırabildiğini gösteren bir grafik oluşturur.

```

Cikis_test = label_binarize(Cikis_test, classes=[0, 1, 2])
Cikis_pred = label_binarize(Cikis_pred, classes=[0, 1, 2])

```

ROC eğrileri doğası gereği ikili (binary) sınıflandırma için tasarlanmıştır. Üç sınıfınız (0, 1, 2) olduğu için, her sınıfı diğerlerinden ayırmak üzere veriyi "bir'e karşı hepsi" (one-vs-all) formatına dönüştürmeniz gerekir.

İşlem: `label_binarize` fonksiyonu, hem gerçek etiketleri (`Cikis_test`) hem de modelin tahminlerini (`Cikis_pred`) üç sütunlu bir matrise dönüştürür.

Örneğin, etiket **0** olan bir örnek artık `[1, 0, 0]` olarak temsil edilir. Etiket **1** olan bir örnek `[0, 1, 0]` olur.

```

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(Cikis_test[:, i], Cikis_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

```

- **Döngü:** Her bir sınıf (0, 1, 2) için ayrı ayrı döngü yapılır.
- **roc_curve :** Bu fonksiyon, ikili formata dönüştürülmüş test ve tahmin verilerini kullanarak o sınıf için ROC eğrisinin koordinatlarını hesaplar:
 - **FPR (False Positive Rate):** Yanlış Pozitif Oranı (Yatay Eksen)
 - **TPR (True Positive Rate):** Doğru Pozitif Oranı (Dikey Eksen)
- **auc :** Hesaplanan FPR ve TPR değerleri arasındaki alanı (Area Under the Curve) hesaplar. **AUC değeri ne kadar 1'e yakınsa, modelin o sınıfı ayırma başarısı o kadar yüksektir.**

```

colors = cycle(['blue', 'red', 'green'])

for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color,

```

```
label=' {0} Sinifina ait ROC eğrisi (AUC = {1:0.2f})' ''.format(i,  
roc_auc[i]))
```

- **Çizim:** Her bir sınıfın (0, 1, 2) FPR ve TPR değerleri, atanmış farklı bir renkle (blue , red , green) grafiğe çizilir.
 - **Etiketleme:** Her çizgiye, hangi sınıfa ait olduğu ve o sınıfa ait hesaplanan AUC değeri etiketlenir.

```
import numpy as np

import PIL.Image as img

import os

import pandas as pd

bakteri_yaprak_yanik="C:/Users/User/Desktop/deneyapegitim/yapayzeka/kod/cice_leaf_diseases/Bacterial leaf blight/"

kahve_nokta="C:/Users/User/Desktop/deneyapegitim/yapayzeka/kod/cice_leaf_diseases/Brown spot/"

yaprak_isi="C:/Users/User/Desktop/deneyapegitim/yapayzeka/kod/cice_leaf_diseases/Leaf smut/"

def dosya(yol):

    return [os.path.join(yol,f) for f in os.listdir(yol)]

def veri_donusturme(klasor_adi,sinif_adi):

    goruntuler=dosya(klasor_adi)

    goruntu_sinif=[]

    for goruntu in goruntuler:

        goruntu_oku= img.open(goruntu).convert('L')

        gorunu_boyutlandirma=goruntu_oku.resize((28,28))
```

```
goruntu_donusturme=np.array(gorunu_boyutlandirma).flatten()

if sinif_adi=="bakteri_yaprak_yanik":

    veriler=np.append (goruntu_donusturme, [0])

elif sinif_adi=="kahve_nokta":

    veriler=np.append (goruntu_donusturme, [1])

elif sinif_adi=="yaprak_isi":

    veriler=np.append (goruntu_donusturme, [2])

else:

    continue

goruntu_sinif.append(veriler)

return goruntu_sinif

yanik_veri=veri_donusturme(bakteri_yaprak_yanik,"bakteri_yaprak_yanik")

yanik_veri_df=pd.DataFrame(yanik_veri)

kahve_nokta_veri=veri_donusturme(kahve_nokta,"kahve_nokta")

kahve_nokta_veri_df=pd.DataFrame(kahve_nokta_veri)

yaprak_isi_veri=veri_donusturme(yaprak_isi,"yaprak_isi")

yaprak_isi_veri_df=pd.DataFrame(yaprak_isi_veri)

tum_veri= pd.concat([yanik_veri_df, kahve_nokta_veri_df,yaprak_isi_veri_df ])

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import train_test_split

from sklearn import metrics

from sklearn.model_selection import GridSearchCV
```

```
Giris=np.array(tum_veri)[:, :784]

Cikis=np.array(tum_veri)[:, 784]

Giris_train, Giris_test, Cikis_train, Cikis_test = train_test_split(Giris,
Cikis,

test_size=0.2, random_state=109)

model= DecisionTreeClassifier()

clf = model.fit(Giris_train,Cikis_train)

Cikis_pred = clf.predict(Giris_test)

print("Doğruluk:",metrics.accuracy_score(Cikis_test, Cikis_pred))

import matplotlib.pyplot as plt

from sklearn.preprocessing import label_binarize

from sklearn.metrics import roc_curve, auc

from itertools import cycle

Cikis_test = label_binarize(Cikis_test, classes=[0, 1, 2])

Cikis_pred = label_binarize(Cikis_pred, classes=[0, 1, 2])

plt.figure(figsize=(60, 40),dpi=150)

n_classes=3

fpr = dict()

tpr = dict()

roc_auc = dict()

for i in range(n_classes):

    fpr[i], tpr[i], _ = roc_curve(Cikis_test[:, i], Cikis_pred[:, i])

    roc_auc[i] = auc(fpr[i], tpr[i])
```

```
colors = cycle(['blue', 'red', 'green'])

for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color,
              label=' {} Sinifina ait ROC eğrisi (AUC = {:.2f})'.format(i,
roc_auc[i]))


plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([-0.05, 1.0])
plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")

plt.show()
```