Karar ağaçlarında kullanılan ayrım (saflık) ölçütlerinden biridir.. Gini bir veri kümesinin heterojenliğini (karmaşıklığını) ölçer. Bir veri kümesindeki elemanların farklı sınıflara dağılımını değerlendirerek, en iyi ayrım noktalarını bulmaya çalışır.

* Düşük Gini Değeri = Veri kümesinin dağılımının homojen olduğunu gösterir.
* Yüksek Gini Değeri = Veri kümesinin dağılımının heterojen olduğunu gösterir.

***! Karar ağacında daha düşük ‘gini değeri’ seçilmeli.***

yazı tipi, metin, beyaz, grafik içeren bir resim

Açıklama otomatik olarak oluşturuldu

[1] Gini formülü

j = Veri kümesindeki sınıf sayısını ifade eder.

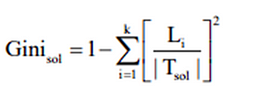
P(j) = Veri kümesindeki sınıfın olasılığını ifade eder.

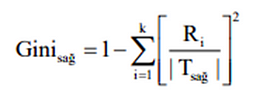
Aşamalar:

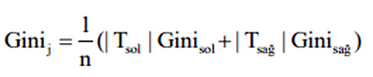
1- Sol ve sağ olacak şekilde kategorileri ayırmalıyız.

2- Ayırdığımız kategorilere göre, ‘Evet’, ‘Hayır’ ve ‘Toplam’ sayıları hesaplanır.

3- Gini hesaplaması için kullanılan formüller:







**Örnek Uygulama**

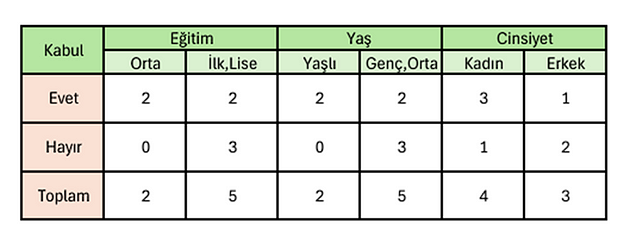


[2] Örnek Datasetimiz

1- Kategorileri Ayırmalıyız.

* Eğitim için = Gini(Sol) = Orta | Gini(Sağ) ∈ {İlk,Yüksek}
* Yaş için = Gini(Sol) = Yaşlı | Gini(Sağ) ∈ {Genç,Orta}
* Cinsiyet için = Gini(Sol) = Erkek | Gini(Sağ) = Kadın

2- Ayırdığımız tablolara göre ‘Evet’-‘Hayır’- ‘Toplam’ sayılarına göre

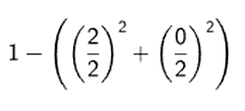


3- Gini(Sol) hesaplamaları yapılır.

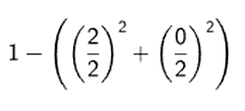
Gini-Eğitim(Sol) = 1 — [(|Orta(Evet)/|Orta|)² + |Orta(Hayır)/Orta|²]

Gini-Yaş(Sol) = 1 — [(|Yaşlı(Evet)/|Yaşlı|)² + |Yaşlı(Hayır)/Yaşlı|²]

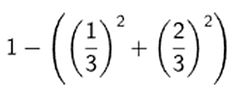
Gini-Cinsiyet(Sol) = 1 — [(|Erkek(Evet)/|Erkek|)² + |Erkek(Hayır)/Erkek|²]



Gini-Eğitim(Sol) = 0



Gini-Yaş(Sol) = 0



Gini-Cinsiyet(Sol) = 0.444

4- Gini(Sağ) hesaplamaları yapılır.

Gini-Eğitim(Sağ) = 1 — [(|{İlk,Lise}(Evet)/|{İlk,Lise}|)² + |{İlk,Lise}(Hayır)/{İlk,Lise}|²]

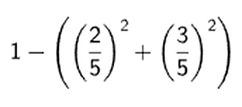
Gini-Yaş(Sağ) = 1 — [(|{Genç,Orta}(Evet)/|{Genç,Orta}|)² + |{Genç,Orta}(Hayır)/{Genç,Orta}|²]

Gini-Cinsiyet(Sağ) = 1 — [(|Kadın(Evet)/|Kadın|)² + |Kadın(Hayır)/Kadın|²]

yazı tipi, beyaz, diyagram, daire içeren bir resim

Açıklama otomatik olarak oluşturuldu

Gini-Eğitim(Sağ) = 0.48



Gini-Yaş(Sağ) = 0.48

yazı tipi, diyagram, beyaz, daire içeren bir resim

Açıklama otomatik olarak oluşturuldu

Gini-Cinsiyet(Kadın) = 0.375

5- Gini ölçütleri hesaplanmalı

Gini Ölçütleri :

Gini Eğitim = 1/7 (2x0)(5x0.48) = 0.3428

Gini Yaş = 1/7 (2x0)(5x0.48) = 0.3428

Gini Cinsiyet = 1/7 (3x0.375)(4x0.444) = 0.4144

! Eğer eşitlik durumu oluşursa, seçilen özelliklerin önemi kalmaz

Seçilen kısım ‘Yaş => Yaşlı’ olarak dallandırılıyor.

**Kod ile Gini**

*Kütüphaneler*

import pandas as pd  
import numpy as np  
from graphviz import Digraph

*Gini saflığını hesaplayan fonksiyonumuz*

def gini\_impurity(labels):  
 if len(labels) == 0:  
 return 0  
 unique\_labels, counts = np.unique(labels, return\_counts=True)  
 probabilities = counts / len(labels)  
 gini = 1 - np.sum(probabilities\*\*2)  
 return gini

*Bilgi Kazancı hesaplayan fonksiyonumuz*

def information\_gain(parent\_labels, left\_child\_labels, right\_child\_labels):  
 parent\_gini = gini\_impurity(parent\_labels)  
 left\_gini = gini\_impurity(left\_child\_labels)  
 right\_gini = gini\_impurity(right\_child\_labels)  
  
 weighted\_gini = (len(left\_child\_labels) / len(parent\_labels)) \* left\_gini + \  
 (len(right\_child\_labels) / len(parent\_labels)) \* right\_gini  
  
 info\_gain = parent\_gini - weighted\_gini  
 return info\_gain

*Ağacı oluşturan fonksiyonumuz*

def build\_tree(df, target, depth=0):  
  
 if len(df[target].unique()) == 1:  
 return df[target].iloc[0]  
  
  
 if len(df) == 0 or len(df.columns) == 1:  
 return df[target].mode()[0]  
  
  
 best\_gain = 0  
 best\_feature = None  
 for feature in df.drop(columns=[target]).columns:  
 unique\_values = df[feature].unique()  
 for value in unique\_values:  
 left\_split = df[df[feature] == value][target].tolist()  
 right\_split = df[df[feature] != value][target].tolist()  
 gain = information\_gain(df[target].tolist(), left\_split, right\_split)  
 if gain > best\_gain:  
 best\_gain = gain  
 best\_feature = (feature, value)  
  
  
 if best\_feature is None or best\_gain == 0:  
 return df[target].mode()[0]  
  
  
 feature, value = best\_feature  
 left\_df = df[df[feature] == value].copy()  
 right\_df = df[df[feature] != value].copy()  
 remaining\_df = df.drop(left\_df.index).drop(right\_df.index)  
  
  
  
  
 left\_tree = build\_tree(left\_df, target, depth + 1)  
 right\_tree = build\_tree(right\_df, target, depth + 1)  
  
 return {f"{feature} == {value}": {"Yes": left\_tree, "No": right\_tree}}

*Ağacı çizen fonksiyonumuz*

def visualize\_tree(tree, graph=None, parent=None, label=None):  
 if graph is None:  
 graph = Digraph()  
  
 if not isinstance(tree, dict):  
 graph.node(str(id(tree)), str(tree), shape="box")  
 if parent is not None:  
 graph.edge(str(parent), str(id(tree)), label=label)  
 return graph  
  
 for key, subtree in tree.items():  
 graph.node(str(id(key)), key)  
 if parent is not None:  
 graph.edge(str(parent), str(id(key)), label=label)  
 visualize\_tree(subtree["Yes"], graph, id(key), "Yes")  
 visualize\_tree(subtree["No"], graph, id(key), "No")  
  
 return graph

*Kuralları yazdıran fonksiyonumuz*

def print\_rules(tree, parent\_rule=""):  
 if not isinstance(tree, dict):   
 print(f"Rule: {parent\_rule} => {tree}")  
 return  
  
 for key, subtree in tree.items():  
 if "Yes" in subtree:  
 current\_rule = f"{parent\_rule} AND {key}" if parent\_rule else key  
 print\_rules(subtree["Yes"], current\_rule)  
  
 if "No" in subtree:  
 negated\_rule = f"{parent\_rule} AND NOT {key}" if parent\_rule else f"NOT {key}"  
 print\_rules(subtree["No"], negated\_rule)

*Tahmin ile veri setini kontrol eden fonksiyonumuz*

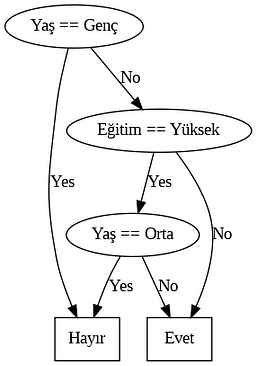
def predict(tree, sample):  
 if not isinstance(tree, dict):  
 return tree  
  
 for key, subtree in tree.items():  
 feature, value = key.split(" == ")  
 if sample[feature] == value:  
 return predict(subtree["Yes"], sample)  
 else:  
 return predict(subtree["No"], sample)  
  
  
predictions = []  
for index, row in df.iterrows():  
 sample = {  
 'Cinsiyet': row['Cinsiyet'],  
 'Yaş': row['Yaş'],  
 'Eğitim': row['Eğitim'],  
 }  
 predictions.append(predict(tree, sample))  
  
df['Tahmin'] = predictions  
print(df[['Kabul', 'Tahmin']])  
  
  
accuracy = (df['Kabul'] == df['Tahmin']).sum() / len(df)  
print(f"Doğruluk Oranı: {accuracy}")

*Diğer kodları*

# Load dataset  
file\_path = input("Enter the path to your CSV file: ")  
df = pd.read\_csv(file\_path)  
  
  
target\_col = input("Enter the target column name: ")  
  
# Karar ağacını oluştur  
# Eksik değerleri temizleme  
df.dropna(subset=[target\_col], inplace=True)  
tree = build\_tree(df, target\_col)  
  
# Karar ağacını görselleştir  
graph = visualize\_tree(tree)  
graph.render("decision\_tree2", format="png", cleanup=True)  
print\_rules(tree)



Oluşan Kurallar



Karar Ağacı