**Twoing Algoritması ve Örnek Uygulama**

**Twoing Algoritması Nedir?**

karar ağacı algoritmalarında kullanılan bir bölme ölçütüdür. Temel amacı, veriyi bölerken farklı sınıfları birbirinden mümkün olduğunca iyi ayırmak ve her bir alt kümenin içindeki homojenliği artırmaktır. Bu sayede, ağacın doğruluğu artar ve genelleme yapabilme yeteneği gelişir.

**Uygunluk Ölçütü Nedir?**

* Bir düğümdeki veri kümesinin nasıl bölüneceğini belirlerken kullanılan matematiksel bir kriterdir.

**Örnek Uygulama**

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu[1] Memnuniyet Durumu

Öncelikle aday bölünmeler seçilmelidir. Hedef değişkenin (örneğin memnuniyet durumu) doğru tahmin edebilmesi için veriyi hangi özellikler ve bu özelliklerin hangi değerleri (kategorik veya sürekli) üzerinden böleceğimize karar verilir. Veriyi sınıflandıran karar ağacında, her bir özelliğin farklı değerleri ile nasıl bölüneceğini seçmeniz gereklidir.

Tsol kavramı için sütunlar içindeki değerler ile oluşturulur.

! Tsol Kavramı genellikle karar ağacında bir bölünme (split) yapılırken kullanılır. Burada **iki alt küme** oluşturulması ve bu alt kümelerin daha iyi ayrıştırılması hedeflenir.

Tabloya göre bölünmeler:

1- Gelir = Normal

2- Gelir = Büyük

3- Gelir = Küçük

4- Eğitim = İlk

5- Eğitim = Orta

6- Eğitim = Lise

7- Sektör = Bilişim

8- Sektör = İnşaat

Tsağ kavramı kendiliğinden oluşur.

! Tsağ kavramı Tsol seçildikten sonra kendiliğinden oluşur. Bu yapılan sol bölünmelere göre sağ bölünmeler oluşur.

1- Gelir ∈ **{**Küçük,Büyük**}**

2- Gelir ∈ {Küçük, Normal}

3- Gelir ∈ {Normal, Büyük}

4- Eğitim ∈ {Orta, Lise}

5- Eğitim ∈ {İlk, Lise}

6- Eğitim ∈ {Orta, Lise}

7- Sektör = İnşaat

8- Sektör = Bilişim

Tsol için oluşan tablo

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

* P(Sol): Verilen aday bölünmenin tablodaki oranı
* Evet: Yapılan aday bölünmenin kaç adet ‘Evet’ karar sınıfı üyesi olduğunu gösterir.
* Hayır: Yapılan aday bölünmenin kaç adet ‘Hayır’ karar sınıfı üyesi olduğunu gösterir.
* P(Evet): Bu bölünme üzerindeki kaç adet olduğuna göre Evet Sayısı / Toplam Veri Sayısı ile hesaplanır.
* P(Hayır): Bu bölünme üzerindeki kaç adet olduğuna göre Hayır Sayısı / Toplam Veri Sayısı ile hesaplanır.

Tsağ için oluşan tablo

metin, sayı, numara, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Tsağ için oluşan tablo aslında Tsol içindeki değerler için tümleyendir.

Bu oluşan tablolara göre “Uygunluk Ölçütü” hesaplanmalı.

“ 2 x P(Sol) x P(Sağ) x [ |P(Evet|Tsol)-P(Evet|Tsol) | + |P(Hayır|Tağ)-P(Hayır|Tsağ)| ] ”

ile her aday bölünme için hesaplanmalıdır.

metin, sayı, numara, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Buradan çıkan sonuca göre ‘Sektör=Bilişim’ ve ‘Sektör=İnşaat’ aynı kazanca sahip olup maksimum uygunluk ölçütüne sahip olduğu için ona göre bölünebilir.

diyagram, daire, metin, taslak içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Kod İle Twoing**

Kütüphaneler

import pandas as pd  
import numpy as np  
import graphviz

Uygunluk ölçütünü hesaplayan algoritmalar

def calculate\_twoing\_criterion(data, target\_column):  
  
 best\_split\_combination = None  
 best\_criterion\_value = -1  
  
 split\_columns = [col for col in data.columns if col != target\_column]  
  
 for col in split\_columns:  
 for split\_value in data[col].unique():  
 data\_copy = data.copy()  
 data\_copy[col] = data\_copy[col].apply(lambda x: split\_value if x == split\_value else 'Other')  
  
 group\_counts = data\_copy.groupby(col)[target\_column].value\_counts()  
 probabilities = group\_counts / group\_counts.groupby(level=0).sum()  
 probability\_table = probabilities.unstack(fill\_value=0)  
  
 if len(probability\_table.columns) == 1:  
 probability\_table['dummy'] = 0  
  
 probability\_table.columns = ['P(No)', 'P(Yes)'][:len(probability\_table.columns)]  
 group\_ratios = data\_copy[col].value\_counts() / len(data\_copy)  
 probability\_table['Group Ratio'] = group\_ratios  
  
 try:  
 normal\_yes\_prob = probability\_table.loc[split\_value, 'P(Yes)']  
 diger\_yes\_prob = probability\_table.loc['Other', 'P(Yes)']  
 normal\_no\_prob = probability\_table.loc[split\_value, 'P(No)']  
 diger\_no\_prob = probability\_table.loc['Other', 'P(No)']  
 except KeyError:  
 continue  
  
 total\_diff = abs(normal\_yes\_prob - diger\_yes\_prob) + abs(normal\_no\_prob - diger\_no\_prob)  
 criterion\_value = (2 \* total\_diff \* group\_ratios[split\_value] \* group\_ratios['Other'])  
  
 if criterion\_value > best\_criterion\_value:  
 best\_criterion\_value = criterion\_value  
 best\_split\_combination = (col, split\_value)  
  
 return best\_split\_combination, best\_criterion\_value

Ağacı oluşturan kodumuz

def build\_tree(data, target\_column):  
  
 if len(data[target\_column].unique()) == 1:  
 return data[target\_column].iloc[0]  
  
 best\_split\_combination, best\_criterion\_value = calculate\_twoing\_criterion(data, target\_column)  
  
 if best\_split\_combination is None:  
 return data[target\_column].value\_counts().idxmax()  
  
 split\_column = best\_split\_combination[0]  
 split\_value = best\_split\_combination[1]  
  
 left\_split = data[data[split\_column] == split\_value]  
 right\_split = data[data[split\_column] != split\_value]  
  
 if len(left\_split) == 0 or len(right\_split) == 0:  
 return data[target\_column].value\_counts().idxmax()  
  
 left\_leaf = build\_tree(left\_split, target\_column)  
 right\_leaf = build\_tree(right\_split, target\_column)  
  
 tree = {split\_column: {split\_value: left\_leaf, 'Other': right\_leaf}}  
 return tree

Ağacı görselleştiren kodumuz

def visualize\_tree(tree):  
  
 dot = graphviz.Digraph(comment="Decision Tree")  
  
 def add\_nodes\_edges(subtree, parent\_name, parent\_label=""):  
 for key, value in subtree.items():  
 # Düğümün etiketi  
 edge\_label = str(key)  
  
 if isinstance(value, dict):  
 child\_name = f"{parent\_name}\_{key}"  
 dot.node(child\_name, label=key)  
 dot.edge(parent\_name, child\_name, label=edge\_label)  
 add\_nodes\_edges(value, child\_name)  
 else:  
 leaf\_name = f"{parent\_name}\_{key}\_leaf"  
 dot.node(leaf\_name, label=str(value), shape="ellipse")  
 dot.edge(parent\_name, leaf\_name, label=edge\_label)  
  
 root\_name = "root"  
 dot.node(root\_name, label="root")  
 add\_nodes\_edges(tree, root\_name)  
  
 return dot

Ağaçtan kuralları oluşturan kodumuz

def print\_tree\_rules(tree, indent=""):  
  
 if isinstance(tree, dict): # Düğümde bölünme varsa  
 for feature, sub\_tree in tree.items():  
 # 'feature' (özellik) üzerinde karar yapılmış  
 for value, subtree in sub\_tree.items():  
 print(f"{indent}If {feature} = {value}:")  
 print\_tree\_rules(subtree, indent + " ")  
 else: # Bir yaprak (sonuç sınıfı) ile karşılaşıldığında  
 print(f"{indent}Result: {tree}")

Diğer Kodlar

file\_path = input("Enter the path of the CSV file: ")   
target\_column = input("Enter the name of the target column: ")   
output\_image\_name = input("Enter the image file name (without extension): ")   
  
data = pd.read\_csv(file\_path)  
  
tree = build\_tree(data, target\_column)  
print("Decision tree:", tree)  
  
decision\_tree\_graph = visualize\_tree(tree)  
decision\_tree\_graph.render(output\_image\_name, format="png", view=True)

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturulduOluşan Kurallardiyagram, çizim, taslak içeren bir resim

Açıklama otomatik olarak oluşturulduKarar Ağacı