T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN EDEBİYAT FAKÜLTESİ
İSTATİSTİK BÖLÜMÜ

IST 4171-Veri Sınıflandırma Yöntemleri

HAZIRLAYANLAR

BERNA NUR ÇETİNKAYA 19023074
FATMA ERDEM 19023017
LEYLA TUNÇ 20023036
TUĞÇE YAZICI 19023050
UMUT AKGÜMÜŞ 19023010

Dersi Veren Öğretim Üyesi: GÜLHAYAT GÖLBAŞI ŞİMŞEK

VERİ

Veri Kümesi Hakkında

Bu veri seti gözlemlerden manuel olarak toplanır. Sütün kalitesini tahmin etmek için makine öğrenimi modelleri oluşturmamıza yardımcı olur.
Bu veri seti pH, Sıcaklık, Tat, Koku, Yağ, Bulanıklık ve Renk olmak üzere 7 bağımsız değişkenden oluşur.
Genellikle sütün derecesi veya kalitesi bu parametrelere bağlıdır. Bu parametreler sütün tahmin analizinde hayati bir rol oynar.

Hedef

Tat, Koku, Yağ ve Bulanıklık optimum koşullarla karşılanıyorsa 1, aksi takdirde 0 atanır.
Veri setinde sıcaklık ve ph gerçek değerleri verilmiştir.

Sütün kalitesini tahmin etmek amacıyla istatistiksel ve tahmine dayalı modeller oluşturmak için veri ön işleme ve veri artırma teknikleri uygulamalıyız.

Sütün kalitesinin sınıflandırılması:

- Düşük (Kötü)
- Orta (Orta)
- Yüksek (İyi)

Veri seti 7 değişkenden oluşur:

- pH
- Sıcaklık
- Tat
- Koku
- Yağ
- Bulanıklık
- Renk

Sınıflandırma:

- KNN Algoritması
- Random Forest
- Karar Ağacı

Veri Analizi

[8]:
```
import pandas as pd
```

[9]:
```
import numpy as np
```

[10]:
```
import seaborn as sns
```

[14]:
```
data = pd.read_csv('/Users/fatmaerdem/Downloads/milknew.csv',sep=';')
```

[15]:
```
data.shape
```

[15]:
```
(1059, 8)
```

[16]:
```
data.head(5)
```

[16]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour	Grade
0	6,6	35	1	0	1	0	254	high
1	6,6	36	0	1	0	1	253	high
2	8,5	70	1	1	1	1	246	low
3	9,5	34	1	1	0	1	255	low
4	6,6	37	0	0	0	0	255	medium

[17]:
```
data.describe()
```

[17]:

	Temprature	Taste	Odor	Fat	Turbidity	Colour
count	1059.000000	1059.000000	1059.000000	1059.000000	1059.000000	1059.000000
mean	44.226629	0.546742	0.432483	0.671388	0.491029	251.840415
std	10.098364	0.498046	0.495655	0.469930	0.500156	4.307424
min	34.000000	0.000000	0.000000	0.000000	0.000000	240.000000
25%	38.000000	0.000000	0.000000	0.000000	0.000000	250.000000
50%	41.000000	1.000000	0.000000	1.000000	0.000000	255.000000
75%	45.000000	1.000000	1.000000	1.000000	1.000000	255.000000
max	90.000000	1.000000	1.000000	1.000000	1.000000	255.000000

[18]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1059 entries, 0 to 1058
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   pH          1059 non-null   object
 1   Temprature  1059 non-null   int64 
 2   Taste       1059 non-null   int64 
 3   Odor        1059 non-null   int64 
 4   Fat         1059 non-null   int64 
 5   Turbidity   1059 non-null   int64 
 6   Colour      1059 non-null   int64 
 7   Grade       1059 non-null   object
dtypes: int64(6), object(2)
memory usage: 66.3+ KB
```

[19]:
```
data.isna().sum()
```

[19]:
```
pH            0
Temprature    0
Taste         0
Odor          0
Fat           0
Turbidity     0
Colour        0
Grade         0
dtype: int64
```

[20]:
```
data.groupby('Grade').size()
```

[20]:
```
Grade
high      256
low       429
medium    374
dtype: int64
```

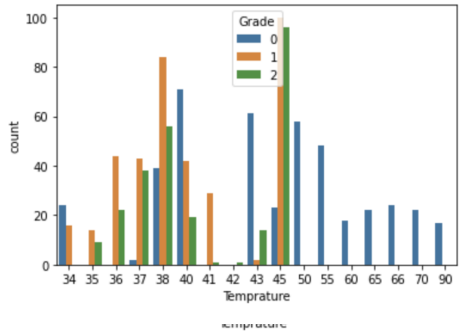
[21]:
```
data['Grade']=data['Grade'].map({'low':0,'medium':1,'high':2})
data.head()
```

[21]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour	Grade
0	6,6	35	1	0	1	0	254	2
1	6,6	36	0	1	0	1	253	2
2	8,5	70	1	1	1	1	246	0
3	9,5	34	1	1	0	1	255	0
4	6,6	37	0	0	0	0	255	1

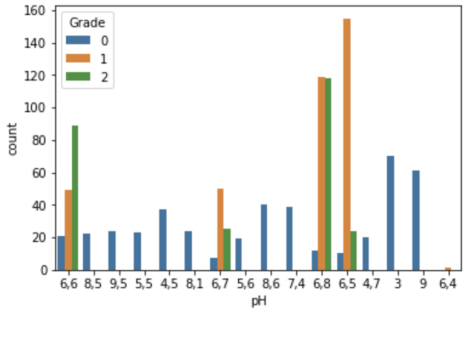
[22]:
```
sns.countplot(x=data['Temprature'],hue=data['Grade'], palette = "tab10")
```

[22]: <AxesSubplot:xlabel='Temprature', ylabel='count'>



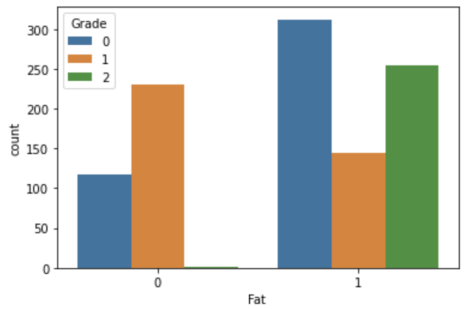
[23]:
```
sns.countplot(x=data['pH'],hue=data['Grade'], palette = "tab10")
```

[23]: <AxesSubplot:xlabel='pH', ylabel='count'>



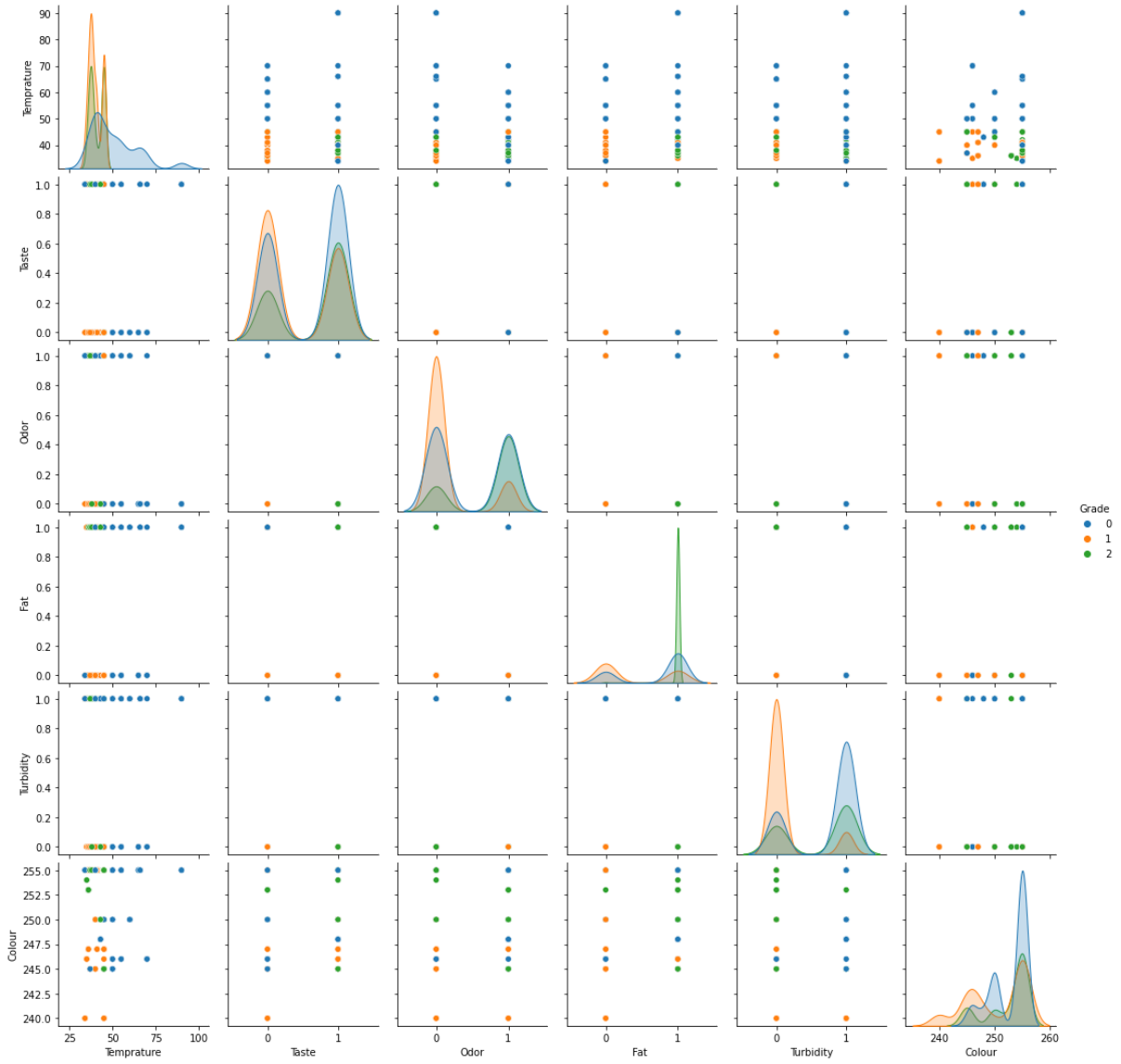
[24]:
```
sns.countplot(x=data['Fat '],hue=data['Grade'], palette = "tab10")
```

[24]: <AxesSubplot:xlabel='Fat ', ylabel='count'>



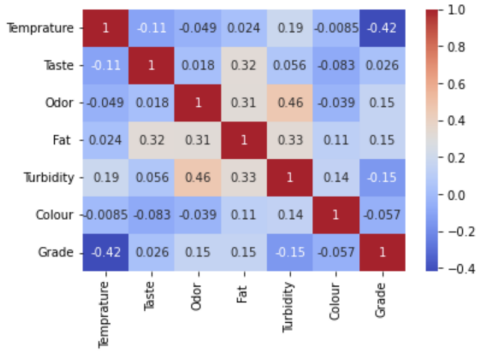
[26]:
```
sns.pairplot(data=data, hue='Grade', palette = "tab10")
```

[26]: <seaborn.axisgrid.PairGrid at 0x7fa2006f5cd0>



[27]:
```
sns.heatmap(data.corr(), annot=True, cmap='coolwarm')
```

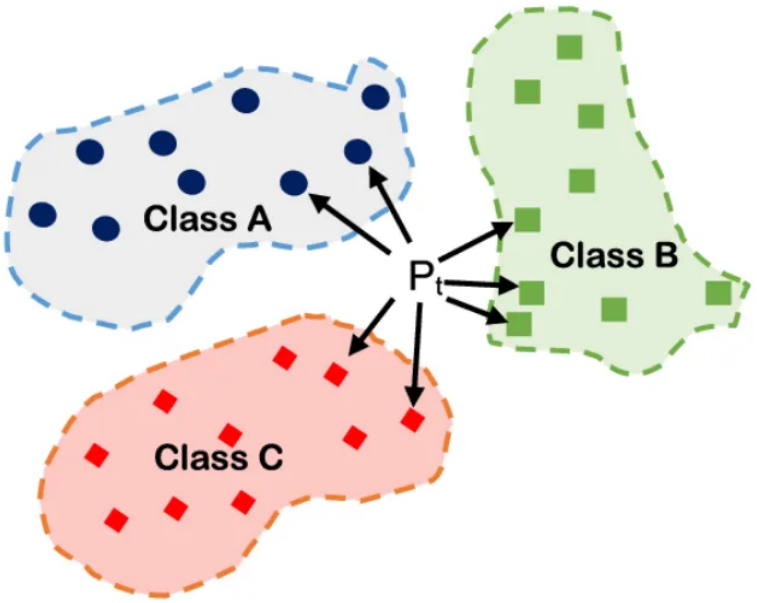
[27]: <AxesSubplot:>



KNN (K-Nearest Neighbors) Algoritması

K-en yakın komşu (k-nearest neighbors, KNN) algoritması, gözlemlerin birbirlerine olan benzerlikleri üzerinden tahminlerin yapıldığı gözetimli makine öğrenmesi modellerinde regresyon ve sınıflandırma problemlerinde kullanılan bir algoritmadır.

Bu algoritma kapsamında tahminde bulunmak istediğimiz gözlem birimine en yakın K adet farklı gözlem birimi tespit edilir ve bu K adet gözlem biriminin bağımlı değişkenleri üzerinden ilgili gözlem için tahminde bulunulur.

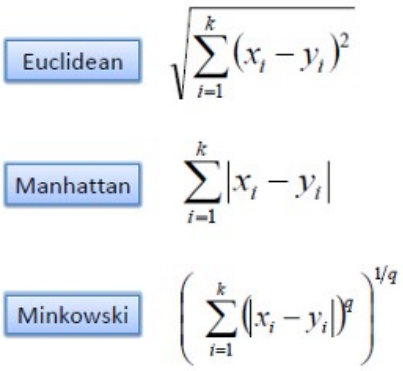


Yukarıdaki resim incelendiğinde, genel itibari ile, benzer olan sınıfların birbiri ile yakın mesafede oldukları gözlemlenmektedir. Dolayısıyla KNN algoritması da bu gözleme dayanarak, yeni gelecek tahminler bu noktalara yakınlığa göre tahminlenir.

KNN (K-Nearest Neighbors) Algoritması iki temel değer üzerinden tahmin yapar;

Distance (Uzaklık): Tahmin edilecek noktanın diğer noktalara uzaklığı hesaplanır. Bunun için Minkowski uzaklık hesaplama fonksiyonu kullanılır.

Minkowski mesafesi : Minkowski mesafesi Öklid uzayında bir metriktir, Öklid mesafesi ve Manhattan mesafesinin bir genelleştirilmesi ile oluşturulur. Burada eklenen p parametresine göre işlemler yapılır p=1 olduğunda, Minkowski mesafesi, Manhattan mesafesine eşit olur. p=2 olduğunda, Minkowski mesafesi, Öklid mesafesine eşit olur.



K (komşuluk sayısı): En yakın kaç komşu üzerinden hesaplama yapılacağını söyleriz. K değeri sonucu direkt etkileyecektir. K 1 olursa overfit etme olasılığı çok yüksek olacaktır. Çok büyük olursa da çok genel sonuçlar verecektir. Bu sebeple optimum K değerini tahmin etmek problemin asıl konusu olarak karşımızda durmaktadır.,

KNN (K-Nearest Neighbors) Algoritması ile üretilmiş bir modelin başarısını ölçmek için genel olarak kullanılan 3 adet indikatör vardır.

Jaccard Index: Doğru tahmin kümesi ile gerçek değer kümesinin kesişim kümesinin bunların birleşim kümesine oranıdır.1 ile 0 arası değer alır. 1 en iyi başarım anlamına gelir.
F1-Score: Confusion Matriks üzerinden hesaplanan Precision ve Recall değerlerinden hesaplanır. Pre=TP/(TP+FP) Rec=TP/(TP+FN) F1-Score= 2((Pre Rec)/(Pre+Rec)) 1 ile 0 arası değer alır. 1 en iyi başarım anlamına gelir.
Log Loss: Logistic Regresyon sonunda tahminlerin olasılıkları üzerinden Log Loss değeri hesaplanır. 1 ile 0 arası değer alır. Yukarıdaki iki değerden farklı olarak 0 en iyi başarım anlamına gelir.

KNN algoritmasının avantajları :

Algoritma basit ve uygulanması kolaydır.
Bir model oluşturmaya, birkaç parametreyi ayarlamaya veya ek varsayımlar yapmaya gerek yoktur.
Algoritma çok yönlüdür. Sınıflandırma, regresyon ve arama için kullanılabilir

KNN algoritmasının Dezavantajları:

Örneklerin ve/veya tahmin edicilerin/bağımsız değişkenlerin sayısı arttıkça algoritma önemli ölçüde yavaşlar. Hesaplama yönünden pahalıdır (Computationally expensive)
Yüksek memory ihtiyacı. Çünkü bütün train datasını hafızada tutar
Örneklem sayısı artınca, tahmin uzun sürebilir.

In [25]:
```
# Splitting Data
X=data.drop(['Grade'],axis=1)
y=data['Grade']
X.head()
```

Out[25]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour
0	6.6	35	1	0	1	0	254
1	6.6	36	0	1	0	1	253
2	8.5	70	1	1	1	1	246
3	9.5	34	1	1	0	1	255
4	6.6	37	0	0	0	0	255

In [26]:
```
y.head()
```

Out[26]:
```
0    2
1    2
2    0
3    0
4    1
Name: Grade, dtype: int64
```

Mevcut verimizi bağımlı değişken ve bağımsız değişken olarak ayrırız .Daha sonra uygulayacağımız KNN algoritması için gerekli kütüphaneleri tanımlarız.

In [32]:
```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
```

In [33]:
```
# Standardization using StandardScaler
sc=StandardScaler()
sc.fit(X)
xscaled=sc.transform(X)
X_scaled=pd.DataFrame(data=xscaled,columns=data.columns[:-1])
# Split Test and Train data
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
X_train.head()
```

Out[33]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour
331	6.8	40	1	0	1	0	245
44	6.6	45	0	0	0	1	250
307	6.8	45	1	1	1	0	245
323	9.0	43	1	0	1	1	250
1009	9.0	43	1	0	1	1	250

Veriyi standartlaştırma (standard scaling) ya da z-score normalleştirmesi, veri setindeki özellikleri (değişkenleri) bir standart normal dağılım (ortalama=0, standart sapma=1) etrafında yeniden ölçeklendirmek için kullanılan bir yöntemdir. Bu işlem, veri setindeki özelliklerin farklı birimlere sahip olması durumunda, modelin performansını artırmak ve karşılaştırmaları daha anlamlı hale getirmek amacıyla uygulanır.

Standartlaştırma yapıldıktan sonra veriyi bağımsız değişkenler ve bağımlı değişken için test ve train olarak böleriz. Böylece algoritma verinin train kısmından öğrendiklerini test verisi üzerinde test edebilsin.

```
In [34]: knn=KNeighborsClassifier(n_neighbors=5)
         knn.fit(X_train,y_train)
         knn_preds=knn.predict(X_test)

In [36]: knn_score=accuracy_score(y_test,knn_preds)*100

In [37]: print(knn_score)

         98.58490566037736
```

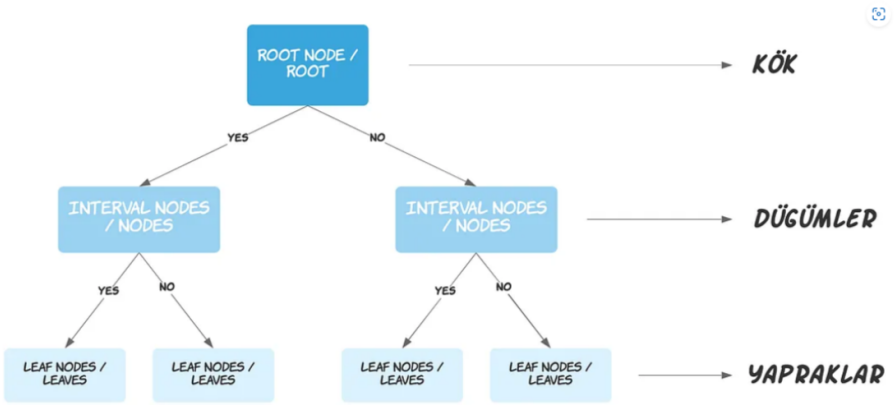
Karar Ağacı

Karar ağacı, belirli bir soruna yönelik tüm potansiyel çözümleri haritalandıran akış şeması benzeri bir diyagramdır. Bir dizi karar almanın tüm olası sonuçlarını karşılaştırarak en uygun hareket tarzını belirlemeye yardımcı olmak için kullanılır. Karar ağacı, her bir iç düğümün bir özelliği her bir dalın bir karar kuralını ve her bir yaprak düğümünün sonucu temsil ettiği akış şeması benzeri bir ağaç yapısıdır. Bir karar ağacındaki en üst düğüm kök düğüm olarak bilinir ve herhangi bir alt düğümü olmayan düğümler yaprak düğüm olarak adlandırılır. Karar ağacı algoritması kök düğümden başlar ve bir yaprak düğüme ulaşana kadar girdi özellik değerlerine göre bir karar vererek ağaç boyunca ilerler. Yaprak düğümdeki değer, tahmin edilen çıktı değerini temsil eder.Bir karar ağacı algoritmasında kök düğümün yaprak düğüme ayrılması aşağıdaki gibi açıklanabilir:

Kök Düğüm: Kök düğüm tüm veri kümesini temsil eder ve ağacı başlatmak için kullanılır. Ağacın başlangıç noktasıdır ve verileri maksimum bilgi kazancı veya minimum Gini Impurity sağlayan özelliğe göre böler.

İç Düğüm: Her bir iç düğüm, verileri iki veya daha fazla alt kümeye ayıran bir özelliği temsil eder. Bölme işlemi özelliğin değerine göre gerçekleştirilir ve her bir gözlemin izleyeceği yolu belirler. İç düğüm daha sonra birden fazla alt düğüme bölünür.

Yaprak Düğüm: Yaprak düğüm, verilerin daha fazla bölünemeyen bir alt kümesini temsil eder. Kendisine ulaşan gözlemler için nihai tahmini içerir. Tahmin, alt kümedeki çoğunluk sınıfına veya hedef değişkenin ortalama değerine dayanır.



Karar Ağacı Nasıl Çalışır ?

Karar ağaçları, verilerdeki kalıpları belirlemeye yardımcı olan bir tür makine öğrenimi modelidir. Bir dizi girdi değeri alarak ve ardından ağacın mevcut veriler için en iyi kararın ne olduğuna inandığına bağlı olarak bunları farklı dallara ayırarak çalışır. Ağacın verdiği her karar, mümkün olan en iyi seçeneğin belirlenmesinde bir adım olarak düşünülebilir. Bir karar ağacının nihai sonucu genellikle her bir girdi değerinin (önceden belirlenmiş) çıktı değerlerinden birine karşılık gelme olasılığını yansıtan belirli bir çıktı değerleri kümesidir.Karar ağaçlarının diğer modellere göre sahip olduğu önemli bir avantaj, karmaşık veri setleriyle hızlı bir şekilde başa çıkabilmeleridir. Bunun nedeni, verileri daha küçük parçalara bölmek için basit kurallar kullanmaları ve daha sonra bu parçalar içindeki kalıpları aramalarıdır. Bu yöntem genellikle "böl ve yönet" olarak adlandırılır ve karar ağaçlarının sorunları diğer model türlerinden çok daha hızlı bir şekilde ele almasını sağlar.Karar ağaçlarının bir diğer büyük avantajı da çok yönlü olmalarıdır. Bu, örüntü tanıma, tahmin ve sınıflandırma dahil ancak bunlarla sınırlı olmamak üzere çeşitli farklı görevler için kullanılabilecekleri anlamına gelir.

Karar Ağacı Oluşturma

Makine öğreniminde karar ağacı, karar vermeye yardımcı olan bir veri yapısıdır. Ağaç, girdi verilerinin düğüm adı verilen daha küçük kümelere bölünmesi ve ardından düğümlerin bazı kriterlere göre gruplandırılmasıyla oluşturulur. Ağaçtaki her bir düğümde verilen kararlar daha sonra nihai kararı üretmek için birleştirilir. İki ana ağaç türü vardır: ikili ve çok sınıflı. İkili bir karar ağacı her düğümde iki değer (doğru/yanlış) üzerinde eğitim verirken, çok sınıflı bir karar ağacı ikiden fazla değer (iki veya daha fazla sınıf) üzerinde eğitim verir.Bir karar ağacı oluşturmanın en basit yolu, girdi verilerini eğitim ve test kümelerine bölmektir. Ardından, her düğümde test setini iki parçaya ayırın: bir eğitim seti ve bir doğrulama seti. Bu bilgileri, modeli eğitmek için hangi özelliklerin kullanılacağına ve doğrulama için hangi özelliklerin kullanılacağına karar vermek için kullanabiliriz.Karar ağaçları genellikle çok fazla verinin olduğu ancak doğru bir tahmin yapmak için yeterli bilginin olmadığı problemler için kullanılır.Karar ağaçları, bir düğümü iki veya daha fazla alt düğümde bölmeye karar vermek için birden fazla algoritma kullanır. Alt düğümlerin oluşturulması, alt düğümlerin homojenliğini arttırır. Başka bir ifadeyle, düğümün saflığının hedef değişkenlere göre arttığını söyleyebiliriz.Algoritma seçimi, hedef değişkenin tipine dayanır. Karar ağaçlarında en sık kullanılan algoritmalar; kategorik değişkenler için Entropi, Gini, Sınıflandırma Hatası; sürekli değişkenler için ise En Küçük Kareler yöntemi şeklindedir.Entropi, verilerinizle ilgili belirsizliğin bir ölçüsüdür. Sezgisel olarak, bir veri kümesinin yalnızca bir etiketi varsa (örneğin, her yolcu hayatta kaldı), daha düşük bir entropiye sahip olduğunu düşünebiliriz. Dolayısıyla verilerimizi, entropiyi en aza indirecek bir şekilde bölmemiz gerekmektedir. Bölünmeler ne kadar iyi olursa, tahminimiz de o kadar iyi olur.

$$H = -\Sigma\, p(x) \log p(x)$$

(Entropi Denklemi, Claude Shannon, 1948.)

Burada, p (x) belirli bir sınıfa ait grubun yüzdesini ve H ise entropiyi belirtmektedir. Karar ağacımızın entropi değerini en aza indirgeyen bölünmeler yapmasını isteriz. En iyi bölünmeyi belirlemek içinde bilgi kazancını kullanırız. Bilgi kazancı aşağıdaki eşitlik ile hesaplanır:

$$Gain(S, D) = H(S) - \sum_{V \in D} \frac{|V|}{|S|} H(V)$$

Bilgi Kazancı

ID3 Algoritması

ID3 (Iterative Dichotomiser 3), karar ağacı oluşturmak için kullanılan bir sınıflandırma algoritmasıdır. ID3 algoritması, veri kümesindeki özellikleri kullanarak bir karar ağacı oluşturur. Bu algoritma, veri kümesini bölerek en homojen alt kümeleri elde etmeye çalışırken özellik seçimi yapar. ID3 algoritması, her bir özniteliğin ayrı ayrı bilgi kazancını hesaplayarak karar ağacı oluşturur. Bu hesaplama, veri kümesinin hangi özniteliğin bölünmesinin en çok bilgi kazancını sağlayacağını belirlemeye yöneliktir.

Algoritmanın çalışma adımları şu şekildedir:

Genel Entropi Hesaplama: Başlangıçta, veri kümesindeki farklı sınıfların dağılımına göre genel entropi hesaplanır. Entropi, belirli bir durumun düzensizliğini ölçer. Eğer bir düğümde sınıflar homojen ise (örneğin, tüm veri noktaları aynı sınıfa aitse), entropi değeri düşüktür.

Her Öznitelik İçin Entropi Hesaplama: Veri kümesindeki her öznitelik için ayrı ayrı entropi hesaplanır. Bu hesaplama, özniteliğin bölünmesinin sınıflar arasındaki homojenliği ne kadar artıracağını gösterir. Öznitelikler, veri kümesini en iyi şekilde bölen öznitelik olarak seçilir.

Bilgi Kazancı Hesaplama: Genel entropi değerinden her öznitelik için hesaplanan entropi değerleri çıkarılarak bilgi kazancı (information gain) elde edilir. Bilgi kazancı, belirli bir özniteliğin seçilmesiyle elde edilen entropi azalışını ifade eder.

En Yüksek Bilgi Kazancına Sahip Özniteliğin Seçimi: Bilgi kazancı en yüksek olan öznitelik, ağaç yapısının o adımda bölünmesi için seçilir. Bu öznitelik, veri kümesini en iyi şekilde bölen öznitelik olarak kabul edilir ve ağaç bu özniteliğe göre dallanır.

Dallanma ve Tekrarlama: Seçilen öznitelik üzerinden veri kümesi bölünerek yeni düğümler oluşturulur. Bu adımlar, her bir yeni düğüm için tekrarlanarak karar ağacının oluşturulmasını sağlar.

ID3 algoritması, özellik seçiminde bilgi kazancını kullanarak veri kümesini en iyi şekilde bölen özniteliği seçer. Bu şekilde, sınıflandırma için en uygun karar ağacını oluşturmaya çalışır.

C4.5 Algoritması

C4.5, sınıflandırma problemleri için kullanılan bir karar ağacı oluşturma algoritmasıdır ve ID3 algoritmasının geliştirilmiş bir versiyonudur. Veri kümesindeki özelliklerin sınıflandırma için en uygun karar ağacını oluşturmak amacıyla kullanılır.

ID3'den farklı olarak, C4.5 algoritması şu iyileştirmeleri içerir:

Sayısal Değerlerin İşlenmesi: ID3 algoritması sadece kategorik (dallanma yapacak özellikler) değerleri işleyebilirken, C4.5 sayısal değerleri de işleyebilir. Bu sayede, sayısal verilerin direkt kullanılmasını sağlar.
Eksik Veri Problemi İle Başa Çıkma: C4.5 algoritması eksik verilerle başa çıkabilir. Veri kümesinde eksik değerler varsa, C4.5 bununla baş etmek için özel stratejiler kullanılabilir.
Bilgi Kazancı Oranı: C4.5'te, öznitelik seçimi sırasında bilgi kazancı yerine bilgi kazancı oranı (gain ratio) kullanılır. Bilgi kazancı oranı, özniteliğin bölünme yeteneğini ölçerken aynı zamanda özniteliğin çok fazla dallanmasını engelleyerek ağacın daha dengeli olmasını sağlar.
C4.5 algoritması, aşağıdaki adımları izler:
Veri Kümesinin İşlenmesi: Başlangıçta, sınıflandırma için kullanılacak veri kümesi üzerinde işlemler yapılır. Eğer sayısal değerler varsa, sayısal değerleri kategorik değerlere dönüştürür.
Kök Düğüm Seçimi: En iyi özniteliği seçmek için özniteliklerin bilgi kazancı oranları hesaplanır. Bilgi kazancı oranı en yüksek olan öznitelik, kök düğüm olarak seçilir.
Düğüm Oluşturma ve Dallanma: Kök düğüm seçildikten sonra, veri kümesi bu özniteliğe göre bölünür. Her bir alt düğüm için bu adımlar tekrarlanır.
Ağaç Oluşturma ve Dallanma: Bu işlem, her bir düğüm için tekrar edilerek karar ağacı oluşturulur.
C4.5, veri setinin yapısını dikkate alarak en uygun özniteliği seçerek ve bu özniteliklere göre ağacı dallandırarak, veri kümesini en iyi şekilde sınıflandırmaya çalışır. Bu iyileştirmeler sayesinde, C4.5 daha geniş bir veri yelpazesine uyum sağlayabilir ve daha dengeli karar ağaçları oluşturabilir.
Twoing Algoritması
Twoing, karar ağaçlarındaki öznitelik seçimi için kullanılan bir yöntemdir. Bu yöntem, veri kümesini iki alt küme arasında bölme stratejisini ölçer ve öznitelikler arasında karşılaştırma yaparak özellik seçiminde kullanılır.Twoing, veri kümesini iki alt küme arasında böldüğünde her bir sınıf için iki alt kümedeki dağılımın homojenliğini ölçer. Alt kümelerdeki sınıf dağılımının homojen olup olmadığını belirleyerek öznitelik seçiminde kullanışlı bir kriter sağlar.
Twoing algoritmasının adımları şu şekildedir:
Veri Kümesinin Bölünmesi: Öznitelikleri değerlendirerek veri kümesini iki alt küme arasında böler. Örneğin, bir öznitelik için bir eşik değer seçerek veri noktalarını bu eşik değere göre iki gruba ayırabilir.
İkili Bölünmenin Hesaplanması: Her bir sınıf için iki alt kümedeki dağılımın homojenliğini ölçer. Örneğin, veri kümesindeki her sınıf için iki alt kümedeki gözlemlerin yüzdesini hesaplayarak sınıfın homojenliğini ölçer.
Twoing İndeksinin Hesaplanması: İkili bölünmenin her sınıf için ölçüsünü alarak, bunların farkını alır ve Twoing indeksini hesaplar. Twoing indeksi, öznitelikler arasında seçim yaparken hangi özniteliğin daha iyi bir bölünme sağladığını belirleyen bir metrik olarak kullanılır.
En İyi Öznitelik Seçimi: Twoing indeksi en yüksek olan öznitelik, karar ağacı oluşturulurken seçilir. Bu öznitelik, veri kümesini en iyi şekilde bölen öznitelik olarak kabul edilir.

Gini Algoritması

Gini indeksi, bir özniteliği iki parçaya bölmek için kullanılan bir kriterdir. Veri kümesini iki alt küme arasında bölerken, her bölümleme adımında Gini impurity (karmaşıklık) değeri hesaplanır.

Veri kümesi bir özniteliğe göre iki alt kümeye ayrıldığında, her alt kümedeki sınıfların homojenliği ölçülür. Her alt küme için Gini impurity hesaplanır ve bu değerler kullanılarak bölümlemenin ne kadar homojen olduğu belirlenmeye çalışılır.

Bir öznitelik için bölümleme adımları şu şekildedir:

Öznitelik Değerlerine Göre Bölünme: Bir öznitelik değeri seçilir ve veri kümesi bu değere göre iki alt kümeye ayrılır.

Gini Impurity Hesaplama: Her bir alt kümedeki sınıfların homojenliği Gini impurity hesaplanarak ölçülür. Bunun için her alt kümedeki sınıf değerlerinin dağılımı dikkate alınır.

Gini Değerlerinin Karşılaştırılması: Her bir bölünme için hesaplanan Gini impurity değerleri karşılaştırılır. Bu adım, hangi bölünmenin daha homojen alt kümeler oluşturduğunu belirlemek için yapılır.

En Küçük Gini Değerine Göre Seçim: Her bir bölünmenin Gini impurity değerleri arasından en küçük olan seçilir. En küçük Gini değerine sahip bölünme, en homojen alt kümeleri oluşturur ve bu şekilde veri kümesinin en iyi şekilde bölünmesini sağlar.

Bu işlem, bir özniteliği iki parçaya bölerken, her bir bölümleme adımında alt kümelerin homojenliğini ölçmek ve en homojen alt kümeleri seçmek için yapılır. Böylece, karar ağacının oluşturulması için hangi özniteliğin hangi bölünmeyi sağlayacağına karar verilir. En düşük Gini impurity değerine sahip bölünmeler, karar ağacı oluşturulurken tercih edilir.

Uygulama

```
fn = ['pH','Temperature','Taste','Odor','Fat','Turbidity','Colour']
cn = ['0', '1', '2']
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
dt=DecisionTreeClassifier(splitter='best')
dt.fit(X_train,y_train)
dt_preds=dt.predict(X_test)
accuracy = accuracy_score(dt_preds, y_test)
print("The accuracy of the Decision Tree is","{:f}".format(accuracy))
```

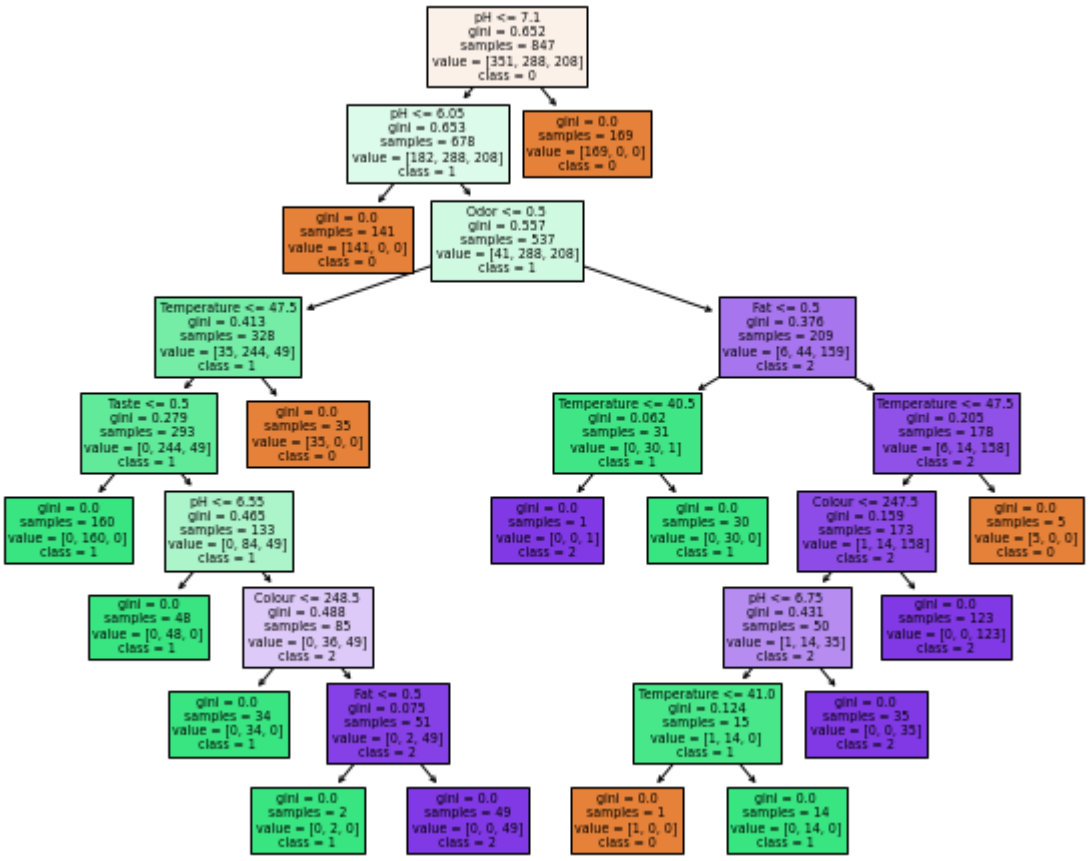
```
The accuracy of the Decision Tree is 0.990566
```

fn,kullanılacak özniteliklerin isimlerini içeren bir listedir. cn, sınıf etiketlerini içeren bir listedir.DecisionTreeClassifier ile bir karar ağacı modeli oluşturulur (splitter='best' ile en iyi bölmeyi seçer) ve fit() fonksiyonu kullanılarak eğitim veri seti (X_train, y_train) üzerinde eğitilir.Oluşturulan model, test veri seti (X_test) üzerinde tahminler yapar (predict() fonksiyonu). Ardından, bu tahminler gerçek test etiketleri (y_test) ile karşılaştırılarak doğruluk (accuracy) değeri accuracy_score() fonksiyonu ile hesaplanır.Doğruluk değeri, modelin doğru tahminlerinin toplam örnek sayısına oranıdır. Yani, bu model test veri seti üzerindeki örneklerin yaklaşık %99.1'ini doğru bir şekilde sınıflandırmıştır.Yüksek bir doğruluk değeri genellikle iyi bir şeydir, ancak tek başına bir modelin performansını

değerlendirmek için yeterli olmayabilir. Özellikle sınıf dengesizliği durumunda, dikkate alınması gereken diğer metrikler de vardır. Hassasiyet, geri çağrı, F1 puanı gibi metrikler, özellikle dengesiz sınıflandırma problemlerinde modelin performansını daha iyi değerlendirebilir.

```
import matplotlib.pyplot as plt
plt.figure(figsize = (10,8))
plot_tree(dt, feature_names = fn, class_names = cn, filled = True);
```

Verilen parametreler şunlardır:
dt: Karar ağacı modeli
feature_names = fn: Öznitelik isimleri
class_names = cn: Sınıf isimleri
filled = True: Karar ağacının dolgu rengi kullanılarak görselleştirilmesi
Bu kod, matplotlib ile belirtilen boyutlarda bir figür oluşturarak plot_tree fonksiyonunu kullanır.



Bu grafik, karar ağacının dallarını ve kararlarını göstererek modelin nasıl kararlar verdiğini anlamanıza yardımcı olur.Karar ağacının kök düğümünde ph <= 7.1 için karar ağacı, veri kümesindeki ph değerini 7.1 değerine göre karşılaştırdı.
gini = 0.652 Bu, bu kararın belirlenmesinde kullanılan kriterin Gini impurity (Gini belirsizliği) skorunu gösterir. Gini impurity, bir düğümdeki farklı sınıfların karışımının ne kadar homojen olduğunu ölçer. Değer ne kadar düşük olursa, o kadar homojen bir düğüm elde edilir.
"samples = [352, 288, 208]": Bu, bu düğüme ait olan sınıflandırma yapısını gösterir. Bu örnekte, sırasıyla [sınıf_1, sınıf_2, sınıf_3] için örnek sayılarını temsil eder.
"class = 0": Bu, bu düğümdeki tahmin edilen sınıfı ifade eder. Bu durumda, bu düğüm sınıf 0 olarak tahmin edilmiştir.
Karar ağacının ilk adımı, veri kümesinin "ph" özelliğini 7.1 değeri ile karşılaştırarak bir bölme yapar. Gini impurity skoru 0.652 olduğu için bu düğüm, belirli bir düzeyde karmaşıklığa sahiptir. Bu düğümdeki örneklerin sınıflandırma yapısı, sınıf 0 için daha ağırlıklıdır ve bu düğümdeki örneklerin çoğunluğu sınıf 0'a aittir.
Karar ağacının bu ilk adımı, veri kümesinin bu belirli noktasında 'ph' özelliğine göre belirli bir sınıflandırma yapmıştır ve sınıf 0'ı tahmin etmiştir. Bu ilk bölme, daha fazla dallanmaya ve daha karmaşık bir ağaç yapısına yol açabilir, ancak bu ağacın temel kararını ve ilk bölmesini temsil eder.

```
# Calculate the confusion matrix
cm = confusion_matrix(y_test, dt.predict(X_test))

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=cn, yticklabels=cn)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Decision Tree Confusion Matrix')
plt.show()
```

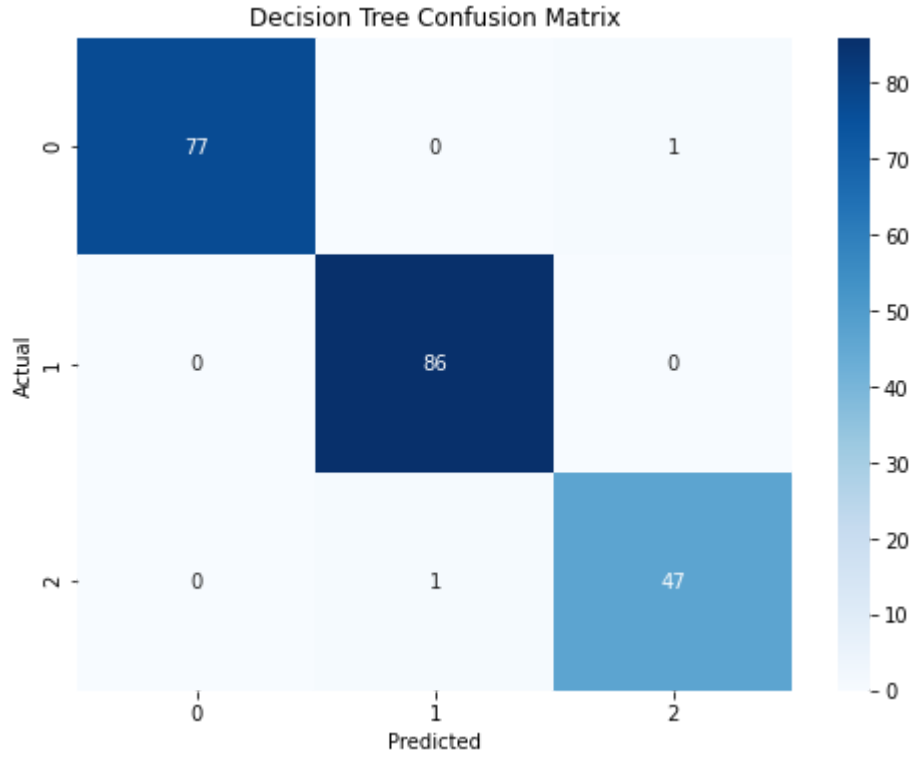
Confusion matrix fonksiyonu, gerçek test etiketleri (y_test) ile modelin test veri seti üzerindeki tahminlerini (dt.predict(X_test)) karşılaştırarak bir karışıklık matrisi oluşturur.sns.heatmap fonksiyonu, seaborn kütüphanesini kullanarak oluşturulan karışıklık matrisini bir ısı haritası şeklinde görselleştirir. annot=True parametresi, her hücredeki değerlerin gösterilmesini sağlar. fmt="d" parametresi, hücrelerdeki değerlerin tamsayı olarak formatlanmasını belirtir. cmap="Blues" parametresi, renk haritasını mavi tonları olarak belirtir.xticklabels ve yticklabels parametreleri, x ve y eksenindeki etiketleri belirler.Matplotlib kütüphanesini kullanarak oluşturulan ısı haritası, ekranda gösterilir.Bu görselleştirme, modelin sınıflandırma performansını daha ayrıntılı olarak incelemek için kullanılır. Karışıklık matrisi, modelin her sınıfı ne kadar doğru veya yanlış tahmin ettiğini gösterir. Görselleştirme, modelin hangi sınıfları karıştırdığını, hangi sınıflarda daha iyi veya daha kötü performans gösterdiğini göstermek için kullanışlıdır.

Confusion matrix, dört farklı terimden oluşur:
True Positive (TP): Gerçek etiketi pozitif olan ve model tarafından doğru bir şekilde pozitif olarak tahmin edilen örnek sayısı.

True Negative (TN): Gerçek etiketi negatif olan ve model tarafından doğru bir şekilde negatif olarak tahmin edilen örnek sayısı.
False Positive (FP): Gerçek etiketi negatif olan ancak model tarafından yanlış bir şekilde pozitif olarak tahmin edilen örnek sayısı (Tip 1 hata olarak da bilinir).
False Negative (FN): Gerçek etiketi pozitif olan ancak model tarafından yanlış bir şekilde negatif olarak tahmin edilen örnek sayısı (Tip 2 hata olarak da bilinir).
Bu terimlerin kombinasyonu, confusion matrixin aşağıdaki şekildeki gibi dört karelik bir matris oluşturur:

$$\begin{bmatrix} \text{True Negative (TN)} & \text{False Positive (FP)} & \text{False Positive (FP)} \\ \text{False Negative (FN)} & \text{True Positive (TP)} & \text{False Negative (FN)} \\ \text{False Positive (FP)} & \text{False Negative (FN)} & \text{True Positive (TP)} \end{bmatrix}$$

Confusion matrix, modelin performansını değerlendirmek için kullanılır ve çeşitli performans metriklerinin hesaplanmasında temel oluşturur. Bu metrikler arasında doğruluk (accuracy), hassasiyet (precision), geri çağrı (recall) ve F1 puanı (F1-score) gibi değerler bulunur.
Doğru sınıflandırmaların ve hataların hangi sınıflar arasında gerçekleştiğini görselleştiren bu matris, modelin hangi sınıflarda daha iyi veya daha kötü performans gösterdiğini anlamak için önemlidir.



Sınıf 0:
77 örnek, sınıf 0'a ait ve doğru bir şekilde sınıflandırılmıştır (True Negative).
0 örnek, sınıf 0'a ait olmasına rağmen başka bir sınıfa yanlışlıkla sınıflandırılmıştır (False Positive).
1 örnek, sınıf 0'a ait olmayan ancak sınıf 0 olarak yanlışlıkla sınıflandırılmıştır (False Negative).

Sınıf 1:
86 örnek, sınıf 1'e ait ve doğru bir şekilde sınıflandırılmıştır (True Negative).
0 örnek, sınıf 1'e ait olmasına rağmen başka bir sınıfa yanlışlıkla sınıflandırılmıştır (False Positive).
0 örnek, sınıf 1'e ait olmayan ancak sınıf 1 olarak yanlışlıkla sınıflandırılmıştır (False Negative).
Sınıf 2:
47 örnek, sınıf 2'ye ait ve doğru bir şekilde sınıflandırılmıştır (True Negative).
0 örnek, sınıf 2'ye ait olmasına rağmen başka bir sınıfa yanlışlıkla sınıflandırılmıştır (False Positive).
1 örnek, sınıf 2'ye ait olmayan ancak sınıf 2 olarak yanlışlıkla sınıflandırılmıştır (False Negative).

RANDOM FOREST

Bir Ensemble learning yöntemi olan Random Forest anlatılmadan önce temelini anlamak amacıyla kendisinin de dahil olduğu topluluk öğrenme tekniklerine değinilecektir.

Ensemble Öğrenme Yöntemleri (Topluluk Öğrenmesi) nedir?

Birden fazla algoritmanın gücünü birlikte kullanılarak daha yüksek başarı elde etmeyi amaçlar. Bu yöntemde farklı doğruluk skorlarına sahip sınıflandırıcıların sonuçları farklı yöntemlerle (oylama, ortalama vb.) birleştirilir.Bu nevi ortak akıl gibi düşünülebilir.Başta sınıflandırma algoritmaları olmak üzere regresyon yöntemlerinde de kullanılabilir.

Ensemble öğrenme yöntemlerinin çeşitleri nelerdir?

Bu yöntemler homojen ve heterojen olarak değerlendirilebilir. Homojen ensemble metodunda tüm modeller aynı makine öğrenmesi algoritması ile eğitilir.Eğer kullanılacak olan algoritma karar ağacı ise bu güzel bir random forest örneğidir. Bagging ve Boosting metodları homojen ensamble metodlarına girer. Heterojen ensemble yöntemleri, farklı öğrenme algoritmalarını veya farklı öğrenme türlerini bir araya getirir. Örneğin, bir karar ağacı, bir destek vektör makinesi ve bir kNN (k-en yakın komşu) algoritması gibi farklı öğrenme türlerini birleştirmek heterojen bir ensemble örneğidir. Burada farklı bakış açılarını birleştirerek daha geniş bir çeşitlilik ve kapsamlı öğrenme sağlanabilir.

Bu modeller aynı train datasını kullanırlar.(Bu ileride açıklanacaktır.)

Weak Learners(Zayıf öğrenen) ve Strong Learners(Güçlü öğrenen) nedir?

İki ensemble yönteminde de tekil algoritmalar zayıf öğrenen olarak tanımlanır., bunlar optimum öğrenici değillerdir. Zayıf öğrenenlerin tek başına performansı düşük olabilir, ancak bir araya getirildiklerinde güçlü bir şekilde işbirliği yaparak daha güçlü bir öğrenen oluşturabilirler. Bagging

veya boosting gibi tekniklerle zayıf öğrenicileri bir araya getirerek güçlü bir öğrenen oluşturulur. Örneğin Random Forest gibi bir algoritma, zayıf tahmincileri (decision tree) bir araya getirerek güçlü bir tahminci oluşturur ve bu sayede yüksek performans elde edilir.

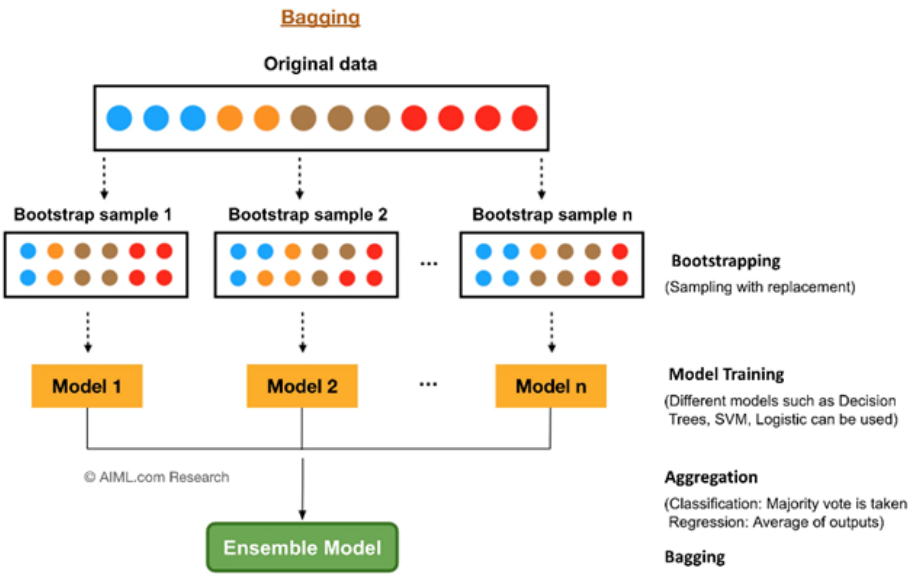
Bootstrap Örneklemesi Nedir?

Bu örneklemede amaç, oluşturulan örneklemlerin istatistiklerinin popülasyon parametresini iyi bir şekilde tahmin edebilecek halde olmasıdır.Örneklemler, veriyi iyi bir şekilde temsil etmelidir. Her bir örnek veri seti, asıl veri setinden örneklenirken, veri setinin özelliklerini korur ve bu şekilde yapılan analizler daha güvenilir sonuçlar sağlayabilir.Örneklemdeki veriler rastgeledir fakat örneklemlerin istatistikleri korunur. Veri setinin sınırlı olduğu durumlarda kurtarıcı bir yöntemdir.Her bir örneklem aslında popülasyonun bir yaklaşığıdır.

Homojen Ensemble öğrenme yöntemlerinden : Bagging Nedir?

Bagging yönteminde tüm modeller paralel olarak inşa edilir,ve birbirlerinden bağımsızlardır. Bu.yöntemde boostrap örneklemelerindeki tüm gözlemler eşit olarak ele alınacaktır. Gözlemlerde bir ağırlıklandırma olmayacaktır.Modellerin çalışması tamamlandığında çıkan sonuçlar arasında en çok çıktısı verilen sınıf tahmin olarak alınır.Bu bir sınıflandırma algoritması ise en çok tahmin sonucuna ulaşılan sınıf çıktı olarak alınır, regresyon algoritması ise çıkan sonuçların ortalaması alınır. Bu ortalamaların ağırlıkları eşittir. Bagging yöntemine “boostrap aggregating” da denir

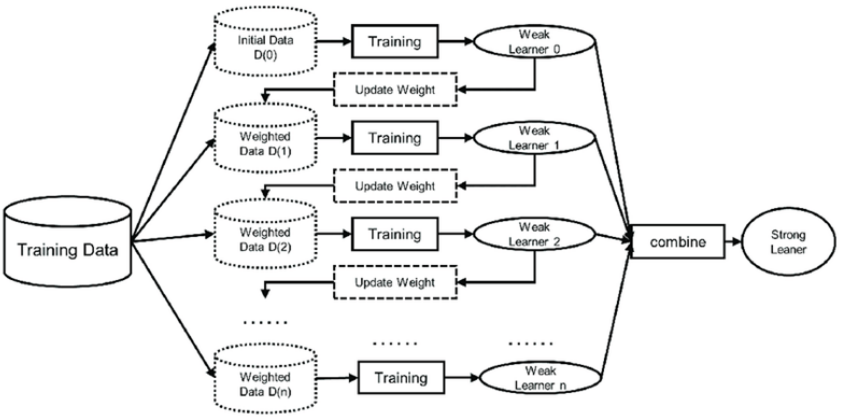
.



Kullanım avantajı olarak bagging, Çoğunluk oylama yaklaşımını kullanarak tüm model çıktılarının ortalaması alındığı için overfitting ile başa çıkma konusunda iyidir. Dezavantaj olarak da Regresyon modelleri için tahmin edilen değer optimize edilemeyeceği için modellerden herhangi biri daha fazla sapsa da , çıkış değeri tüm modellerin ortalaması olacağından sonuca dahil edilecektir.

Homojen Ensemble yöntemlerinden : Boosting Nedir?

Bagging yönteminin farklı bir versiyonudur. Fark; öğrenme sonuçlarının bir sonraki öğrenici için kullanılıyor olmasıdır. Eğitim için ayrılan veri setinden bir temel öğrenici için rastgele seçim yapılır. Öğrenme gerçekleşir, model test edilir. Sonuçlardan yanlış sınıflandırılan örnekler belirlenir. Bunlar bir sonraki öğrenici için örnek seçiminde önceliklendirilir (seçilme olasılıkları arttırılır). Her seferinde bu bilgi güncellenir. Bagging yönteminde her bir örneğin seçilme şansı eşitken burada ağırlıklandırma vardır.

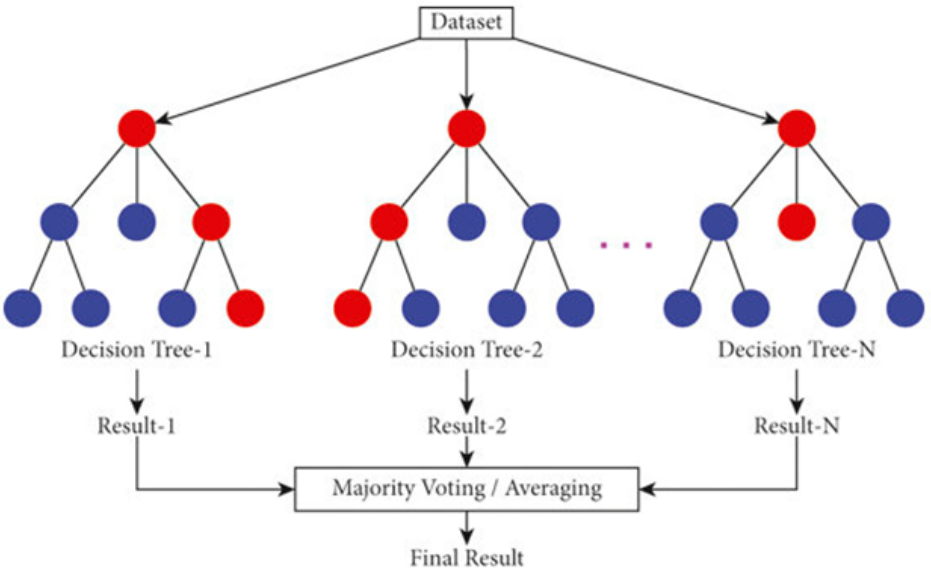


Artı yönleri :Diğer yöntemlere göre daha yaygındır, hızlı çalışır az bellek kullanır.Her bir model, bir önceki modelin hatasını düzeltmeye çalıştığı için bias'ı azaltır.

Eksi Yönleri : overfitting ile başa çıkamaz.Hyper parametre işlemi yaparken özenli çalışma gerektirir.

Random Forest

2001 Yılında Leo Breiman tarafından geliştirilmiştir. RF algoritması Begging yöntemi ve Random Subspace yöntemlerinin birleşimidir. Bir toplu öğrenme yöntemi olan Random Forest, sınıflandırma işlemi esnasında birden fazla karar ağacı üreterek sınıflandırma başarısını yükseltmeyi hedefler. Bireysel olarak oluşturulan karar ağaçları bir araya gelerek karar ormanı oluşturur.



Bir ensemble yöntem kullanıldığında sonuç tek bir sınıflandırıcıdan daha iyi olmalıdır. aksi halde ensemble yöntem kullanmanın anlamı kalmaz. Çünkü tek bir sınıflandırıcıya göre toplu öğrenme yönteminin hesaplama maliyeti daha yüksektir.

Algoritma çalışırken nasıl bir yol izliyor?

1)İlk olarak orijinal veri kümesinden alt kümeler oluşturulur.(bootstrap)

2)Karar ağacındaki her düğümde en iyi bölünmeye karar vermek için rastgele bir özellik kümesi dikkate alınır.

3)Her bir alt kümeye bir karar ağacı modeli uygulanır.

4)Nihai tahmin tüm karar ağaçlarından tahmin edilen değerlerin ortalaması olarak hesaplanır.(Regresyon modelinde ortalama, sınıflandırma çalışmalarında mod değeri alınır.)

Random Forest Algoritmasının tipik bir Begging algoritmasından temel farkı nedir?

Toplu öğrenme yöntemlerinden Bagging'in geliştirilmiş bir hali olan Random forest, Karar ağacının her bir düğümde en iyi bölünmeye karar vermeye yarayan bir dizi özelliği (feature) rastgele seçmektedir. Böylece Bagging algoritması, en iyi bölünmeye karar vermek için tüm özellikleri alırken Random Forest'da oluşturulan ağaçlar en iyi bölünmeye karar vermek için her düğümdeki özelliklerin rastgele bir alt kümesini kullanır.Bu sayede daha iyi çeşitlilik sağlar ve tekrarlanabilirlik konusunda daha iyidir. Bagging yöntemlerini anlatırken bahsettiğimiz gibi Random Forest algoritması da hem sınıflandırma hem de regresyon yöntemleri için kullanılabilir.

Random Forest'ın bazı kullanım alanları

Finansal Tahminler, Tıbbi Teşhisler, Pazarlama ve Müşteri Segmentasyonu, Görüntü ve Ses Tanıma, Biyoçeşitlilik ve Ekoloji Analizleri, Anomalileri Tespit Etme

Avantaj ve Dezavantajlar

-Aynı Random Forest algoritması hem bir regresyon probleminde hem de sınıflandırma probleminde kullanılabilir.

-Sınıflandırma problemlerinde overfitting(aşırı öğrenme) durumuyla iyi bir şekilde başa çıkar.

-Feature Engineering(Özellik Mühendisliği) çalışmalarında kullanılabilir.

Uygulama

RandomForestClassifier kodunda önemli 2 parametre bulunmaktadır. "n-estimators" parametresi ile bu algoritmanın kaç adet ağaç ile oluşturulacağı belirtilir.

"random_state" : Zorunlu olmayan bu parametre, fonksiyon tekrarlandığı zaman aynı değerleri mi yoksa farklı değerleri mi rastgele üretmesini istediğimizi belirtmek için kullanılır. random_state = 42 değerini verdiğiniz her durumda tablodan aynı rastgele değerler çekilecektir.

```
model = RandomForestClassifier(n_estimators=100,random_state= 42)
model.fit(X_train,y_train)
model_predictions = model.predict(X_test)
accuracy = accuracy_score(y_test,model_predictions) * (100)
print("Accuracy:",accuracy)
print("Kullanılan Ağaç Sayısı:",model.n_estimators)
```

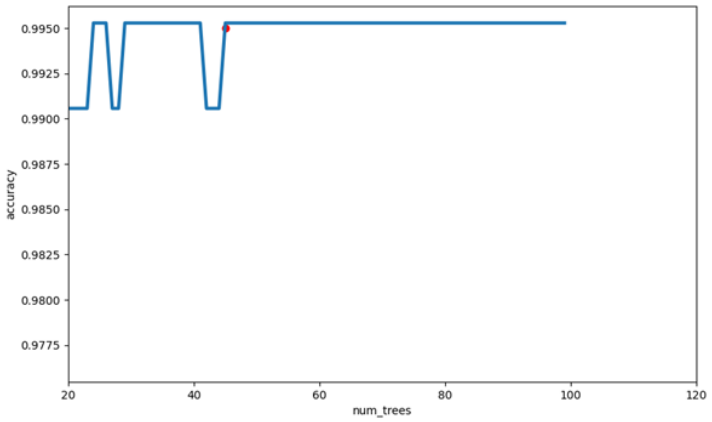
```
Accuracy: 99.52830188679245
Kullanılan Ağaç Sayısı: 100
```

Burada doğruluk değeri çok yüksek çıkmıştır, diğer sınıflandırma yöntemlerine göre daha yüksek olması beklendiği gibidir.Ayrılan %20'lik test datasını neredeyse birebir tahmin edebilmektedir.

Optimal "n_estimators" değeri nasıl seçilir?

Tahminci yani "ağaç" sayısı belirlenirken belirli tahminci sayısı – doğruluk oranı eşleştirmeleri yapılıp karar verilebilir.Bir tahminci-doğruluk oranı grafiği bu konuda pratiklik kazandırır.

```
predictions = []
for tree in rf.estimators_:
    predictions.append(tree.predict_proba(X_test)[None, :])
predictions = np.vstack(predictions)
cum_mean = np.cumsum(predictions, axis=0)/np.arange(1, predictions.shape[0] + 1)[:, None, None]
scores = []
for pred in cum_mean:
    scores.append(accuracy_score(y_test, np.argmax(pred, axis=1)))
    plt.figure(figsize=(10, 6))
plt.plot(scores, linewidth=3)
plt.xlabel('num_trees')
plt.ylabel('accuracy')
plt.xlim(20,120)
plt.scatter(45, 0.995, color="red")
plt.show()
```



Grafikteki kırmızı nokta doğruluk değerinin 0.995 olmadan hemen önceki tahminci sayısına karşılık gelen kısmı gösterir. Görüldüğü gibi ağaç sayısı 46'dan büyük olduğunda tahminci sayısı maksimum doğruluk değerine göre optimal olacaktır.Tahminci sayısını daha çok artırmaya gerek olmayabilir.

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
kf = KFold(n_splits=20, shuffle=True, random_state=42)
cv_results = cross_val_score(model, X, y, cv=kf, scoring='accuracy')
round(np.mean(cv_results),3)
```

```
0.998
```

Ayrıca K-Fold Cross Validation işlemi yapıldığında da doğruluk oranının çok yüksek olduğu görülmektedir.

KAYNAKLAR

https://medium.com/machine-learning-t%C3%BCrkiye/knn-k-en-yak%C4%B1n-kom%C5%9Fu-7a037f056116
https://arslanev.medium.com/makine-%C3%B6%C4%9Frenmesi-knn-k-nearest-neighbors-algoritmas%C4%B1-bdfb688d7c5f
https://ab.org.tr/ab16/bildiri/102.pdf
Makine Öğrenmesi Karar Ağacı (Decision Tree) Nedir? - Bulutistan Blog
Makine Öğrenimi Bölüm-5 (Karar Ağaçları) | by E. Kaan Ulgen | Medium
https://urmiparekh.medium.com/random-forest-algorithm-abfee84bb48c
https://en.wikipedia.org/wiki/Random_forest
https://miracozturk.com/python-ile-siniflandirma-analizleri-rastgele-orman-random-forest-algoritmasi/
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
https://www.veribilimiokulu.com/random-forest-regresyon-python-ornek-uygulamasi/
https://www.ibm.com/topics/random-forest
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
https://medium.com/analytics-vidhya/what-is-a-confusion-matrix-d1c0f8feda5
Milk Quality Prediction (kaggle.com)