Sınıflandırma Problemleri:

Sınıflandırma, verilen veri noktalarının sınıfını tahmin etme işlemidir.Sınıflandırma, nesnelerin ve fikirlerin önceden belirlenmiş kategoriler halinde tanınması, anlaşılması ve gruplandırılması süreci olarak tanımlanır, Buna “alt popülasyonlar” denir.

En çok kullanılan 5 sınıflandırma algoritması,

- Random Forest
- Lojistik Regresyon
- Naive Bayes
- K-En Yakın Komşu
- Karar Ağaçları
- Destek Vektör Makineleri

Random Forest:

Random Forest, çeşitli ağaçlar oluşturarak ve bu ağaçların sonuçlarını bir araya getirerek tahminler yapmak için kullanılan bir makine öğrenmesi algoritmasıdır. Her bir ağaç, belirli bir özellik alt kümesini kullanarak eğitilir ve birbirinden bağımsızdır. Bu nedenle, Random Forest, hem değişkenler arasındaki etkileşimleri hem de veri setindeki gürültüyü azaltarak, yüksek kaliteli tahminler yapabilir.

Random Forest algoritması, karar ağacı algoritmasından türetilmiştir. Ancak, Random Forest algoritması, tek bir ağaçtan ziyade birden çok ağaç kullanır. Her bir ağaç, bağımsız bir şekilde özellikleri analiz ederek ve bağımlı değişkenlerin özelliklerini en iyi şekilde tahmin etmek için kullanılır. Random Forest, bu ağaçların son sonuçlarını birleştirerek bir tahmin yapar. Bu tahmin, sınıflandırma veya regresyon gibi belirli bir probleme bağlı olarak değişebilir.

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# iris veri setini yükle
iris = load_iris()

# özellikleri ve hedef sınıfı ayır
X = iris.data
y = iris.target

# verileri eğitim ve test kümeleri olarak ayır
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Random Forest modelini oluştur
random_forest = RandomForestClassifier(n_estimators=100)

# Modeli eğit
random_forest.fit(X_train, y_train)

# Test seti üzerinde modelin doğruluğunu ölç
y_pred = random_forest.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Lojistik Regresyon:

Lojistik Regresyon, bir Doğrusal Sınıflandırıcı'dır. Bir sınıfı belirleyen bir veya daha fazla bağımsız değişken bulunan bir veri kümesini analiz

etmek için kullanılan istatistiksel bir yöntemdir. Sonuç, ikili bir değişkenle ölçülür (yalnızca iki olası sonuç vardır). O örnek, aradığımız sınıfa aittir veya değildir(Diğer sınıflardan birine aittir).

Örneğin, ev fiyatı tahminini ele alalım. Elimizde kimi noktasal(örneğin evin oda sayısı), kimi sürekli olan(örneğin evin metrekaresi) bir özellik sınıfı mevcuttu. Bu özelliklere göre, yine bir sürekli değişken olan *ev fiyatı* özelliğini tahmin etmeye çalışıyorduk.

Lojistik Regresyon uygulamalarında da, özellik seti yine sürekli değerler barındırır. Ancak hedefimiz bir sürekli değişken yerine
bir sınıf belirlemektir. Bu sınıfı belirlerken de bir eşik değer kullanırız. Örneğin evin fiyatı 50.000 doların altında ise ucuz(y=0 ile temsil edebiliriz) eğer 50.000 doların üzerinde ise pahalı (y=1 ile temsil edebiliriz).

Navie Bayes:

Bayes Teoremi:

$$P(A|B) = \frac{P(B|A)\ P(A)}{P(B)}$$

P(A|B) = B'nin doğru olduğu bilindiğinde A'nın olma olasılığı

P(B|A) = A'nın doğru olduğu bilindiğinde B'nin olma olasılığı

P(A) = A’nın olma olasılığı

P(B) = B’nin olma olasılığı

Naive Bayes 18.yy’da Thomas Bayes’in Bayes Teoremi temel alarak geliştirilimiştir. Hemen bir örnekle açıklayalım.

Aşağıda hava durumu(X) ve ona karşılık gelen kategorik oyunu oynama durumları(y) yer alıyor.

Öncelikle verilerimizi frekans tablosuna çeviriyoruz ve olasılıkları hesaplıyoruz. Sonrasında teoremi uygulamaya başlıyoruz.

Hava	Oynama D.
Güneşli	Hayır
Bulutlu	Evet
Yağmurlu	Evet
Güneşli	Evet
Güneşli	Evet
Bulutlu	Evet
Yağmurlu	Hayır
Yağmurlu	Hayır
Güneşli	Evet
Yağmurlu	Evet
Güneşli	Hayır
Bulutlu	Evet
Bulutlu	Evet
Yağmurlu	Hayır

Sıklık Tablosu		
Hava	**Hayır**	**Evet**
Bulutlu	**0**	**4**
Yağmurlu	**3**	**2**
Güneşli	**2**	**3**
Toplam	**5**	**9**

Olasılık Tablosu				
Hava	**Hayır**	**Evet**		
Bulutlu	0	4	4 / 14	0.29
Yağmurlu	3	2	5 / 14	0.36
Güneşli	2	3	5 / 14	0.36
Hepsi	5	9		
	5 / 14	9 / 14		
	0.36	0.64		

Örneğin sorumuz şu olsun:

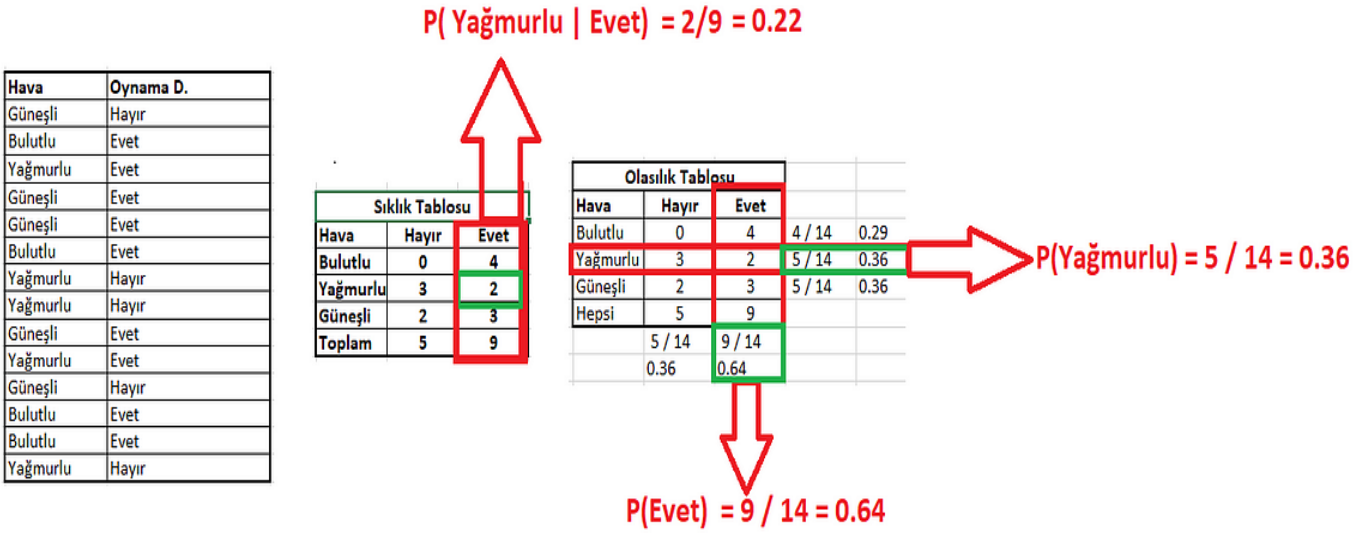
Hava yağmurlu olduğunda oyun oynar mıyım?

P(Evet | Yağmurlu) = P(Yağmurlu | Evet) * P(Evet) / P (Yağmurlu)

P(Yağmurlu | Evet) = 2 / 9 = 0.22

P (Evet) = 9/14 = 0.64

P(Yağmurlu)=5.14 = 0.36



Hava	Oynama D.
Güneşli	Hayır
Bulutlu	Evet
Yağmurlu	Evet
Güneşli	Evet
Güneşli	Evet
Bulutlu	Evet
Yağmurlu	Hayır
Yağmurlu	Hayır
Güneşli	Evet
Yağmurlu	Evet
Güneşli	Hayır
Bulutlu	Evet
Bulutlu	Evet
Yağmurlu	Hayır

Sıklık Tablosu

Hava	Hayır	Evet
Bulutlu	0	4
Yağmurlu	3	2
Güneşli	2	3
Toplam	5	9

Olasılık Tablosu

Hava	Hayır	Evet		
Bulutlu	0	4	4 / 14	0.29
Yağmurlu	3	2	5 / 14	0.36
Güneşli	2	3	5 / 14	0.36
Hepsi	5	9		
	5 / 14	9 / 14		
	0.36	0.64		

Şimdi, P(Evet | Yağmurlu) = 0.22 * 0.64 / 0.36 = 0.39 olarak çıktı.

Burada tek sütunlu bir özellik(Hava durumu) kullandık. Eğer birden fazla sütunumuz olsaydı, bu işlemlerin aynısını her sütun için yapacaktık. Yani Naive Bayes'te her özellik(X1,X2,…Xn) birbirinden bağımsız olarak değerlendirilecekti.

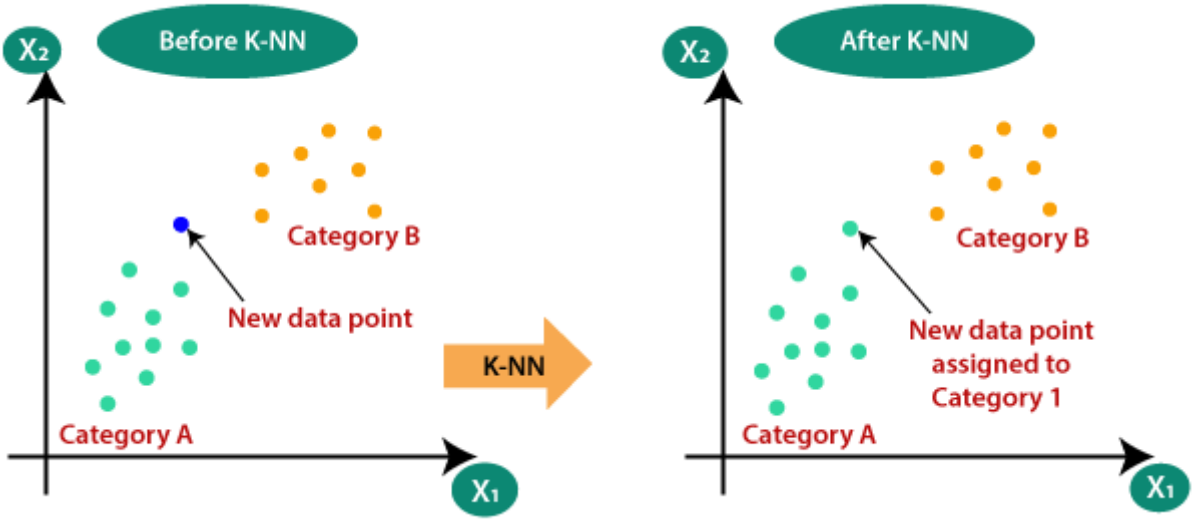
$$P(y|x_1,\ldots,x_n) = \frac{P(x_1|y)P(x_2|y)\ldots P(x_n|y)P(y)}{P(x_1)P(x_2)\ldots P(x_n)}$$

K-En Yakın Komşu:

K-NN (K-Nearest Neighbors) algoritması, model oluşturmaksızın; sınıflandırma ve regresyon problemlerinin her ikisi üzerinde de çalışabilen, denetimli makine öğrenmesi algoritmasıdır. Veriler üzerinde

basit denebilecek bir çıkarımla tahmin yapmaktır. Bu çıkarıma göre; etiket değeri tahmin edilecek bir veri için, n-boyutlu bir uzayda, kendisine en yakın *k* komşunun etiket değerleri referans alınmalıdır.

Örnek olması açısından; e-postalar için bir spam filtresi geliştiriliyor olsun. Spam e-postaları, yalnızca başlıklarını kontrol ederek ayırt edebileceğimizi varsayalım. Yeterli sayıda e-posta, spam ve spam olmayan şeklinde elle etiketlenerek basit bir veri seti oluşturulsun. Yapmamız gereken: Gelen her yeni e-postayı, veriseti üzerindeki tüm e-postalarla karşılaştırmak ve hangilerine en çok benzediğini bulmak. Gelen e-postaya komşu olan, yani ona en çok benzeyenlerden *k* kadarının çoğunluğu ne olarak etiketlenmişse (spam veya spam olmayan), gelen e-posta için de aynı etiket kabul edilmelidir. Bu şekilde her yeni e-postanın spam olup olmadığı belirlenir.



K-NN’nin uygulanması

Algoritmanın adımlarını özetlersek:

1. *k* değerinin belirlenmesi,

2. Veriseti üzerindeki komşuların belirlenmesi,

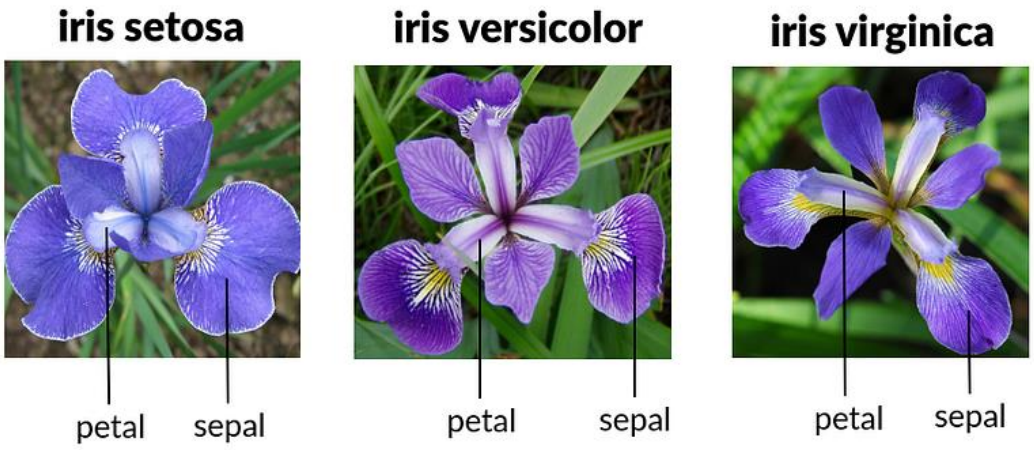
3. En yakın *k* komşunun etiket değerlerinin belirlenmesi,

4. *k* komşu arasından çoğunluğun etiketine göre, yeni verinin etiketinin belirlenmesi.

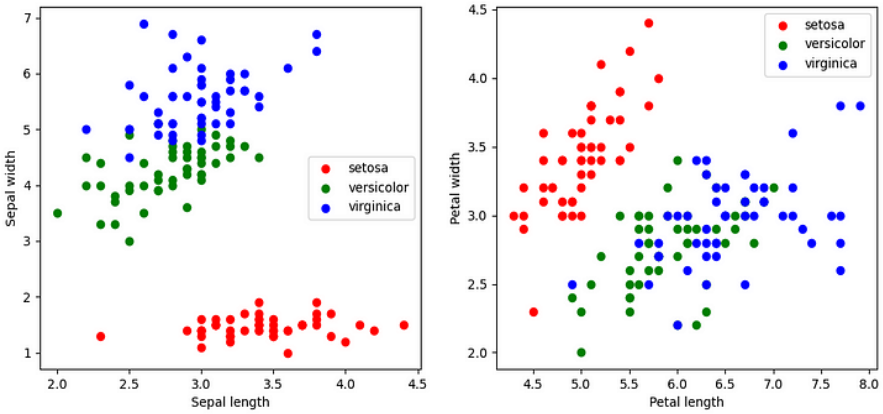
k sayısı, genellikle üzerinde çalışılan probleme ve verisetine göre, kendine özgü olarak belirlenmektedir.

Uygulama:

Algoritmayı üzerinde çalıştırmak için Iris verisetini kullanabiliriz. Makine öğrenmesi alanındaki en yaygın verisetlerinden birisi olan Iris: *Setosa*, *Versicolor* ve *Virginica* türlerindeki 3 farklı Iris çiçeğinden 50'şer örnek içermektedir. Örnekler, irislerin *sepal* ve *petal* yapraklarının genişlik ve uzunluklarından oluşan toplam 4 özelliğe (*feature*) sahiptir. Amaç; veriseti üzerinde test edilmek üzeren ayrılan ve etiket değeri gizli tutulan örneklerin, sepal ve petal yapraklarına göre hangi iris türüne ait olduğunun belirlenmesidir.



Iris çiçeklerinin setosa, versicolor ve virginica türleri Verisetini aşağıdaki gibi görselleştirmek mümkün:



Iris örneklerinin, sepal ve petal özelliklerine göre dağılımı.

K-NN Algoritmasını, basit bir sınıf yapısıyla oluşturabiliriz:

```python
from iris import load_iris, visualize_dataset, visualize_accuracy
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error, confusion_matrix
import math

class KNN:
    def __init__(self, k_value):
        self.k = k_value

    def dist(self, row0, row1): # euclid
        total = 0
        for i, j in zip(row0, row1):
            total += math.pow(i-j, 2)

        return math.sqrt(total)

    def get_nearest_neighbors(self, row_to_search):

        distances, neighbors = [], []
        for i, x_row in enumerate(self.x_train):
            d = self.dist(row_to_search, x_row)
            distances.append([d, i]) # dist, index

        distances.sort(key = lambda x: x[0])

        for i in range(self.k):
            neighbors.append(distances[i])

        return neighbors

    def predict(self, X_test, X_train, Y_train):

         for x_row in X_test:

             neighbors = self.get_nearest_neighbors(x_row)
             targets = []
             for n in neighbors:
                 ind = n[1]
                 targets.append(self.y_train[ind])

             y_predict.append(max(targets, key = targets.count))

         return y_predict

iris = load_iris()
# visualize_dataset()

x_train, x_test, y_train, y_test = train_test_split(
    iris['data'],
    iris['target'],
    test_size = 0.33,
    random_state = 0)

knn = KNN(k_value=5)

y_pred = knn.predict(x_test, x_train, y_train)

print(r2_score(y_pred, y_test))
print(mean_squared_error(y_pred, y_test))
print(confusion_matrix(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)

visualize_accuracy(cm)
```

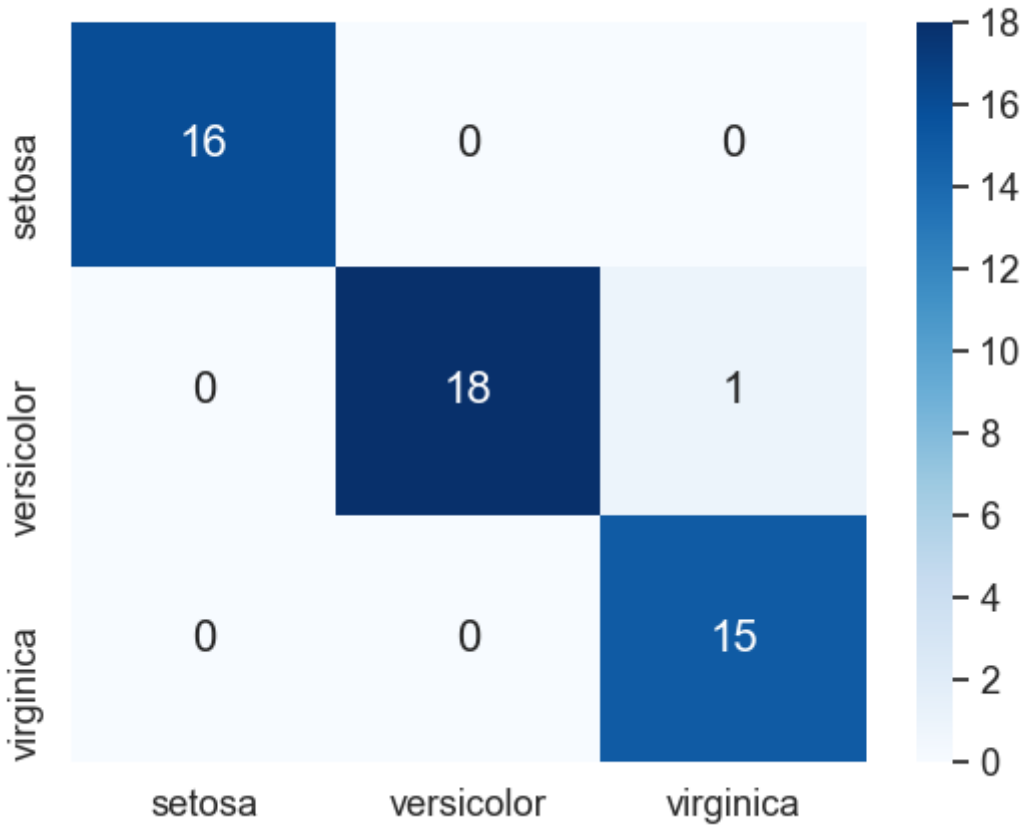
- *dist*: Veriseti üzerindeki iki satırın/örneğin özellik değerlerinin uzaklığını hesaplamaktadır. Bunun için Öklid uzaklık formülü kullanılmaktadır.

- *get_nearest_neighbors:* Parametre olarak aldığı örneği, veriseti üzerindeki diğer örneklerle karşılaştırarak, herbirine olan uzaklığı hesaplamaktadır. Uzaklıkların tutulduğu *distances* listesi, nihayetinde en yakından uzağa doğru sıralanır ve baştan *k* tanesi *neighbors* listesine eklenerek geri döndürülür.

- *predict:* Kullanıcıdan test ve eğitim örneklerini almaktadır. Bahsettiğimiz gibi K-NN algoritması, model oluşturmaksızın çalışmaktadır. Dolayısıyla buradaki eğitim örnekleri, doğrudan test örnekleriyle karşılaştırılarak kullanılmaktadır. Her bir test örneği için en yakın *k* komşu hesaplanır ve bunların etiket değerleri *targets* listesine eklenir. Listede en fazla bulunan etiket değeri de, test örneğinin etiket değeri olarak atanır ve bu şekilde tahmin edilen tüm örnekler, *y_predict* listesiyle geri döndürülür.

Uygulamayı çalıştırmak için:

Verisetinden 50 veriyi (%33) test örneği olarak ayırdık. Tahmin edilen örnekler için bir hata matrisi çizdirelim:



Confusion matrix (Hata Matrisi)

Hata matrisine inceleyerek; setosa ve versicolor türlerinden toplam 34 örneği doğru şekilde sınıflandırırken, aslında virginica türünde olan 16 örnekten 1'inin versicolor olarak yanlış sınıflandırıldığını söyleyebiliriz.

Karar Ağaçları:

En basit tanım ile, olası sonuç için oluşturulabilecek karar yollarının oluşturulmasıdır. Elimizdeki veri setindeki bir feature (kolon) seçip onun diğer alanlar üzerindeki dallanmalarını ilişkilendirip bir karar ağacı çıkartırız.

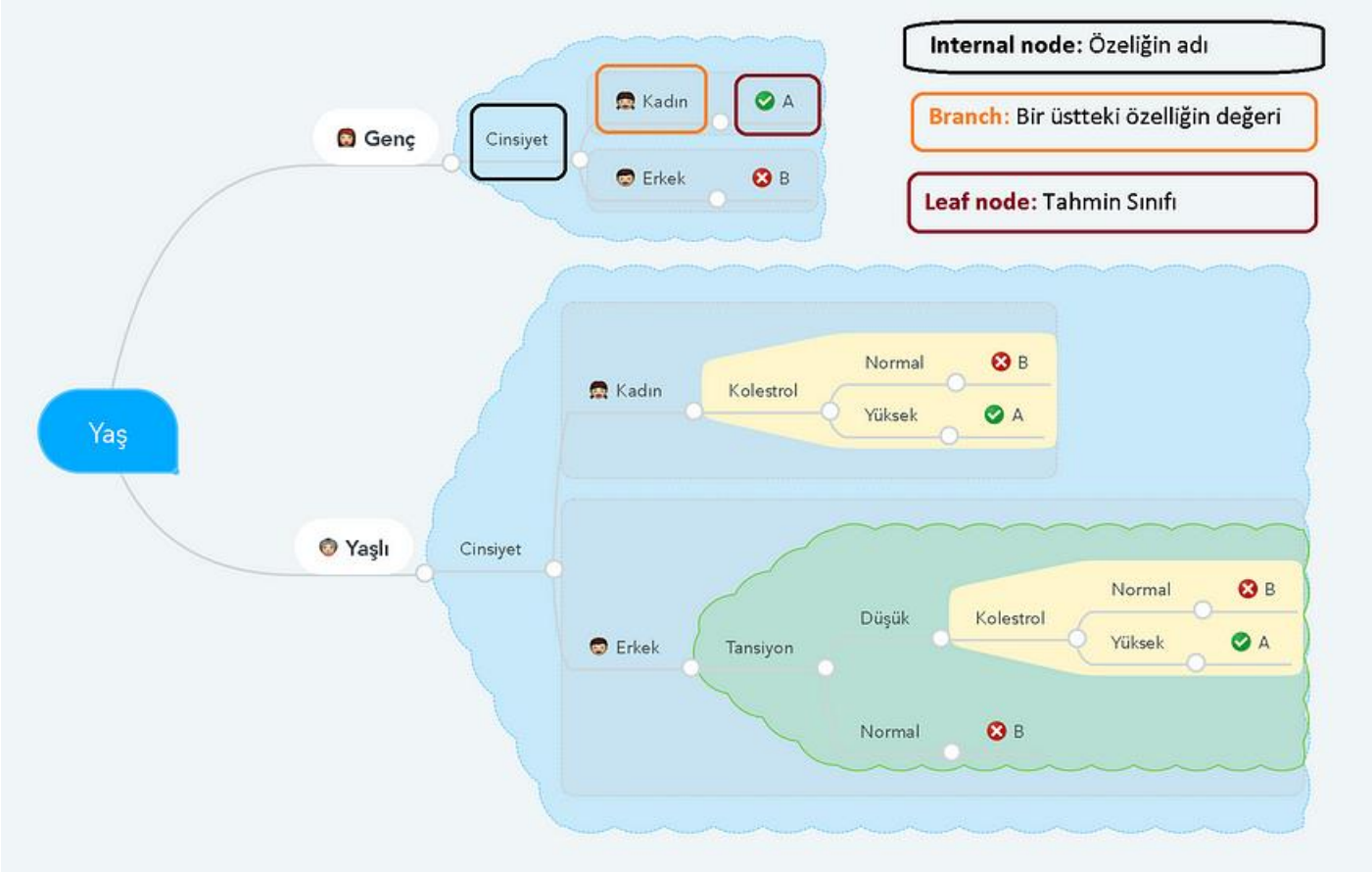
Örnek olması açısından aşağıdaki gibi veri setimiz olduğunu varsayalım. Hastaların kişisel özellikleri ve kullandıkları ilacın ne olduğunun gösteren

bu veri üzerinden yorum yaparak H15 olarak kodlanmış hastaya hangi ilacı vermemiz gerektiğini bulabilirmiyiz. Bu problem klasik bir karar ağacı problemi olarak ele alınabilir.

Hasta	Yaş	Cinsiyet	Tansiyon	Kolestrol	BKE	İlaç
H1	Genç	K	Yüksek	Normal	Şişman	A
H2	Genç	K	Yüksek	Yüksek	Aşırı Şişman	A
H3	Orta Yaşlı	K	Yüksek	Normal	İdeal	B
H4	Yaşlı	K	Normal	Normal	İdeal	B
H5	Yaşlı	E	Düşük	Normal	Şişman	B
H6	Yaşlı	E	Düşük	Yüksek	İdeal	A
H7	Orta Yaşlı	E	Düşük	Yüksek	Zayıf	B
H8	Genç	K	Normal	Normal	İdeal	A
H9	Genç	E	Düşük	Normal	İdeal	B
H10	Yaşlı	E	Normal	Normal	Şişman	B
H11	Genç	E	Normal	Yüksek	Şişman	B
H12	Orta Yaşlı	K	Normal	Yüksek	Şişman	B
H13	Orta Yaşlı	E	Yüksek	Normal	Aşırı Şişman	B
H14	Yaşlı	K	Normal	Yüksek	Şişman	A
H15	Orta Yaşlı	K	Düşük	Normal	Normal	?

Örnek Hasta Veri Seti.

Bir karar ağacı yaparken önce herhangi bir özelliği alıp onun alabildiği seçenekler ile diğer özelliğin seçeneklerinden bir yol oluştururuz. Burada önemli olan, sonucu farklılaştıran kırılımları yola dahil etmektir. Aşağıdaki örnekte görebileceğiniz gibi, Genç olan hastanın A yada B ilacını almasına karar vermek için sadece cinsiyet verisi kullanılabilir. “ Ki bu veriye göre vereceğimiz kararlardan biri doğrunun bu olacağı anlamına gelmiyor.” Fakat hasta yaşlı olduğu zaman ağacımızın dalları daha uzamaya ve farklılaşmaya başlıyor. Hasta Yaşlı ve Kadın ise sadece Kolestrol verisi bize hangi ilacı seçeceğimiz konusunda yardımcı oluyor. Eğer hastamız Yaşlı ve Erkek ise o zaman Tansiyon verisini de kullanmamız gerekiyor.



Karar Ağaçları (Decision Tree) Algoritması

Yukarıdaki resimde gördüğünüz karar ağacı Yaş özelliği ile başlıyor. Bu özellik kök olarak seçmek için en uygun olanı mı? Daha kısa yollar ile sonuca ulaşmamızı sağlayacak bir kök olabilir mi? Bu soruların yanıtı için seçilen özelliğin tüm veri seti içerisindeki öneminin bilinmesi gerekmektedir.

Özelliğin önemine nasıl karar vereceğiz? Bunun için her bir özellik için hedef verinin dağılımına bakmamız gerekiyor. Hangisinde dağılım daha homojen ise onun ayırt ediciliği dolayısı ile önemi daha fazladır.

Mesela aşağıdaki tablolara göre hangi özeliğin altındaki seçeneklerde ilaç dağılımının nasıl olacağını görebiliyoruz. Yaş Özelliğine göre

sınıflandırmak yerine cinsiyete göre sınıflandırmaya başlamak bu tabloya göre daha doğru görünüyor. Çünkü Erkek(E) yada Kadın(K) oluşuna göre olasılıkların yarısını eleyebildiğimiz gibi eğer hasta erkek ise 6/7 oranında B ilacını alacağını söyleyebiliriz.

YAŞ	
Özellik/İlaç	CI
Genç	5
A	3
B	2
Orta Yaşlı	4
B	4
Yaşlı	5
A	2
B	3
Grand Total	14

CİNSİYET	
Özellik/İlaç	CI
E	7
A	1
B	6
K	7
A	4
B	3
Grand Total	14

Kolestrol	
Özellik/İlaç	CI
Normal	8
A	2
B	6
Yüksek	6
A	3
B	3
Grand Total	14

TANSİYON	
Özellik/İlaç	CI
Düşük	4
A	1
B	3
Normal	6
A	2
B	4
Yüksek	4
A	2
B	2
Grand Total	14

BKE	
Özellik/İlaç	CI
Aşırı Şişman	2
A	1
B	1
İdeal	5
A	2
B	3
Şişman	6
A	2
B	4
Zayıf	1
B	1
Grand Total	14

Karar Ağaçları (Decision Tree) Algoritması — Doğru feature seçimi

Karar Ağaçları (Decision Tree) Algoritmasının asıl meselesi işte kademeli olarak doğru özelliğin seçilerek en kısa yol ile doğru seçenek için %100'e yakın olasılığı bulmaktır.

Bunu bulmak için her düğümde o düğüm için entropi hesaplanır. Entropi, bir verideki rastgeleliğin ölçülmesidir. Örnek olması için yukarıdaki tablolara bakacak olursak. Cinsiyet düğümünün entropisi 1'dir. Çünkü E ve K değerlerindeki örnek dağılımı 7'ye 7'dir. Fakat Yaş->Orta Yaşlı seçeneğinin entropisi 0'dır. Çünkü Orta yaşın altındaki değerlerin tamamı B'dir. Entropi ne kadar düşük olursa o veri seti o kadar homojen demektir.

Karar Ağaçları (Decision Tree) Algoritması bu entropi değerini nasıl kullanır? Direkt bu değeri kullanmak yerine Information Gain (Edinilen Bilgi) değerini kullanılır. Information gain, adından anlaşılacağı üzerine her adımda doğru bilgiye ulaşma oranı olarak tanımlamanabilir. Entropi küçüldükçe Information Gain artar.

Entropi her seçenek için ayrı ayrı hesaplanırken Information gain node için hesaplanır. Hesaplarken bir önceli düğümün entropisiden altındaki seçeneklerin entropisinden hesaplanan ağırlıklı entropi değeri çıkartılır. Sonuç olarak hangi düğümün Information Gain değeri daha yüksek ise öncelikli olarak o özelliklik kök düğüm olarak seçilir.

Python ile Karar Ağaçları (Decision Tree) Algoritması Örneği

Şu ana kadar anlatmış olduğumuz Karar Ağaçları (Decision Tree) Algoritmasının Python ve Scikit-Learn ile bir uygulamasını yapalım. Örneğimiz yukarıda bahsettiğimiz ilaç seçme uygulaması olacak.

```
df=pd.read_csv("ilac_tahmin.csv")
df.head()
```

	Yas	Cinsiyet	Tansiyon	Kolestrol	SoPo_Oran	İlaç
0	23	K	YÜKSEK	YÜKSEK	25.355	ilacY
1	47	E	DÜŞÜK	YÜKSEK	13.093	ilacC
2	47	E	DÜŞÜK	YÜKSEK	10.114	ilacC
3	28	K	NORMAL	YÜKSEK	7.798	ilacX
4	61	K	DÜŞÜK	YÜKSEK	18.043	ilacY

Analiz için veri seti

Veri Hazırlığı

Öncelikli olarak elimizdeki verinin analize hazır hale getirilmesi gerekiyor. Scikit-Learn ile modelimizi eğitmeden önce tahmin edeceğimiz özellik “ilaç” ve diğer özellikleri iki ayrı değişkene atayacağız. X tahmin için kullanacağımız özelliklerden oluşan matris olacak. y ise tahmmin edeceğimiz veriden oluşan bir vektör olacak.

```
X=df[["Yas","Cinsiyet","Tansiyon","Kolestrol","SoPo_Oran"]].values
X[0:5]
```

```
array([[23, 'K', 'YÜKSEK', 'YÜKSEK', 25.355],
       [47, 'E', 'DÜŞÜK', 'YÜKSEK', 13.093],
       [47, 'E', 'DÜŞÜK', 'YÜKSEK', 10.113999999999999],
       [28, 'K', 'NORMAL', 'YÜKSEK', 7.797999999999999],
       [61, 'K', 'DÜŞÜK', 'YÜKSEK', 18.043]], dtype=object)
```

Sklearn Decision Tree Algoritması kategorik verileri analiz edemiyor. Bu sebeple Cinsiyet, Tansiyon,Kolestrol verilerini sayısal kategorik verilere çevirmemiz gerekiyor. Bunu sağlamak için label encoding metodunu kullanırız.

```
from sklearn import preprocessing

label_cins=preprocessing.LabelEncoder()
label_cins.fit(["K","E"])
X[:,1]=label_cins.transform(X[:,1])

label_tans=preprocessing.LabelEncoder()
label_tans.fit(["YÜKSEK","DÜŞÜK","NORMAL"])
X[:,2]=label_tans.transform(X[:,2])

label_kol=preprocessing.LabelEncoder()
label_kol.fit(["YÜKSEK","NORMAL"])
X[:,3]=label_kol.transform(X[:,3])
```

```
X[0:5]
```

```
array([[23, 1, 2, 1, 25.355],
       [47, 0, 0, 1, 13.093],
       [47, 0, 0, 1, 10.113999999999999],
       [28, 1, 1, 1, 7.797999999999999],
       [61, 1, 0, 1, 18.043]], dtype=object)
```

```
y=df[["İlaç"]].values
y[0:5]
```

```
array([['ilacY'],
       ['ilacC'],
       ['ilacC'],
       ['ilacX'],
       ['ilacY']], dtype=object)
```

Modelin Oluşturulması

Öncelikli olarak verimizi train ve test olarak iki kısma ayıracağız. Yani elimizide X_train, X_test, y_train y_test olmak üzere 4 veri seti olacak.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state=3)
```

Oluşturduğumuz veri setlerinin boyutlarının analiz için birbiri ile uyumlu olması gerekiyor.

```
print("X_train boyut:",X_train.shape," y_train boyut:",y_train.shape)
print("X_test boyut:",X_test.shape," y_test boyut:",y_test.shape)
```

```
X_train boyut: (140, 5)  y_train boyut: (140, 1)
X_test boyut: (60, 5)  y_test boyut: (60, 1)
```

İlk olarak ilacTree isminde bir *DecisionTreeClassifier* instance oluşturacağız. Bu instance içinde, her bir düğümün information gain değerini görebilmemiz için *criterion="entropy"* olarak tanımlayacağız.

```
ilacTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
ilacTree # Aşağıda default parametreleri göreceğiz.
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=4, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Artık modeli eğitebiliriz.

```
ilacTree.fit(X_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=4, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Tahmin & Değerlendirme

Yukarıda oluşturmuş olduğumuz modeli kullanarak tahmin yapalım ve sonucları bir değişkende toplayalım.

```
ilacTahmin=ilacTree.predict(X_test)
```

```
ilacTahmin[0:5]
```

```
array(['ilacY', 'ilacX', 'ilacX', 'ilacX', 'ilacX'], dtype=object)
```

```
y_test[0:5]
```

```
array([['ilacY'],
       ['ilacX'],
       ['ilacX'],
       ['ilacX'],
       ['ilacX']], dtype=object)
```

İlk beş değer için mükemmel çalıştı :) Şimdi tüm veri seti üzerinden modelin başarımına bakalım.

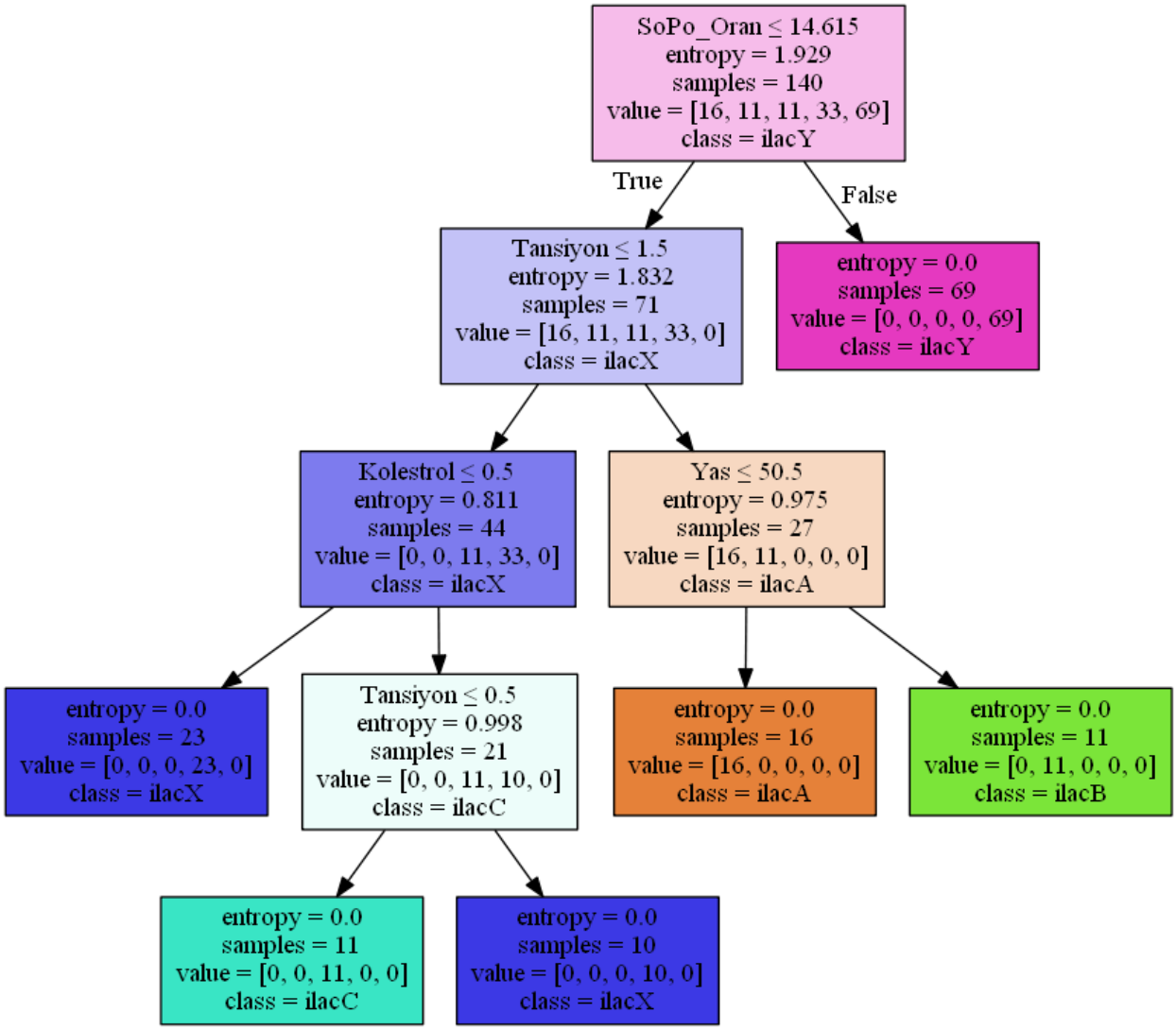
```
from sklearn import metrics
print ("Karar Ağacı Modeli Accuracy: ",metrics.accuracy_score(y_test,ilacTahmin))
```

```
Karar Ağacı Modeli Accuracy:  0.9833333333333333
```

Tüm test verisi üzerinden modelimizin başarımı %98 gibi gayet iyi bir değer verdi.

Karar ağacı modelinin görselleştirilmesi

Modelin oluşturdurğu karar ağacı modeline incelemek istersek onu bir resim olarak görüntüleyebiliriz. Oluşturulan modelin çıktısı aşağıdaki gibi olacaktır.



Decision Tree — Karar Ağacı

Destek Vektör Makineleri:

Destek Vektör Makineleri, temel olarak iki sınıfa ait verileri birbirinden en uygun şekilde ayırmak için kullanılır. Bunun için karar sınırları yada diğer bir ifadeyle hiper düzlemler belirlenir.

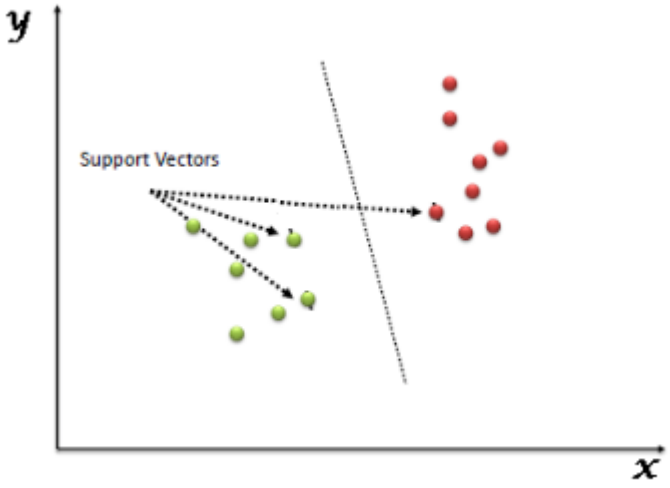
DVM’ler günümüzde yüz tanıma sistemlerinden, ses analizine kadar birçok sınıflandırma probleminde kullanılmaktadırlar.

Destek Vektör Makineleri, veri setinin doğrusal olarak ayrılabilme ve ayrılamama durumuna göre ikiye ayrılmaktadır.

1- Doğrusal Destek Vektör Makineleri

Destek vektör makineleriyle sınıflandırmada, iki sınıfa ait örneklerin doğrusal olarak dağıldığını varsayalım. Bu durumda bu iki sınıfın, eğitim verisi kullanılarak elde edilen bir karar fonksiyonu yardımıyla birbirinden ayrılması amaçlanır.

Veri setini ikiye ayıran doğru karar doğrusu olarak isimlendirilmektedir. Sonsuz tane karar doğrusu çizebilme imkanı mevcut olsada önemli olan optimal yani en uygun karar doğrusunu belirlemektir.

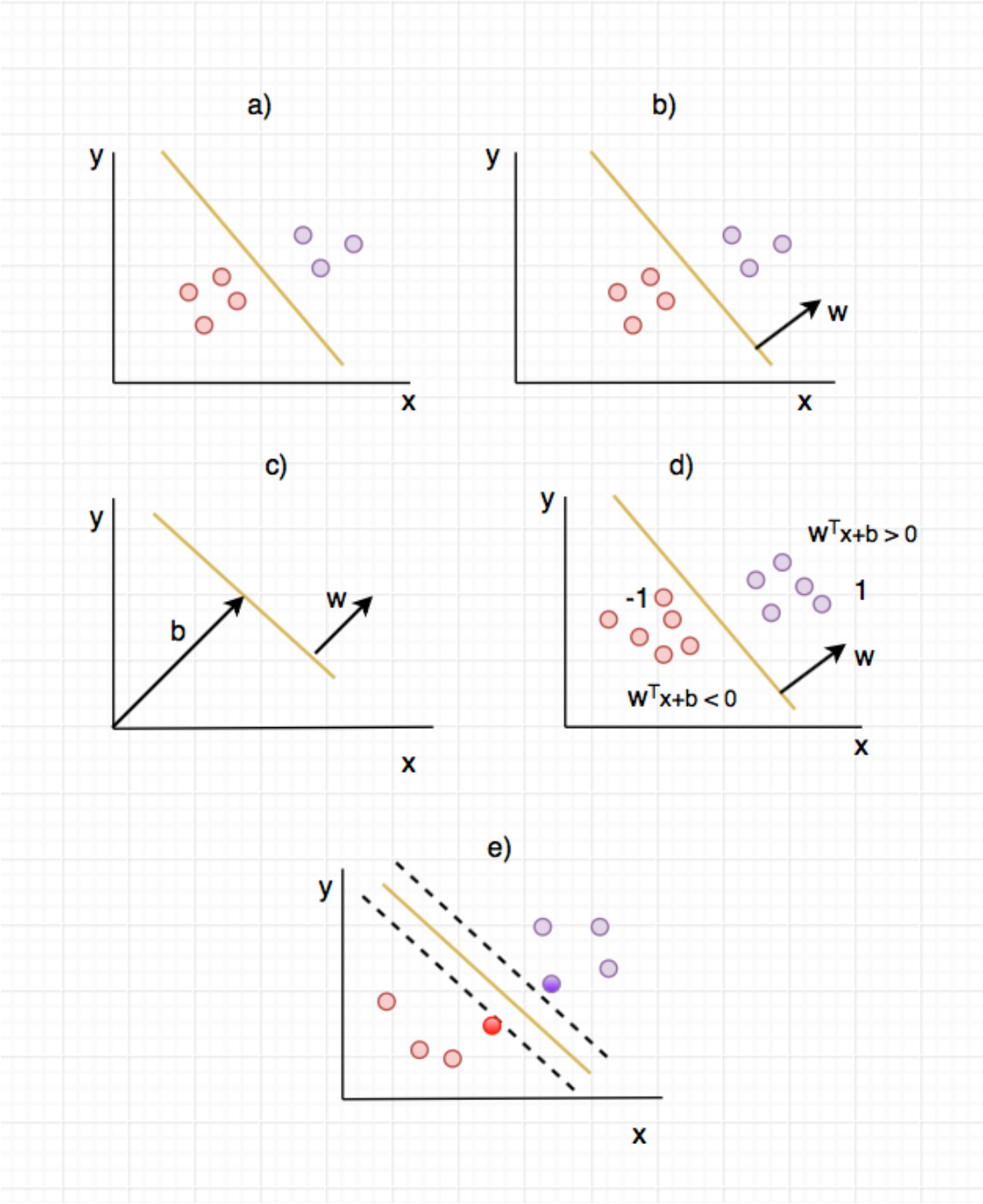


Şekil-1: Destek vektörleri

Karar doğrusunun yeni katılacak olan veriye karşı dayanıklı olabilmesi için sınır çizgisinin, iki sınıfın sınır çizgisilerine en yakın uzaklıkta olması

gerekmektedir. Bu sınır çizgisine en yakın noktalar, destek noktaları olarak adlandırılmaktadır.

Destek vektör makineleriyle sınıflandırmada genellikle (-1,+1) şeklinde sınıf etiketleri kullanılmaktadır.



Şekil-2: Doğrusal DVM

Destek vektörlerinin (Şekil-3'de kısım “e”) üzerinde bulunduğu ve kesikli çizgilerle gösterilmiş düzlemlere sınır düzlemleri denir. Sınır düzlemlerinin tam ortasından geçen ve her iki düzleme de eşit uzaklıkta bulunan düzlem ise hiper düzlem olarak ifade edilir. Şekil-3'de (-1, +1) sınıf etiketlerini, w ağırlık vektörünü (hiper-düzlemin normali) ve b ise eğilim değerini göstermektedir (Osuna ve ark., 1997; Kavzoğlu, T., ve Çölkesen, İ., 2010).

Bu şekilde doğrusal olarak dağılan veri setimiz DVM’ler yardımıyla sınıflandırılır.

1.1 Python ile Uygulama

Doğrusal SVM için basit bir python uygulaması yaparak aşağıdaki sonuçlar elde edilmiştir. Bu uygulama için “iris” verisinden yararlanılmıştır.

```
In [1]: import numpy as np
        from sklearn import datasets
        from sklearn.pipeline import Pipeline
        from sklearn.preprocessing import StandardScaler
        from sklearn.svm import LinearSVC

In [2]: # Iris Verisi
        iris = datasets.load_iris()
        X = iris["data"][:,(2,3)] # "taçyaprak(Petal) uzunluğu","petal genişliği"
        y = (iris["target"] == 2).astype(np.float64) # "Virginica" türü
        print(y)

[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.]

In [3]: # Pipeline
        svm_clf = Pipeline((
            ("scaler",StandardScaler()),
            ("linear_svc", LinearSVC(C=1, loss="hinge")),
            ))
        svm_clf.fit(X,y)

Out[3]: Pipeline(steps=(('scaler', StandardScaler(copy=True, with_mean=True, with
        _std=True)), ('linear_svc', LinearSVC(C=1, class_weight=None, dual=True,
        fit_intercept=True,
             intercept_scaling=1, loss='hinge', max_iter=1000, multi_class='ovr',
             penalty='l2', random_state=None, tol=0.0001, verbose=0))))

In [4]: svm_clf.predict([[4.5,2.1]])

Out[4]: array([ 1.])
```

Yaprak uzunluğu 4.5 cm, yaprak genişliği 2.1 cm olan bir yaprağı sınıflandırıcımız Iris-Virginica olarak tahmin etti :)

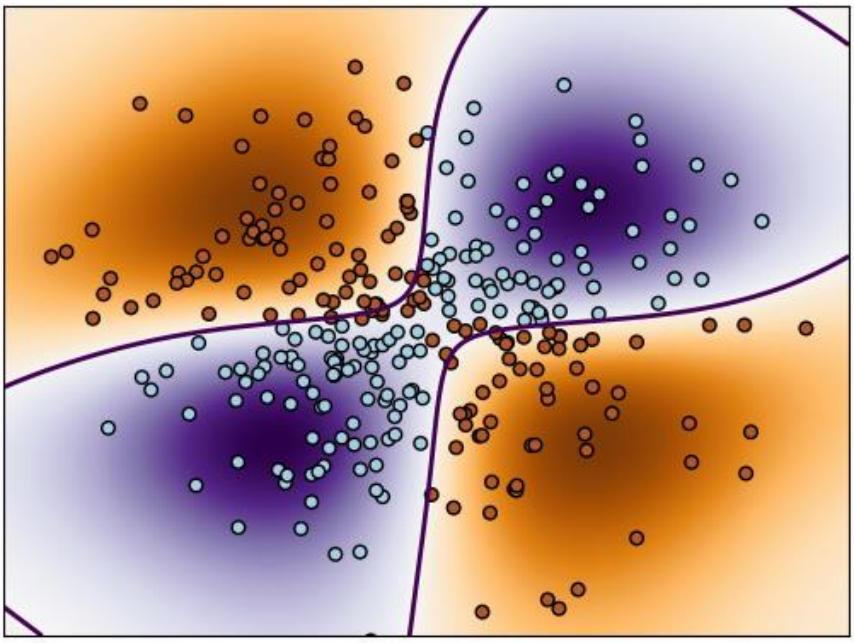
Python ile Doğrusal DVM Örneği

2- Doğrusal Olmayan Destek Vektör Makineleri

Doğrusal olarak dağılan örneklerde Destek Vektör Makineleri'nin nasıl çalıştığını ve bunları Python'da nasıl uygulayabileceğimizi konuştuktan sonra sıra doğrusal olmayanlara geldi. Doğrusal olmayan bir veri kümesinde DVM'ler doğrusal bir hiper-düzlem çizemez. Bu nedenle çekirdek numarası olarak adlandırılan *kernel trick*'ler kullanılır. Çekirdek yöntemi, doğrusal olmayan verilerde makine öğrenimini yüksek oranda arttırmaktadır.

En çok kullanılan çekirdek yöntemleri:

- Polynomial Kernel
- Gaussian RBF (Radial Basis Function) Kernel



Doğrusal olmayan DVM