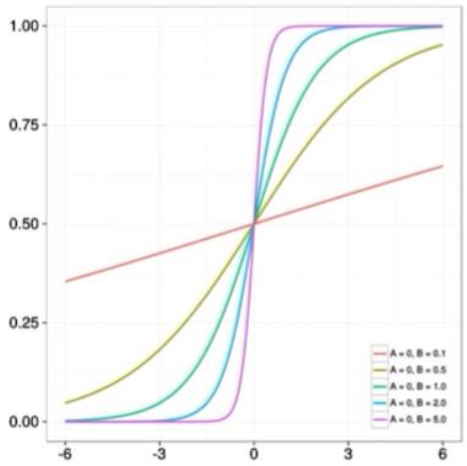
CLASSIFICATION

Logistic Function



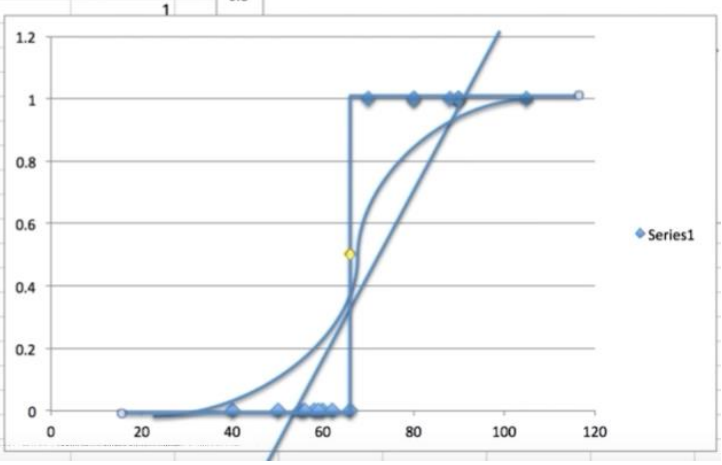
$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

$$t = \beta_0 + \beta_1 x \qquad \text{t=A+Bx}$$

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m = \beta_0 + \sum_{i=1}^{m} \beta_i x_i$$

	A	B	C	D	E	F
1	ulke	boy	kilo	yas	cinsiyet	
2	tr	130	30	10	e	1
3	tr	125	36	11	e	1
4	tr	135	34	10	k	0
5	tr	133	30	9	k	0
6	tr	129	38	12	e	1
7	tr	180	90	30	e	
8	tr	190	80	25	e	
9	tr	175	90	35	e	
10	tr	177	60	22	k	
11	us	185	105	33	e	
12	us	165	55	27	k	
13	us	155	50	44	k	
14	us	160	58		k	
15	us	162	59	41	k	
16	us	167	62	55	k	
17	fr	174	70	47	e	
18	fr	193	90		e	
19	fr	187	80	27	e	
20	fr	183	88	28	e	
21	fr	159	40	29	k	
22	fr	164	66	32	k	
23	fr	166	56	42	k	
24						



veri setini eğitim ve test kümelerine bölerken train_test_split fonksiyonunu kullanırsanız ve random_state = 0 olarak ayarlarsanız, her zaman aynı şekilde bölecektir. Bu, çalışmanızın tekrarlanabilirliğini sağlar ve sonuçların diğerleriyle karşılaştırılmasını kolaylaştırır.

KARMAŞIKLIK MATRİSİ(confusion matrix)

from sklearn.metrics import confusion_matrix

Karmaşıklık Matrisi

	C_1	C_2
C_1	True positive	False negative
C_2	False positive	True negative

classes	buy_computer = yes	buy_computer = no	total	recognition(%)
buy_computer = yes	6954	46	7000	99.34
buy_computer = no	412	2588	3000	86.27
total	7366	2634	10000	95.52

- Accuracy M, acc(M): model M için yüzde kaç doğru sınıflandırma olduğudur
  - Error rate (misclassification rate) = 1 – acc(M)
  - Alternatif ölçümler (e.g., for cancer diagnosis)

```
sensitivity = t-pos/(t-pos+f-neg)          /* true positive recognition rate */
specificity = t-neg/(t-neg+f-pos)          /* true negative recognition rate */
precision =  t-pos/(t-pos + f-pos)
accuracy = sensitivity * pos/(pos + neg) + specificity * neg/(pos + neg)
```

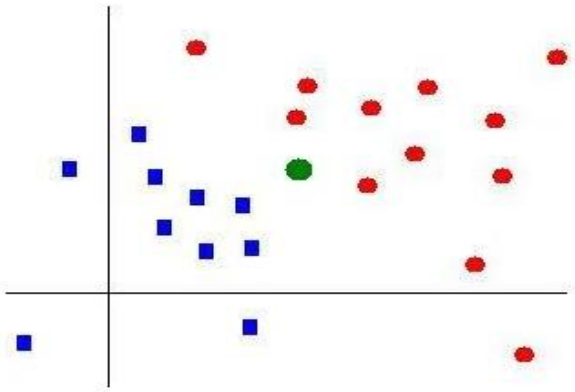
(sınıflandırma başarısını ölçmek)

```
log_reg = LogisticRegression(random_state=0)
log_reg.fit(x_train_sc, y_train)

y_pred = log_reg.predict(x_test_sc)

cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[ 9  5]
 [ 0 14]]
```

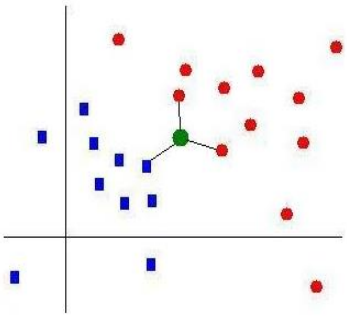
- Model, 9 örneği doğru bir şekilde negatif olarak sınıflandırmıştır (True Negatives - TN).
- Model, 5 örneği yanlış pozitif olarak sınıflandırmıştır (False Positives - FP).
- Model, 0 örneği yanlış negatif olarak sınıflandırmıştır (False Negatives - FN).
- Model, 14 örneği doğru bir şekilde pozitif olarak sınıflandırmıştır (True Positives - TP).

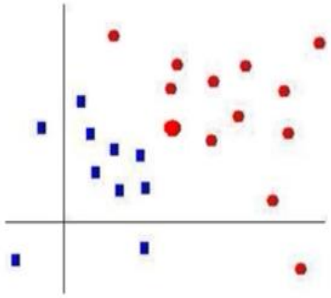
KNN (K nearest neighborhood, en yakın k komşu)



Yeşil örneğin sınıflandırılması için K-NN algoritmasi kullanılırsa,

En yakın 3(k==3 ise) komsudan 2 komsu kirmizi olduğu için yeşil örnek kırmızı sınıfına dahil edilecektir





K == 4[cift] icin en yakın komsuların 2 kırmızı 2 mavi olduğu durumda hangilerinin mesafesi daha yakınsa o sınıfa dahil edilir

Yakın örneklerin bulunması için *oklid bağlantısı* kullanıyor

[(x1, y1) ve (x2, y2) -> d = √(x2−x1)^2 +(y2−y1)^2]

K-NN'de kullanılan bazı öğrenme (learning) yöntemleri şunlardır:

1. **Lazy Learning (Tembel Öğrenme)**: K-NN, tembel öğrenme olarak bilinen bir yaklaşımı kullanır. Bu modelin eğitim aşamasında hiçbir şey yapmaması ve sadece tahmin yapmak için gelen veriye dayanması anlamına gelir. Veri seti alındığında, K-NN tüm veriyi belleğe yükler ve tahmin yapmak için gelen yeni örneklerle karşılaştırır. Bu nedenle, K-NN gerçek anlamda öğrenme yapmaz, sadece veri setinin doğrudan öğrenilmesi ve saklanması şeklinde çalışır.
2. **Instance-Based Learning (Örnek Temelli Öğrenme)**: K-NN, örnek temelli bir öğrenme yöntemidi Bu, modelin sınıflandırma yapmak için etiketli örneklerin bir koleksiyonunu kullanması anlamına gelir. Yeni bir örneği sınıflandırmak için, K-NN, bu örneği en yakın komşularına dayanarak sınıflandırır. Yani, öğrenme süreci, veri setinin örneklerine dayanır ve herhangi bir özniteliği çıkarmaz veya özetlemez.
3. **Distance-Based Learning (Uzaklık Temelli Öğrenme)**: K-NN, sınıflandırma yaparken örnekler arasındaki uzaklığı ölçerek karar verir. Bu nedenle, K-NN'nin bir tür uzaklık temelli öğrenme olduğu söylenebilir. Örneğin, en yakın komşularını seçerken, öklid mesafesi gibi bir uzaklık metriği kullanılır. Uzaklık temelli öğrenme, K-NN'nin temelinde yatan prensiplerden biridir.

```
NearestNeighbors(n_neighbors=3, algorithm='any_distance_function')
```

Metrics intended for real-valued vector spaces:

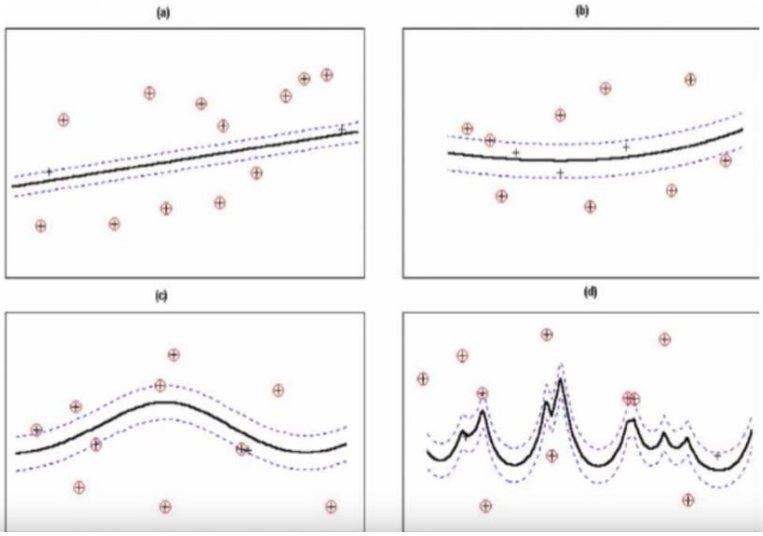
identifier	class name	args	distance function
"euclidean"	EuclideanDistance	•	`sqrt(sum((x - y)^2))`
"manhattan"	ManhattanDistance	•	`sum(|x - y|)`
"chebyshev"	ChebyshevDistance	•	`max(|x - y|)`
"minkowski"	MinkowskiDistance	p, w	`sum(w * |x - y|^p)^(1/p)`
"seuclidean"	SEuclideanDistance	V	`sqrt(sum((x - y)^2 / V))`
"mahalanobis"	MahalanobisDistance	V or VI	`sqrt((x - y)' V^-1 (x - y))`

```python
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5, metric='minkowski')
'''

metric =  'euclidean'
          'manhattan'
          'chebyshev'
          'minkowski'
          'seuclidean'
          'mahalanobis'
'''
knn.fit(x_train_sc, y_train)

y_pred = knn.predict(x_test_sc)

cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
In [22]: runf
KNN(k_nearest
machine_learn
[[14  1]
 [ 2 11]]
C:\Users\terz
column-vector
example using
  return self

In [23]: runf
KNN(k_nearest
machine_learn
[[14  1]
 [ 2 11]]

In [24]: runf
KNN(k_nearest
machine_learn
[[14  1]
 [ 2 11]]
```

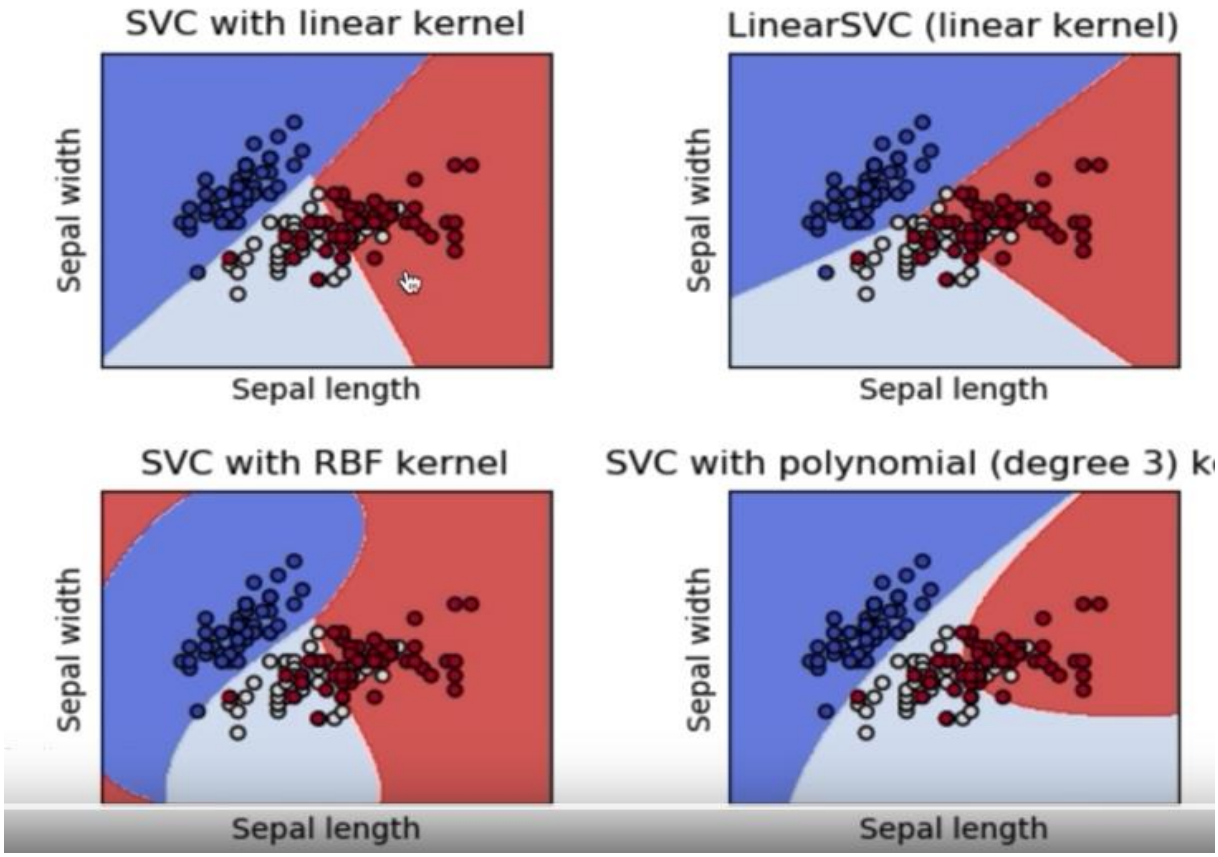
- Model, 14 örneği doğru bir şekilde negatif olarak sınıflandırmıştır (True Negatives - TN).
- Model, 1 örneği yanlış pozitif olarak sınıflandırmıştır (False Positives - FP).
- Model, 2 örneği yanlış negatif olarak sınıflandırmıştır (False Negatives - FN).
- Model, 11 örneği doğru bir şekilde pozitif olarak sınıflandırmıştır (True Positives - TP).

Destek Vektör Regresyonu (Support Vector Regression)

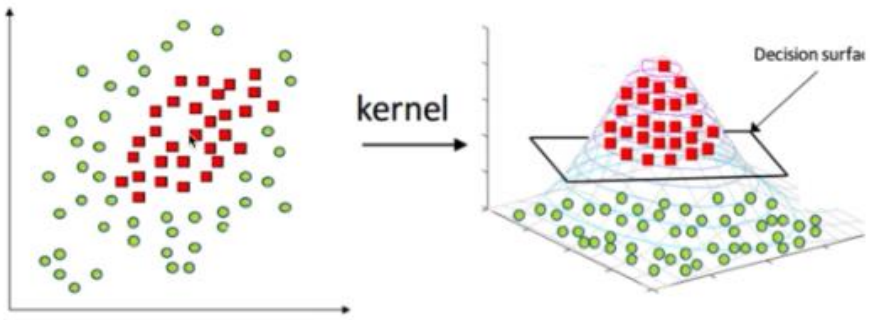


a. Doğrusal (Linear)
b. Polinom (Polynomial)
c. Guassian (RBF)
d. Üssel (Exponential)

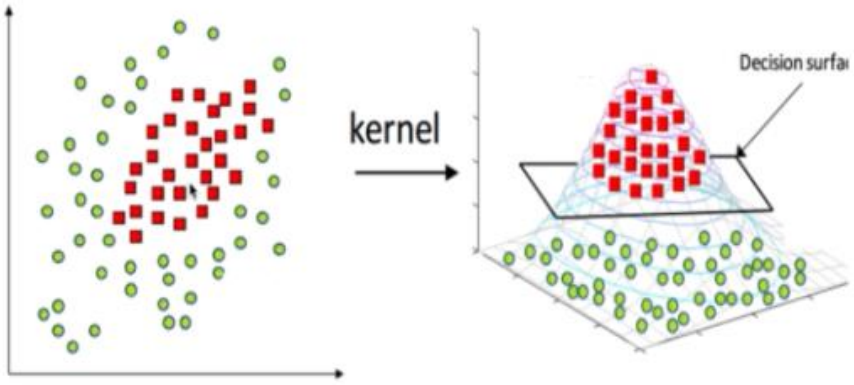
Support vector classifier



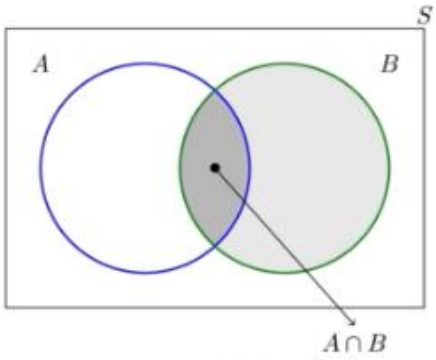
DVM Çekirdek Hilesi



SVM'NİN doğrusal ayrılma özelliğini 3.boyuta taşıyarak bir kernel trick(çekirdek hilesi) yaparak doğrusal olarak ayrıştırılamayacak verileri ayrıştırmayı hedefliyoruz



Naive Bayes (Koşullu Olasılık)



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

Naive Bayes (Naif Bayes)

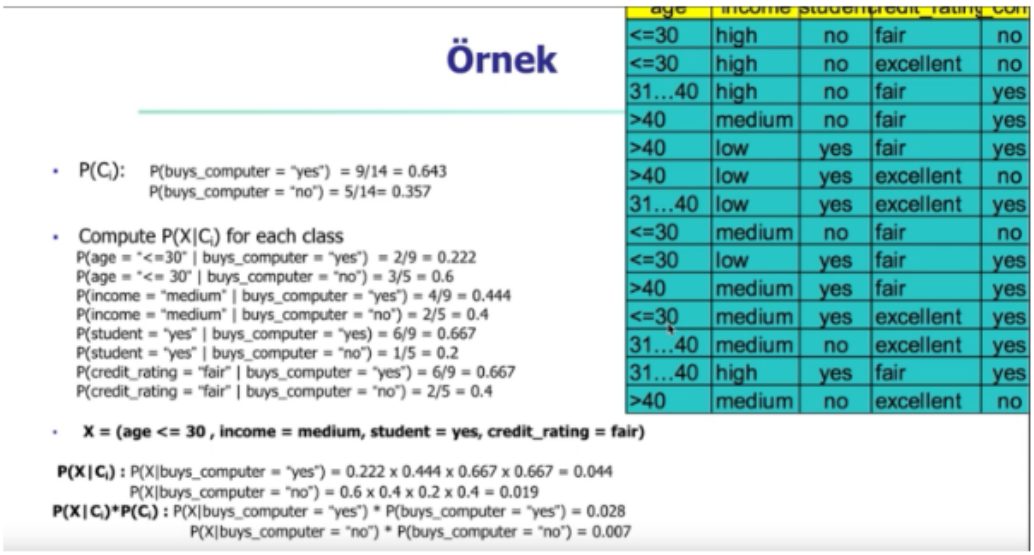
$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}$$

Sınıf:
C1:buys_computer = 'yes'
C2:buys_computer = 'no'

Örnek Veri
X = (age <=30,
Income = medium,
Student = yes
Credit_rating = Fair)

age	income	student	credit_rating	com
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

X'in gerçeklesmesinden sonra C'nin gerçekleşme ihtimali (veya tam tersi)



age	income	student	credit_rating	com
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Naive Bayes bir sınıflandırma algoritması ancak sınıflandırma işlemi sırasında **olasılıkları** kullanıyor **Verilerin olasılıksal dağılımlarına bakarak bir sınıflandırma işlemi yapıyor**

Bütün dağılıma bakarak her bir bireyin(kolonun) dağılım üzerindeki etkisini ölçerek olasılıksal bir şekilde modelliyor.

Naive bayes de KNN gibi lazy learning yönetmini kullanıyor.

Lazy learning(tembel öğrenme) : Veri geldikten sonra elimizdeki veri setine göre gelen veriyi sınıflayacak . Yeni bir veri gelmeden herhangi bir işlem yapmaz . (veri geldikten sonra sormak)

Eager learning (istekli öğrenme) : veri gelmeden önce elindeki veri kümesi üzerinden öğrenir (bütün ihtımalleri veri gelmeden hesaplamaları yapar) bütün olasılıkları bi tarafta bekletiyor olacak .

(veri kümesini unutabilir) öğrendigi olasılıkları aklında tutarak gelen veriyi sınıflandırabilir

Dezavantaj: veri kümesi cok detaylı ve karmaşık olduğunda eager learning calistirmak maliyetlidir

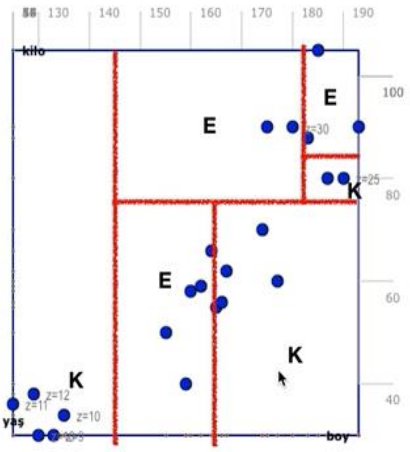
Temel 3 yöntem ;

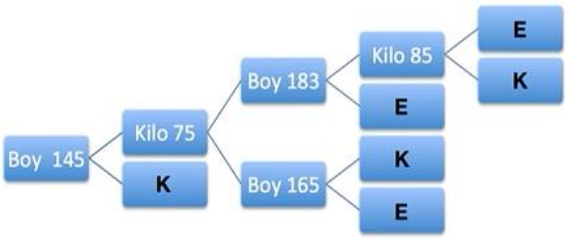
Gaussian Naive Bayes : tahmin edilecek sınıf sürekli bir değerse(ondalıklı reel) ise

Multinomial Naive Bayes : Örnek olarak: üniversite isimleri 1,2,3,4,5 gibi tam sayılar verip tahmin için kullanılabilir

Bernouilli Naive Bayes : Örnek olarak: kadın erkek , sigara içiyor içmiyor (binary nominal değerler)

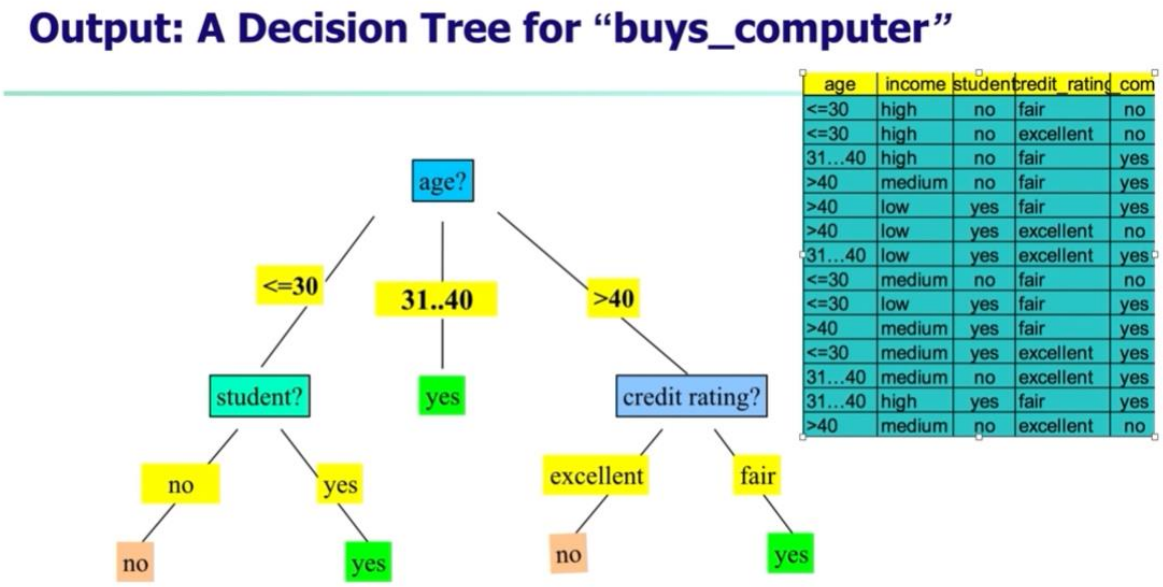
Karar Ağacı ile Sınıflandırma (Decision Tree)





Bu örnekteki sağ alttaki K sınıfını alternatif olarak daha fazla alt sınıflara bölebilir veya çoğunluğa göre (Majority voting) sınıflandırma yapılabilir

Ancak daha fazla alt sınıflara bölmek overfitting'e yol acabilir.



age	income	student	credit_rating	com
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Buradaki Problem Kök Bağımsız değişkenin hangisi olacağını seçmek(NEDEN AGE ?)

Bunun için farkli algoritmalar kullanılabilir;

ID3 algoritması aşağıdaki adımları izler:

1. **Başlangıç:** Veri kümesi ve bu veri kümesi için sınıflar verilir.
2. **Özellik Seçimi:** ID3, her bir özelliğin veri kümesini nasıl ayırdığına bakarak en iyi özelliği seçer. Bunu yapmak için, her bir özelliğin veri kümesini nasıl daha homojen alt kümelerde böldüğünü ölçer. Veri kümesi içindeki en büyük bilgi kazancını (information gain) sağlayan özelliği seçer. Bilgi kazancı, bir özelliğin veri kümesini ne kadar iyi ayırdığına dair ölçüdür. Bilgi kazancı, ayrıldıktan sonra en homojen alt kümelere sahip olmayı sağlayan özelliği seçer.
3. **Karar Ağacı Oluşturma:** Seçilen özelliği kullanarak bir düğüm oluşturulur ve bu düğüm altındaki veri kümesi bu özellikle ayrıştırılır. Ardından, her bir alt küme için aynı işlem tekrarlanır ve ağaç derinleştirilir. Bu işlem, belirlenen bir durum oluşana veya ayrıştırılabilecek veri kalmayana kadar devam eder.
4. **Tekrar Etme:** Yukarıdaki adımlar, tüm veri kümesi tek bir sınıfa ait olana kadar tekrarlanır veya belirli bir kurala veya kriterlere ulaşılıncaya kadar devam eder.
5. **Ağaç Oluşturma Bitirme:** Ağaç oluşturma işlemi tamamlandığında, bir sonraki adıma geçilir.
6. **Pruning (budama):** Oluşturulan ağaç, gereksiz dallardan arındırılabilir. Bu, ağacın daha basit ve daha genelleştirici olmasını sağlar ve aşırı uyum (overfitting) riskini azaltır.

ID3 algoritması, ağacın oluşturulması sırasında veri kümesinin her durumu için tüm özellikleri inceleyerek, tüm olası ağaçları oluşturur. Bu, büyük veri kümeleri için hesaplama açısından maliyetli olabilir, bu nedenle genellikle daha gelişmiş algoritmalara tercih edilebilir.

Information Gain (Enformasyon Kazanımı) ID3 için

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times I(D_j)$$

$$Gain(A) = Info(D) - Info_A(D)$$

ID 3

- Class P: buys_computer = "yes"
- Class N: buys_computer = "no"

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.940$$

$$Info_{age}(D) = \frac{5}{14}I(2,3) + \frac{4}{14}I(4,0) + \frac{5}{14}I(3,2) = 0.694$$

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

$\frac{5}{14}I(2,3)$ means "age <=30" has 5 out of 14 samples, with 2 yes'es and 3 no's. Hence

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

Similarly,

$$Gain(income) = 0.029$$
$$Gain(student) = 0.151$$
$$Gain(credit\_rating) = 0.048$$

ncepts and Techniques

BTK AKADEMİ

ALTERNATİF ALGORİTMALAR ;

ID3 gibi temel karar ağacı algoritmalarının bazı sınırlamaları vardır ve bu nedenle daha gelişmiş veya optimize edilmiş algoritmalar tercih edilebilir. İşte bazı alternatifler:

1. **C4.5 ve C5.0:** ID3'ün geliştirilmiş versiyonlarıdır. ID3'ün aksine, eksik verilerle başa çıkabilirler ve sayısal verilere de uyum sağlarlar. Ayrıca, C4.5 ve C5.0, ağacı daha sonra budamak için bir mekanizma sunar, bu da aşırı uyumu azaltmaya yardımcı olur.
2. **CART (Classification and Regression Trees):** Sınıflandırma ve regresyon ağaçlarını destekler. CART, her adımda en iyi bölünmeyi seçmek için bir karmaşık fonksiyon kullanır ve bu sayede daha karmaşık veri yapılarını ele alabilir.
3. **Random Forest:** Random Forest, birden fazla karar ağacının bir araya getirilmesiyle oluşan bir ensemble (topluluk) öğrenme algoritmasıdır. Her bir ağaç, rastgele özellikler alt kümesi üzerinde eğitilir ve sonuçlarını bir araya getirerek daha genelleştirilmiş ve daha güvenilir bir sonuç elde edilir. Ayrıca, overfitting'e karşı daha dirençlidirler.
4. **Gradient Boosting Trees:** Bu, zayıf öğrenicileri (genellikle karar ağaçlarını) bir araya getirerek güçlü bir tahminci oluşturan bir ensemble öğrenme tekniğidir. XGBoost ve LightGBM gibi kütüphaneler, bu algoritmayı uygulamak için popüler seçeneklerdir. Gradient boosting, karmaşık yapıları modellemek ve yüksek doğruluk sağlamak için etkilidir.
5. **Quinlan'ın Yeni Karar Ağaçları:** Bu algoritma, C4.5'ten daha hızlıdır ve daha az bellek tüketir. Daha büyük veri setleriyle çalışırken daha uygun olabilir.

İşte C4.5 ve C5.0'ın ana özellikleri:

1. **Eksik Veri Uyumluluğu:** C4.5 ve C5.0, eksik verilerle başa çıkabilir. Eksik verilerle karşılaşıldığında, bu algoritmalar eksik verileri dikkate alarak en iyi bölünmeyi bulmaya çalışırlar.
2. **Sayısal Değerlerin Kullanımı:** C4.5 ve C5.0, sayısal özellikleri doğrudan işleyebilir. Bu, ID3'te olduğu gibi sayısal değerlerin önceden kategorilere dönüştürülmesine gerek olmadığı anlamına gelir.
3. **Budama (Pruning):** C4.5 ve C5.0, gereksiz dalları budamak için bir mekanizma sunar. Bu, ağacın daha genelleştirilmiş ve daha az karmaşık olmasını sağlar, böylece aşırı uyumu önler.
4. **Karmaşıklık Azaltma:** C4.5 ve C5.0, ağacın karmaşıklığını azaltmak için özellik seçiminde farklı yöntemler kullanır. ID3'ten farklı olarak, C4.5 ve C5.0, bilgi kazancı yerine normalize edilmiş bilgi kazancını kullanır. Bu, daha dengeli bir karar ağacı oluşturulmasına yardımcı olur.

CART algoritması aşağıdaki adımları izler:

1. **Başlangıç:** Veri kümesi ve bu veri kümesi için hedef değişken (sınıflandırma için sınıf etiketleri, regresyon için sürekli değerler) verilir.
2. **Özellik ve Eşik Değeri Seçimi:** Her adımda, veri kümesini en iyi şekilde bölecek bir özellik ve eşik değeri seçilir. Bu seçim, belirli bir kriter kullanılarak yapılır (örneğin, sınıflandırma için Gini impurity veya Entropy, regresyon için ortalama hata azalması gibi).
3. **Bölünme İşlemi:** Seçilen özellik ve eşik değeri kullanılarak veri kümesi iki alt kümeye bölünür. Her bir alt küme için aynı işlem tekrarlanır.
4. **Tekrar Etme:** Yukarıdaki adımlar, belirli bir duruma ulaşılıncaya veya bölünebilecek veri kalmayana kadar tekrarlanır.
5. **Ağaç Oluşturma Bitirme:** Ağaç oluşturma işlemi tamamlandığında, bir sonraki adıma geçilir.
6. **Pruning (Budama):** Oluşturulan ağaç gereksiz dallardan arındırılabilir. Bu, ağacın daha basit ve daha genelleştirici olmasını sağlar ve aşırı uyumu azaltır.

Confusion Matrix

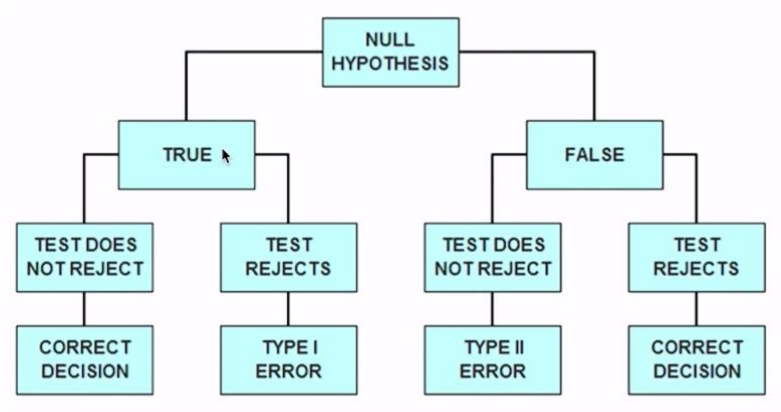
Örnek Değerler

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

- **Accuracy:** Kaç doğru sınıflandırma var?
  - (TP+TN)/total = (100+50)/165 = 0.91
- **Misclassification Rate:** Kaç yanlış sınıflandırma yapılmış?
  - (FP+FN)/total = (10+5)/165 = 0.09
  - 1 eksi Accuracy
  - "Error Rate" olarak da geçer
- **True Positive Rate:** Gerçekte Yes ise kaçı doğru sınıflanmış?
  - TP/actual yes = 100/105 = 0.95
  - Aynı zamanda "Sensitivity" veya "Recall" da denir
- **False Positive Rate:** Tahmin No ise bu sonuçların kaçı doğru?
  - FP/actual no = 10/60 = 0.17
- **Specificity:** Gerçekte No ise bunların kaçı doğru sınıflanmış?
  - TN/actual no = 50/60 = 0.83
  - 1 eksi False Positive Rate olarak da hesaplanır
- **Precision:** Tahmin Yes ise kaçı doğru?
  - TP/predicted yes = 100/110 = 0.91
- **Prevalence:** Gerçekteki Yes dağılmı oranı
  - actual yes/total = 105/165 = 0.64

Hata

- False Positive (Tip 1 Hata)
- False Negative (Tip 2 Hata)



```
[[16  0  0]
 [ 0 14  5]
 [ 0  2 13]]
```

- 16 örneğin gerçek sınıfı 0, ve algoritma bu örnekleri doğru bir şekilde sınıflandırmış.
- 14 örneğin gerçek sınıfı 1, algoritma 14'ünü doğru bir şekilde tahmin etmiş, ancak 5 örneği yanlış sınıflandırmış.
- 13 örneğin gerçek sınıfı 2, algoritma 13'ünü doğru bir şekilde tahmin etmiş, ancak 2 örneği yanlış sınıflandırmış.

Accuracy Paradox

Accuracy Paradox
Doğruluk Paradoksu

	Tahmin C_1	Tahmin C_2
Gerçek C_1	9900	0
Gerçek C_2	100	0

ZeroR : algoritması

	Tahmin C_1	Tahmin C_2
Gerçek C_1	9500	400
Gerçek C_2	50	50

Burada anlatılmak istenen;

Herhangi bir algoritmanın başarısını sadece accuracy değerine bakıp ölçemezsin. Accuracy değerini attıran fakat başarıyı düşüren *(yukarıda görülen ZeroR algoritması) durumlar olabilir.*

[ZeroR = makine öğrenmesi uygulama, test verisindeki çoğunluk hangi sınıftaysa yeni gelen veriyi o sınıfa dahil et] (en islevsiz durum)

Receiver Operating Characteristic (ROC)

$$tpr = \frac{TP}{TP + FN}$$

$$fpr = \frac{FP}{FP + TN}$$

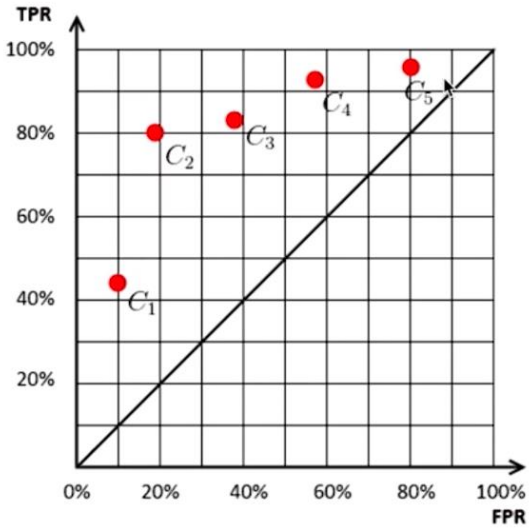
True Positive Rate: Gerçekte Yes ise kaçı doğru sınıflanmış?
- TP/actual yes = 100/105 = 0.95
- Aynı zamanda "Sensitivity" veya "Recall" da denir

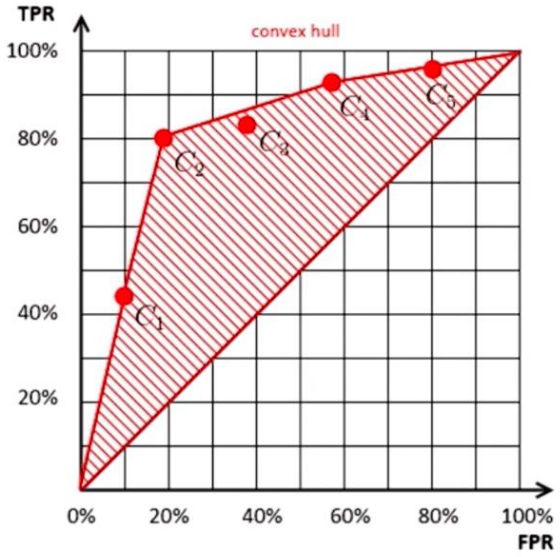
- **False Positive Rate:** Gerçekte No ise bu sonuçların kaçı doğru?
  - FP/actual no = 10/60 = 0.17

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

True Positive Rate (TPR), gerçek pozitif oranı ifade eder ve doğru pozitiflerin toplam pozitifler içindeki oranını belirtir. False Positive Rate (FPR), yanlış pozitif oranıdır ve yanlış pozitiflerin toplam negatifler içindeki oranını belirtir.

ROC Convex Hull





C'leri sınıflandırma algoritmaları olarak düşünebiliriz. Accuracy paradox olduğunu varsayıp ROC ile karşılaştıralım

Doğrusal olarak birleştirilmesi sonucunda bazı algoritmalar bu doğrunun içinde kalıyor (C3).

Bu algoritmalar faydalı algoritma değil ve çıkartılabilir

Classifier comparison