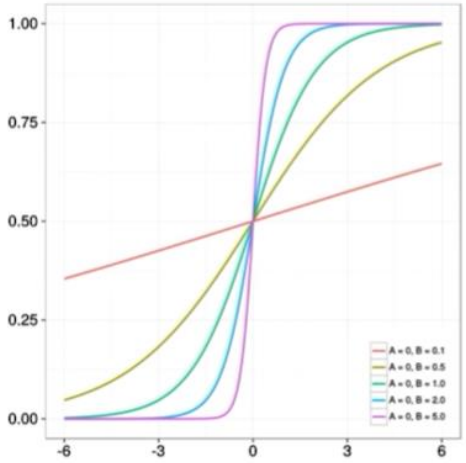
CLASSIFICATION

Logistic Function



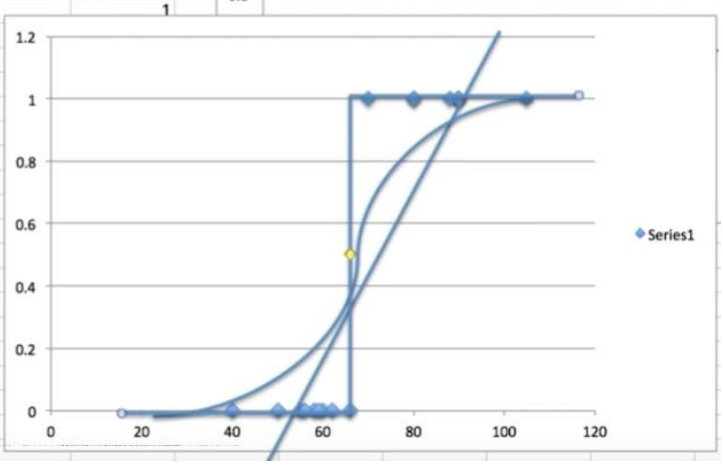
$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

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

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

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

	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)
```

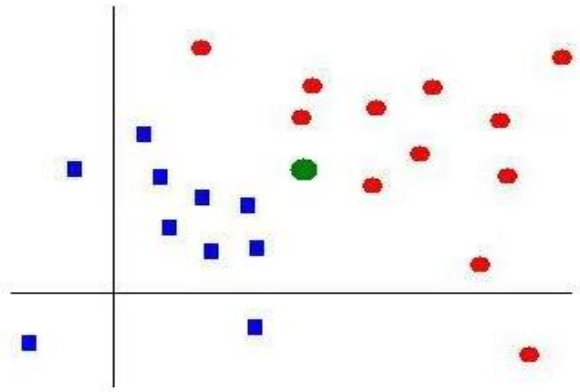
(sınıflandırma başarısını ölçmek)

```
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]]
```

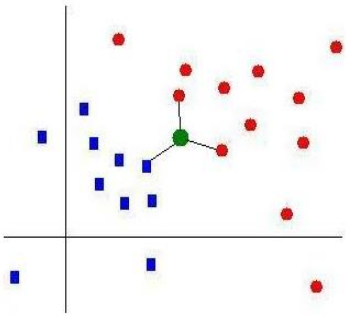
- Model, 9 örneği doğru bir şekilde negatif olarak sınıflandırmıştır (True Negatives - TN).
- Model, 5 örneği yanlış pozitif olarak sınıflandırmıştır (False Positives - FP).
- Model, 0 örneği yanlış negatif olarak sınıflandırmıştır (False Negatives - FN).
- Model, 14 örneği doğru bir şekilde pozitif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

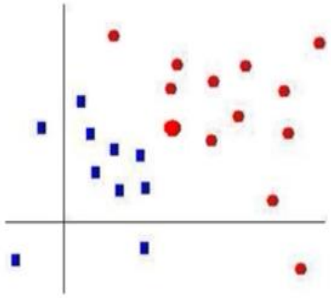
KNN (K nearest neighborhood, en yakın k komşu)



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]

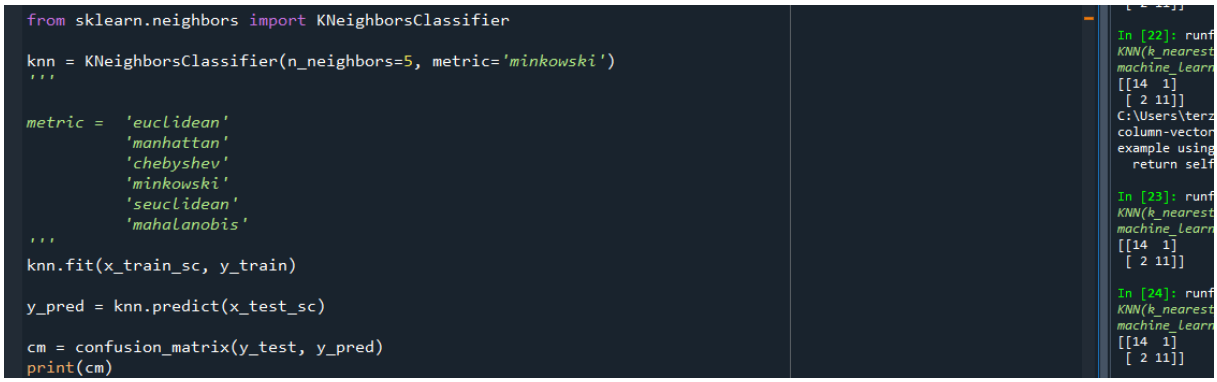
K-NN'de kullanılan bazı öğrenme (learning) yöntemleri şunlardır:

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 yöntemidir. 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))`

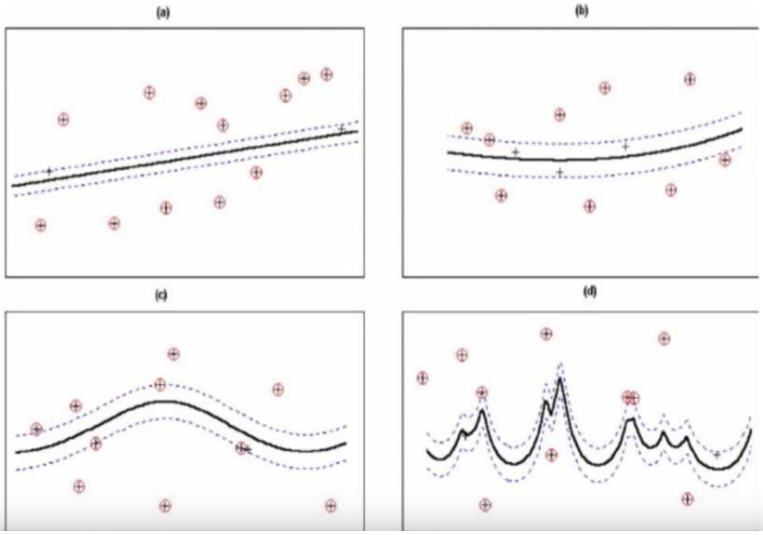


```
from sklearn.neighbors import KNeighborsClassifier

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