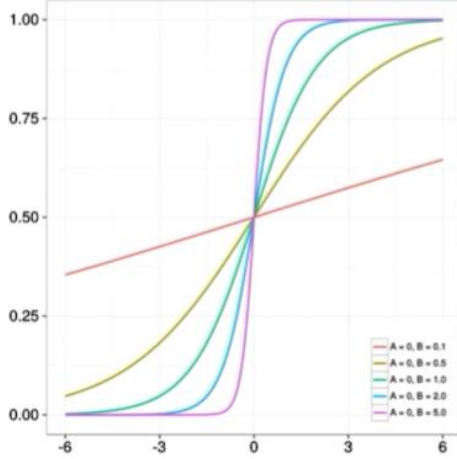


## CLASSIFICATION

# Logistic Function

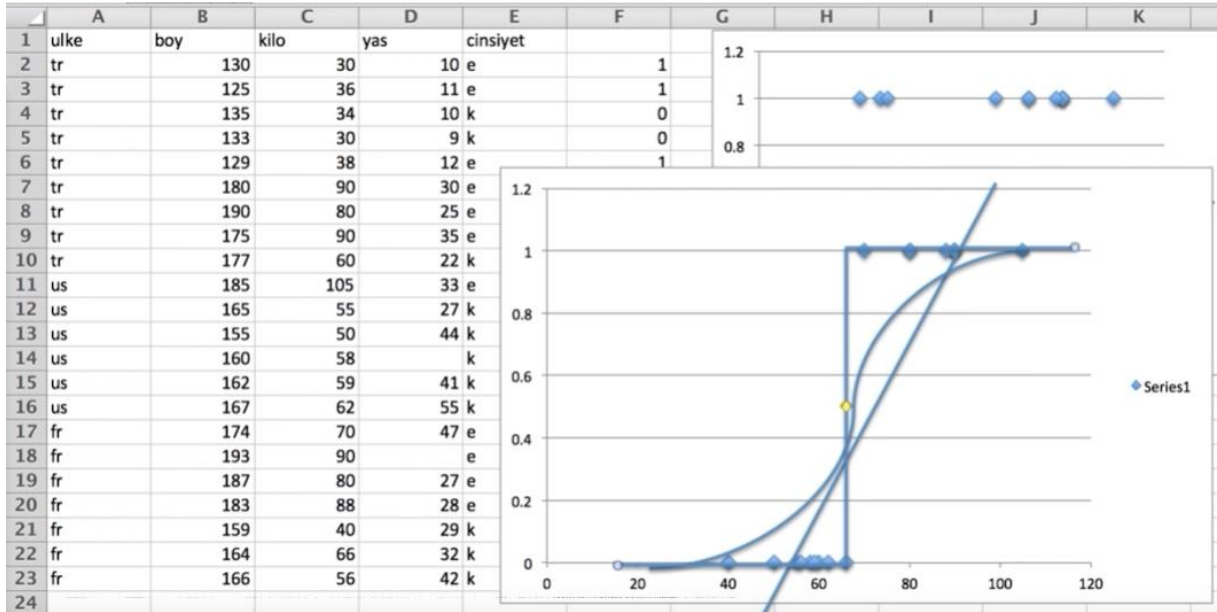


$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

$$t = \beta_0 + \beta_1 x \quad t = A + Bx$$

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

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



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 <sub>1</sub>	C <sub>2</sub>
C <sub>1</sub>	True positive	False negative
C <sub>2</sub>	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) =  $1 - \text{acc}(M)$
  - Alternatif ölçümler (e.g., for cancer diagnosis)
    - sensitivity =  $t\text{-pos} / (t\text{-pos} + f\text{-neg})$  /\* true positive recognition rate \*/
    - specificity =  $t\text{-neg} / (t\text{-neg} + f\text{-pos})$  /\* true negative recognition rate \*/
    - precision =  $t\text{-pos} / (t\text{-pos} + f\text{-pos})$
    - accuracy =  $\text{sensitivity} * \text{pos} / (\text{pos} + \text{neg}) + \text{specificity} * \text{neg} / (\text{pos} + \text{neg})$

(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)
```

```
wdir='C:/Users/t
basari_orani= %
C:/Users/terzi\A
vector y was pas
using ravel().
y = column_or_

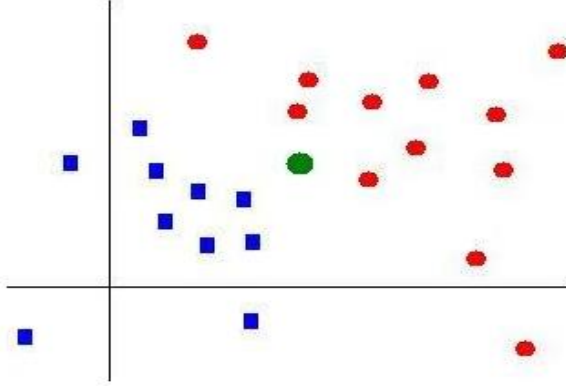
In [30]: runfile
wdir='C:/Users/t
[[ 9  5]
 [ 0 14]]
C:/Users/terzi\A
vector y was pas
using ravel().
y = column_or_

In [31]:
```

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

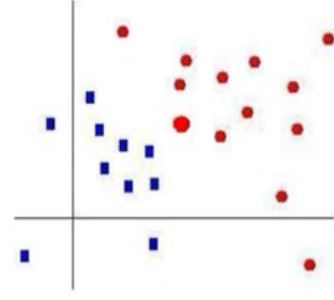
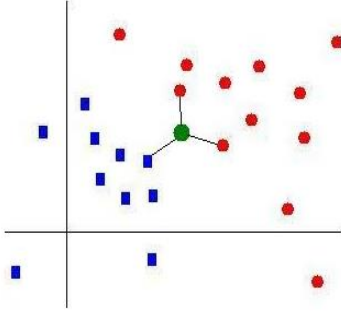
- Model, 9 örneği doğru bir şekilde negatif olarak sınıflandırmıştır (True Negatives - TN).
- Model, 5 örneği yanlış pozitif olarak sınıflandırmıştır (False Positives - FP).
- Model, 0 örneği yanlış negatif olarak sınıflandırmıştır (False Negatives - FN).
- Model, 14 örneği doğru bir şekilde pozitif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

## KNN (K nearest neighborhood, en yakın k komşu)



Yeşil örneğin sınıflandırılması için K-NN algoritması kullanılırsa,

En yakın 3 (k=3 ise ) komşudan 2 komşu kırmızı olduğu için yeşil örnek kırmızı sınıfına dahil edilecektir



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

Yakın örneklerin bulunması için *oklid bağlantısı* kullanıyor

$$[(x_1, y_1) \text{ ve } (x_2, y_2) \rightarrow d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}]$$

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 özneteliği 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).