# KNN (K-Nearest Neighbours, K-En Yakın Komşu) Algoritması Nedir

***En Yakın Komşu Algoritması – EYKA*** *(**K-Nearest Neighbours – KNN**)*; sınıflandırma işleminde bulunulacak örnek veri noktasının bulunduğu sınıfın (öğrenim kümesi) ve en yakın komşunun (elemanın), k değerine (benzerliğe) göre belirlendiği bir denetimli/gözetimli makine öğrenme yöntemi olarak ifade edilmektedir.

Literatür üzerinde yer alan temel/basit ifadesi ile tekrar tanımlanacak olursa;

Sınıfı bilinmeyen verilerin, eğitim setindeki diğer veriler ile karşılaştırılıp bir uzaklık ölçümü gerçekleştirilmesi sonucu hesaplanan uzaklığa göre henüz bir sınıfa atanamamış verinin, en ideal (optimal) sınıfa atanarak sınıflandırılması olarak ifade edilmektedir.

## *ÖZGEÇMİŞ*

***KNN algoritması***; 1950'li yılların başında ***Evelyn Fix*** ve ***Joseph L. Hodges Jr.*** tarafından geliştirilmiş, 1965 yılında N.J. Nilsson'ın ***minimum uzaklık sınıflayıcı çalışmaları*** ile gelişimi hız kazanmış ve 1967 yılında ilk olarak genişletilip istatiksel analizlere (***Regresyon*** – ***Regression***) yönelik kullanılması/önerilmesi ***Thomas M. Cover*** ve ***Peter. E. Hart*** tarafından gerçekleşmiştir.

KNN algoritması; ***hem sınıflandırma*** hem de regresyon çalışmaları içerisinde kullanılması yönü ile popüler olarak tercih edilen bir algoritmadır.

Genel sınıflandırma algoritmaları (modelleri) kendi çözümleri içerisinde bir sınıflayıcı oluşturarak sistem içerisindeki her veri değeri üzerinde bu sınıflayıcıyı kullanır.Genel olarak bu sınıflandırma algoritmalarına nazaran KNN algoritması her değer için, ilgili değere en yakın komşu kümesi üzerinden bir sınıflayıcı oluşturarak değerleri sınıflandırır. (Doğruluk değeri ideal seviyedeki bir sınıflandırma çözümü.)

Elbetteki tekrarlanan bu sınıflayıcı algoritmik çözüm, sınıflandırma sürecinin zaman almasına neden olmaktadır.
Bundan ötürü literatürde ***Tembel Öğrenci Algoritması*** (Yöntemi) olarak da ifade edilmektedir.

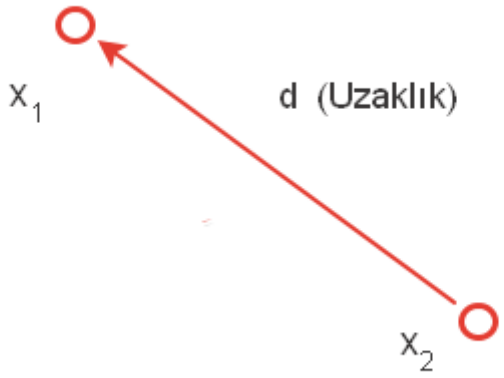
# KNN Uzaklık Hesaplaması Algoritma Türleri

## *Euclidean (Öklidyen) Uzaklık Hesaplaması*

***Euclidean*** (***Öklidyen***) ***Uzaklık Hesaplaması***; ***Öklid Uzayı*** $\mathbb{R}^n$ içinde, iki nokta arası verilen mesafe hesaplaması olarak ifade edilmektedir.Geliştiren: ***Euclid*** (*Öklid*) ve ***Pythagoras*** (*Pisagor*)n boyutlu uzayda Euclidean (Öklidyen) Uzaklık hesaplaması;

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

Euclidean Uzaklık Hesaplaması temsili görünümü; (Noktasal);




## *Manhattan Uzaklık Hesaplaması*

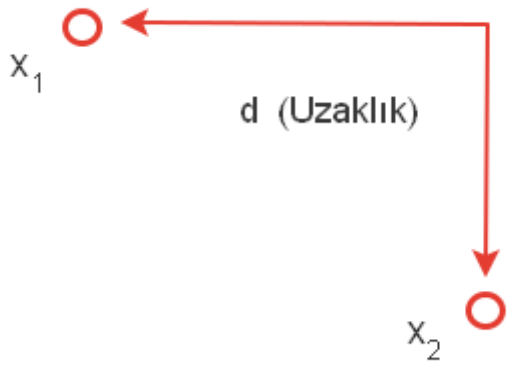
***Manhattan Uzaklık Hesaplaması***; $\mathbb{R}^n$ içinde (uyarlanabilir) eksenler boyunca dik açılarda ölçülen iki nokta arasındaki mesafe hesaplaması olarak ifade edilmektedir.

Geliştiren: ***Hermann Minkowski*** (Bilinen)

n boyutlu uzayda Manhattan Uzaklık hesaplaması;

$$d_1(\mathbf{p}, \mathbf{q}) = \|\mathbf{p} - \mathbf{q}\|_1 = \sum_{i=1}^{n} |p_i - q_i|$$

Manhattan Uzaklık Hesaplaması temsili görünümü; (Noktasal);




## *Chebyshev Uzaklık Hesaplaması*

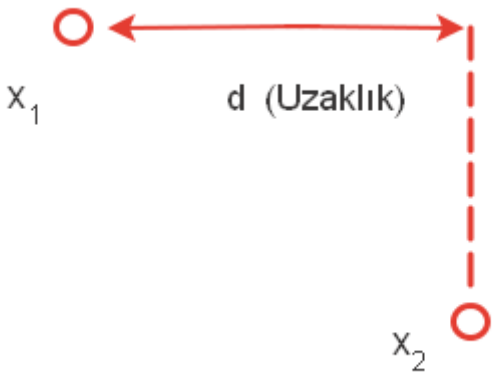
***Chebyshev Uzaklık Hesaplaması***; $R^n$ içinde (uyarlanabilir) herhangi bir koordinat boyutu boyunca iki vektör arasındaki en büyük fark içeren mesafe hesaplaması olarak ifade edilmektedir.

Geliştiren: ***Pafnuty Chebyshev***

n boyutlu uzayda Chebyshev Uzaklık hesaplaması;

$$D_{\text{Chebyshev}} = \max\left(|x_2 - x_1|, |y_2 - y_1|\right)$$

Chebyshev Uzaklık Hesaplaması temsili görünümü; (Noktasal);



## *Hamming Uzaklık Hesaplaması*

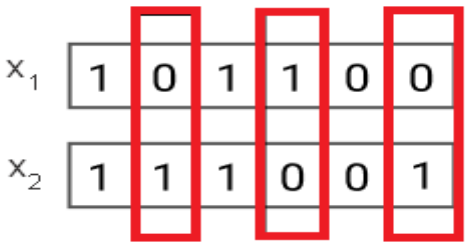
***Hammign Uzaklık Hesaplaması***; $R^n$ içinde (uyarlanabilir) iki vektör arasındaki farklı olan değerlerin sayısı olarak ifade edilmektedir. (Değer sayısı mesafe olarak ifadelendirilir.)

Geliştiren: ***Richard Hamming***

n boyutlu uzayda Hamming Uzaklık hesaplaması; (algoritmik olarak)

**"karolin"** ve **"kathrin"** = 3
**"karolin"** ve **"kerstin"** = 3
**"kathrin"** ve **"kerstin"** = 4
**1011101** ve **1001001** = 2
**2173896** ve **2233796** = 3

Hamming Uzaklık hesaplaması temsili görünümü;




## *Minkowski Uzaklık Hesaplaması*

***Minkowski Uzaklık Hesaplaması***; $R^n$ içinde (uyarlanabilir) normlu kullanılan bir metrik* mesafe hesaplaması olarak ifade edilmektedir.

*Uzaklık kavramının (mesafe) vektör olarak temsil edilmesi. (Genel temsili tanım olarak.)

Geliştiren: ***Hermann Minkowski***

n boyutlu uzayda Minkowski Uzaklık hesaplaması;

$$P = (x_1, x_2, \ldots, x_n) \text{ and } Q = (y_1, y_2, \ldots, y_n) \in \mathbb{R}^n$$

$$\left( \sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

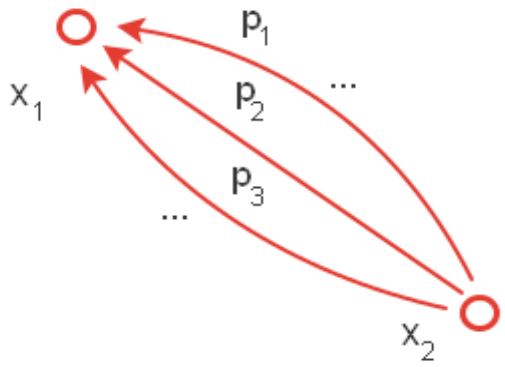
Minkowski Uzaklık Hesaplaması gerçekleştirilirken denkleme yönelik üç ana gereksinim mevcuttur. Bunlar;

- *Sıfır Vektör* (*Zero Vector*): Başlangıç ve bitiş noktası aynı olan yönlü doğru parçalarının temsil ettiği vektör olarak ifade edilmektedir.
  Boyu/Uzunluğu sıfır olan vektördür.
  Temsili olarak;

$\overrightarrow{AA} = \overrightarrow{BB} = \overrightarrow{CC} = \overrightarrow{O} = \dots$ şeklinde çözümlenir ve; $\overrightarrow{O}$ şeklinde gösterilir.

- *Skaler Faktör* (*Scaler Factor*): Tanımlı bir vektörün yönünü koruyan ama uzunluğunu değiştirmeyen değer olarak ifade edilmektedir.
- *Üçgen Eşitsizliği Çözümlemesi* (*Triangle Inequality Analysis*) : Geometrik eleman olan üçgene ait kenarların uzunluklarına yönelik eşitsizlik çözümü olarak ifade edilmektedir.

Minkowski Uzaklık Hesaplaması temsili görünümü; (Noktasal)




## *Mahalanobis Uzaklık Hesaplaması*

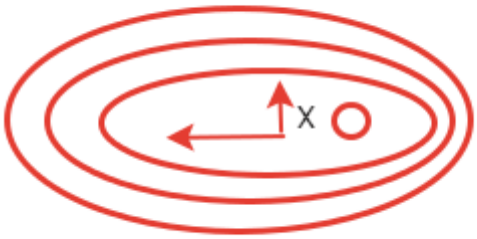
***Mahalanobis Uzaklık Hesaplaması***; $R^n$ içinde (uyarlanabilir) verinin varyans-kovaryans yapısını göz önüne alan ve tekil bir aykırı gözlem denemesi için kullanılabilen bir uzaklık formülü tanımlayan mesafe hesaplaması olarak ifade edilmektedir.

Geliştiren: ***Prasanta Chandra Mahalanobis***

n boyutlu uzayda Mahalanobis Uzaklık hesaplaması;

$$D_M(\vec{x}) = \sqrt{(\vec{x} - \vec{\mu})^\mathsf{T} \mathbf{S}^{-1} (\vec{x} - \vec{\mu})}$$

Mahalanobis Uzaklık Hesaplaması temsili görünümü; (Noktasal)




## *Haversiene Uzaklık Hesaplaması*

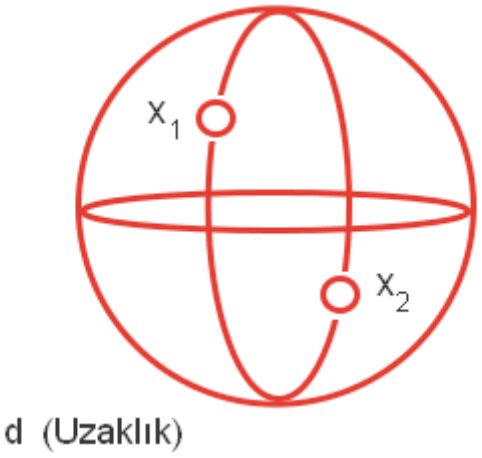
***Haversine Uzaklık Hesaplaması***; $R^n$ içinde (uyarlanabilir) enlem ve boylam ve değerleri kullanarak bir küre üzerindeki (Genel olarak) mesafe hesaplaması olarak ifade edilmektedir.

Geliştiren: ***James Inman***

n boyutlu uzayda Haversine Uzaklık hesaplaması;

$$d = 2r \arcsin\left(\sqrt{\operatorname{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\operatorname{hav}(\lambda_2 - \lambda_1)}\right)$$
$$= 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

Haversine Uzaklık Hesaplaması temsili görünümü; (Noktasal)



## *Levenshtein Uzaklık Hesaplaması*

***Levenshtein Uzaklık Hesaplaması***; iki dizi/dizilim arasındaki benzerliği derecelendirmek olarak ifade edilmektedir. (Değer sayısı mesafe olarak ifadelendirilir.)

Geliştiren: ***Vladimir Levenshtein***

Levenshtein Uzaklık hesaplaması; (Temsili)

| X | $y_1$ | $y_2$ | $y_3$ | $y_4$ | $y_5$ |
|---|---|---|---|---|---|
| a | 1 | 0 | 0 | 0 | 0 |
| b | 0 | 1 | 0 | 1 | 0 |
| c | 0 | 1 | 0 | 0 | 1 |
| d | 1 | 0 | 1 | 0 | 0 |

## *Sørensen-Dice Katsayısı Uzaklık Hesaplaması*

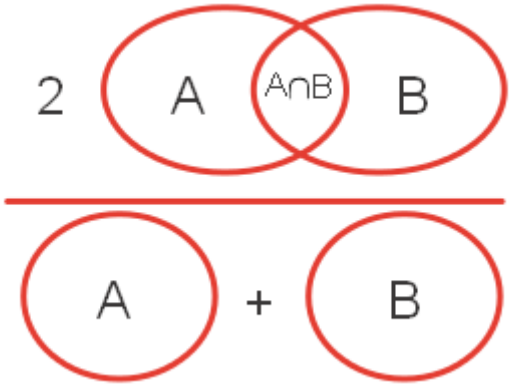
***Sørensen-Dice Uzaklık Hesaplaması***; sezgisel ağırlıklı olarak iki veri seti arasındaki örtüşme yüzdesi incelenip
0 ile 1 arasında bir değer verilerek değerlendirme gerçekleştirilmesi olarak ifade edilmektedir. (Değer mesafe olarak ifadelendirilir.)

Geliştiren: ***Thorvald Sørensen*** *ve* ***Lee Raymond Dice***

Sørensen-Dice Uzaklık hesaplaması;

$$DSC = \frac{2|X \cap Y|}{|X| + |Y|}$$

Sørensen-Dice Uzaklık Hesaplaması temsili görünümü; (Algoritmik)



## *Jacckard İndeksi Uzaklık Hesaplaması*

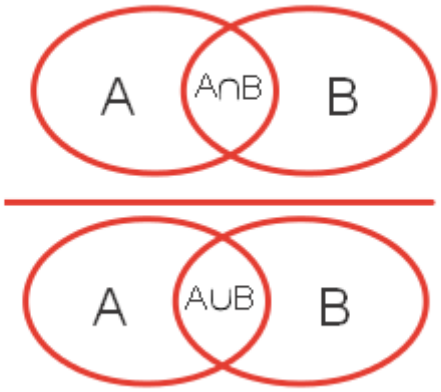
***Jacckard Uzaklık Hesaplaması***; veri setleri üzerindeki benzerlik ya da farklılıkları inceleyip 0 ile 1 arasında bir değer verilerek değerlendirme gerçekleştirilmesi olarak ifade edilmektedir. (Değer mesafe olarak ifadelendirilir.)

Geliştiren: ***Paul Jacckard***

Jacckard Uzaklık hesaplaması;

$$d_J(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

Jacckard Uzaklık Hesaplaması temsili görünümü; (Algoritmik)




## KNN (K-Nearest Neighbours, K-En Yakın Komşu) Algoritması Uygulama Örneği

KNN (K-Nearest Neighbours, K-En Yakın Komşu) algoritması uygulamasına örnek olarak, çokça duyulan ve kullanılan Iris Veri Seti üzerinden basit bir örnek ile çözümlemelerde bulunalım.

Iris Veri Seti 3 Iris bitki türüne (Iris Setosa, Iris Virginica ve Iris Versicolor) ait, her bir türden 50 örnek olmak üzere toplam 150 örnek sayısına sahip bir veri setidir.

Iris Veri Seti içerisinde;

Sınıflar (Türler);

- Iris Setosa,
- Iris Versicolor,
- Iris Virginica.

Veri Özellikleri (Ortak Özellikler);

- Sepal Uzunluk (cm),
- Sepal Genişlik (cm),
- Petal Genişliği (cm)
- Petal Uzunluk (cm).

1-Gerçekleştireceğimiz analizler için kullanacağımız kütüphaneleri sırası ile projemize dahil edelim; (Sklearn, Numpy, Seaborn, MatplotLib)

```
>>> import seaborn as sns
    iris = sns.load_dataset("iris")
    import pandas as pd
    import matplotlib.pyplot as pltis")
```

```
import seaborn as sns
iris = sns.load_dataset("iris")
import pandas as pd
import matplotlib.pyplot as plt
"""ÇALIŞMA ÜZERİNDE KULLANILACAK KÜTÜPHANELERİ DAHİL EDELİM."""
```

2-Çalışmamıza dahil ettiğimiz verilerimizin ön izlemesini gerçekleştirelim;

```
>>> iris.sample(7)
```

```
"""VERİLERE YÖNELİK ÖNİZLEME GERÇEKLEŞTİRELİM."""
iris.sample(7)
```

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| 78 | 6.0 | 2.9 | 4.5 | 1.5 | versicolor |
| 32 | 5.2 | 4.1 | 1.5 | 0.1 | setosa |
| 26 | 5.0 | 3.4 | 1.6 | 0.4 | setosa |
| 119 | 6.0 | 2.2 | 5.0 | 1.5 | virginica |
| 18 | 5.7 | 3.8 | 1.7 | 0.3 | setosa |
| 82 | 5.8 | 2.7 | 3.9 | 1.2 | versicolor |
| 7 | 5.0 | 3.4 | 1.5 | 0.2 | setosa |

**3**-Çalışmamıza dahil ettiğimiz verilerimizin değerlerinin ön izlemesini gerçekleştirelim;

>>> pd.value_counts(iris.species)

```
"""MEVCUT VERİLERİN DEĞERLERİNİN ÖNİZLEMESİNİ GERÇEKLEŞTİRELİM."""
pd.value_counts(iris.species)
```

```
versicolor    50
setosa        50
virginica     50
Name: species, dtype: int64
```

**4**-Çalışmamız üzerindeki veri setinden test ve eğitim veri setleri oluşturalım;

```
>>> train = iris.sample(frac=0.6, random_state=450)
    test = iris.drop(train.index)
```

```
"""VERİ SETİMİZ ÜZERİNDE EĞİTİM VE TEST VERİLERİ OLUŞTURALIM."""
train = iris.sample(frac=0.6, random_state=450)
test = iris.drop(train.index)
```

**5**-Oluşturmuş olduğumuz eğitim ve test veri setleri ile hedef modelimizin kurgusunu gerçekleştirelim;

```
>>> X_train = train.drop(labels='species', axis=1)
    y_train = train.species
    X_test = test.drop(labels='species', axis=1)
    y_test = test.species
    print(X_train.head(),"\n", y_train.head())
```

```
"""X_test - ALGORİTMA TAHMİNİ İÇİN VERİ SETİ
   y_test - HEDEF MODELİ DOĞRULAMAK İÇİN VERİ SETİ"""
X_train = train.drop(labels='species', axis=1)
y_train = train.species
X_test = test.drop(labels='species', axis=1)
y_test = test.species
print(X_train.head(),"\n", y_train.head())
```

```
     sepal_length  sepal_width  petal_length  petal_width
21            5.1          3.7           1.5          0.4
124           6.7          3.3           5.7          2.1
145           6.7          3.0           5.2          2.3
50            7.0          3.2           4.7          1.4
2             4.7          3.2           1.3          0.2
 21         setosa
124     virginica
145     virginica
50     versicolor
2          setosa
Name: species, dtype: object
```

**6**-Oluşturmuş olduğumuz eğitim ve test veri setleri arasındaki değer yoğunluğunu kontrol ettiğimizde;

>>> X_train.shape,X_test.shape
    y_train.shape,y_test.shape

```
"""EĞİTİM VE TEST VERİLERİ DEĞER FARKLILIKLARI"""
X_train.shape,X_test.shape
y_train.shape,y_test.shape
```

```
((90,), (60,))
```

**7**-Çalışmamıza ***Sklearn*** kütüphanesi üzerinden optimizasyon ve ek analizler için paketimizi dahil edelim;

>>> from sklearn.neighbors import KNeighborsClassifier

```
"""ÇALIŞMAMIZA SKLEARN ÜZERİNDEN ANALİZİMİZİ DAHİL EDELİM."""
from sklearn.neighbors import KNeighborsClassifier
```

**8**-Analizimizi en yakın n=4 eleman üzerinden analiz gerçekleştirilecek şekilde kurgulayalım;

>>> knn = KNeighborsClassifier(n_neighbors=4)
    knn.fit(X_train,y_train)

```
"""EN YAKIN N=4 VERİ ÜZERİNDEN ANALİZ GERÇEKLEŞECEĞİNİ KURGULAYALIM.;"""
knn = KNeighborsClassifier(n_neighbors=4)
knn.fit(X_train,y_train)
```

**9**-Mevcut yapımız üzerinde ortalama tahmin gerçekleştirelim;

>>> knn.score(X_test,y_test)

```
"""SONUÇLARI TEST ETMEDEN ORTALAMA TAHMİN GERÇEKLEŞTİRELİM."""
knn.score(X_test,y_test)
```

```
0.9666666666666667
```

**10**-Test sonuçlarımızı kullanabilmek için bir parametre üzerine atayıp tahminleme gerçekleştirelim;

>>> results = knn.predict(X_test)

```
"""TEST SONUCLARIMIZI BİR PARAMETREYE ATAYALIM."""
results = knn.predict(X_test)
```

**11**-Test -y- verilerimizin ön izlemesini gerçekleştirip, listeleyelim;

>>> print(pd.crosstab(y_test, results, rownames=['Real'], colnames=['Predicted'], margins=True, margins_name='total'))

```
"""TEST -y- VERİLERİMİZİN ÖN İZLMESİNİ GERÇEKLEŞTİRELİM."""
print(pd.crosstab(y_test, results, rownames=['Real'], colnames=['Predicted'], margins=True, margins_name='total'))
```

```
Predicted   setosa  versicolor  virginica  total
Real
setosa          17           0          0     17
versicolor       0          20          2     22
virginica        0           0         21     21
total           17          20         23     60
```

**12**-Doğruluk kontrolü adına ***Sklearn Metrics*** kütüphanesi çalışmamıza dahil edelim;

>>> from sklearn import metrics

```
"""DOĞRULUK KONTROLÜ GERÇEKLEŞTİRMEK İÇİN KÜTÜPHANEMİZİ DAHİL EDELİM."""
from sklearn import metrics
```

**13**-Doğruluk kontrolü işlemimizi gerçekleştirelim;

>>> print(metrics.classification_report(y_test,results,target_names=iris.species.unique(), digits=3))

```
"""DOĞRULUK KONTROLÜMÜZÜ ÇALIŞTIRALIM."""
print(metrics.classification_report(y_test,results,target_names=iris.species.unique(), digits=3))
```

```
              precision    recall  f1-score   support

      setosa      1.000     1.000     1.000        17
  versicolor      1.000     0.909     0.952        22
   virginica      0.913     1.000     0.955        21

    accuracy                          0.967        60
   macro avg      0.971     0.970     0.969        60
weighted avg      0.970     0.967     0.967        60
```

Kontrol sonrası (9) ve (11) maddelerdeki değerlerin aynı çıktığı gözlemlenmektedir.

**14**-1-25 arasındaki elemanları listeleyelim; (Tercihe bağlı)

>>> k_list = list(range(1,25))

```
"""1-25 ARASINDAKİ ELEMANLARI LİSTELEYELİM."""
k_list = list(range(1,25))
```

**15**-Tercihe bağlı olarak oluşturmuş olduğumuz listemizi görüntüleyelim; (Kontrol edelim)

>>> k_values = dict(n_neighbors=k_list)
print(k_values.keys()),
print(k_values.values())

```
"""LİSTELEDİĞİMİZ ELEMANLARI YAZDIRALIM.-GÖRÜNTÜLEME-"""
k_values = dict(n_neighbors=k_list)
print(k_values.keys()),
print(k_values.values())
```

```
dict_keys(['n_neighbors'])
dict_values([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]])
```

**16**-Kapsamlı analiz için ***Grid Search*** kütüphanemizi çalışmamıza dahil edelim;

>>> from sklearn.model_selection import GridSearchCV

```
"""DEĞERLERİMİZ ÜZERİNDE KAPSAMLI ANALİZ ARAMASI İÇİN
GRİD SEARCH KÜTÜPHANEMİZİ ÇALIŞMAMIZA EKLEYELİM."""
from sklearn.model_selection import GridSearchCV
```

**17**-***Grid Search*** nesnemizi analiz verilerimize yönelik eğitelim;

>>> grid = GridSearchCV(knn, k_values, cv=5, scoring='accuracy')
grid.fit(iris.drop('species', axis=1), iris.species)

```
"""GRID SEARCH NESNEMİZİ ANALİZ VERİLERİMİZE YÖNELİK EĞİTELİM."""
grid = GridSearchCV(knn, k_values, cv=5, scoring='accuracy')
grid.fit(iris.drop('species', axis=1), iris.species)
```

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(n_neighbors=4),
             param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                         13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
                                         23, 24]},
             scoring='accuracy')
```

**17**-Eğitmiş olduğumuz ***Grid Search*** nesnemiz üzerindeki verilerimizi görüntüleyelim;

>>> grid.cv_results_
grid_table = pd.DataFrame(grid.cv_results_)
grid_table.head()

```
"""EĞİTMİŞ OLDUĞUMUZ NESNEMİZİ GÖRÜNTÜLEYELİM."""
grid.cv_results_
grid_table = pd.DataFrame(grid.cv_results_)
grid_table.head()
```

| | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_n_neighbors | params | split0_test_score | split1_test_score |
|---|---|---|---|---|---|---|---|---|
| 0 | 0.003788 | 0.000745 | 0.004455 | 0.001388 | 1 | {'n_neighbors': 1} | 0.966667 | 0.966667 |
| 1 | 0.002194 | 0.000402 | 0.003195 | 0.000394 | 2 | {'n_neighbors': 2} | 0.966667 | 0.933333 |
| 2 | 0.002194 | 0.000398 | 0.002992 | 0.000020 | 3 | {'n_neighbors': 3} | 0.966667 | 0.966667 |
| 3 | 0.001994 | 0.000001 | 0.002986 | 0.000014 | 4 | {'n_neighbors': 4} | 0.966667 | 0.966667 |
| 4 | 0.001988 | 0.000014 | 0.002407 | 0.000490 | 5 | {'n_neighbors': 5} | 0.966667 | 1.000000 |

**18**-Eğitmiş olduğumuz ***Grid Search*** nesnemiz üzerindeki ana parametrelerimizi görüntüleyelim;

>>> grid_table_rank =
grid_table[[‘params’,’mean_test_score’,’std_test_score’,’rank_test_score’]].
loc[grid_table[‘rank_test_score’]==1].sort_values(by=’std_test_score’, ascending=True)
grid_table_rank

```
"""GRID NESNEMİZ ÜZERİNDEKİ ANA ANALİZ DEĞERLERİMİZİ GÖRÜNTÜLEYELİM."""
grid_table_rank = grid_table[['params','mean_test_score','std_test_score','rank_test_score']].loc[grid_table['rank_test_score']==
grid_table_rank
```

| | params | mean_test_score | std_test_score | rank_test_score |
|---|---|---|---|---|
| 5 | {'n_neighbors': 6} | 0.98 | 0.016330 | 1 |
| 6 | {'n_neighbors': 7} | 0.98 | 0.016330 | 1 |
| 9 | {'n_neighbors': 10} | 0.98 | 0.026667 | 1 |
| 10 | {'n_neighbors': 11} | 0.98 | 0.026667 | 1 |
| 11 | {'n_neighbors': 12} | 0.98 | 0.026667 | 1 |

**19**-***Grid Search*** nesnemiz üzerinden doğruluk değeri ve k parametre değerlerini görüntüleyelim;

>>> print(“{} doğrulundaki en iyi k = {}”.format(grid.best_score_,grid.best_params_))

```
"""DOĞRULUK VE k DEĞERİNİ GÖRÜNTÜLEYELİM."""
print("{} doğrulundaki en iyi k = {}".format(grid.best_score_,grid.best_params_))
```

```
0.9800000000000001 doğrulundaki en iyi k = {'n_neighbors': 6}
```

**20-*Grid Search*** nesnemiz üzerinden çapraz doğrulama, standart sapma, en iyi sıra ve k parametre değerlerini görüntüleyelim;

```
>>> print("En iyi k değeri için sıra= {}, k = {}, en yüksek doğruluktaki çapraz doğrulama = {}
        ve en düşük standart sapma ={}".
          format(grid_table.at[grid.best_index_,'rank_test_score'],grid_table.at[grid.best_index_,'params'],
                 grid_table.at[grid.best_index_,'mean_test_score'],grid_table.at[grid.best_index_,'std_test_score']))
```

```
print("En iyi k değeri için sıra= {}, k = {}, en yüksek doğruluktaki çapraz doğrulama = {} ve en düşük standart sapma = {}".forma

En iyi k değeri için sıra= 1, k = {'n_neighbors': 6}, en yüksek doğruluktaki çapraz doğrulama = 0.9800000000000001 ve en düşük
standart sapma = 0.016329931618554516
```

**21-**Analizimiz üzerindeki en iyi sınıflandırıcı parametre değeri ise;

```
>>> print("En iyi sınıflandırıcı parametre değeri: {}".format(grid.best_estimator_))
```

```
print("En iyi sınıflandırıcı parametre değeri: {}".format(grid.best_estimator_))

En iyi sınıflandırıcı parametre değeri: KNeighborsClassifier(n_neighbors=6)
```

**21-**(15) adım üzerinde yer alan listemiz elemanlarımız (komşular) göz önüne alındığında doğruluk değerleri grafik üzerinde görselleştirilirse;

```
>>> graphic = grid.cv_results_['mean_test_score']
    graphic
    plt.figure(figsize=(10,5))
    plt.plot(k_list,graphic,color='navy',linestyle='dashed',marker='o')
    plt.xlabel('K Komşu Numarası', fontdict={'fontsize': 15})
    plt.ylabel('Doğruluk', fontdict={'fontsize': 15})
    plt.title('K Değeri Komşu Bazlı Doğruluk Grafiği', fontdict={'fontsize': 30})
    plt.xticks(range(0,31,3),)
    plt.show()
```

```
graphic = grid.cv_results_['mean_test_score']
graphic

plt.figure(figsize=(10,5))
plt.plot(k_list,graphic,color='navy',linestyle='dashed',marker='o')
plt.xlabel('K Komşu Numarası', fontdict={'fontsize': 15})
plt.ylabel('Doğruluk', fontdict={'fontsize': 15})
plt.title('K Değeri Komşu Bazlı Doğruluk Grafiği', fontdict={'fontsize': 30})
plt.xticks(range(0,31,3),)
plt.show()
```


En Yakın Komşu algoritmasına yönelik basit veri seti üzerinden bir analiz gerçekleştirdik.
Mevcut algoritma kompleks ve bağlantısı yüksek olan veriler üzerinde çokça tercih edilmektedir.
Elbette ki avantaj ve dezavantajlar yer almaktadır.

## Günlük Hayatta Algoritma Uygulama Alanları

- Bankacılık Sektörü

-Kredi ve Risk Analizleri
-İşletme İflas Tahminlemeleri
-Borsa Analizleri (Alım-Satım İşlemleri)

- Spesifik Problemler

-Ekolojik Eylem Analizleri
-Biyolojik Analizler
-Kompleks Metin Madenciliği Analizleri
-Farklılaşmış Hastalık Analizleri (İleri Seviye Kanser/Tümör)
-Benzerliğe Dayalı Kayıp Veri Analizleri (Kompleks Seviye)

en çok tercih edilen alanlar veya problemler olarak dile getirilmektedir.

## Avantaj Dezavantaj

En Yakın Komşu algoritması avantajları olarak;

1-Eğitim işlemlerinin diğer algoritmalara nazaran daha kolay olması,
(kimi zaman olmaması, ki yok (algoritmada))

2-Süreçlerin ve analizlerin analitik/sayısal olarak takip edilebilir olması,

3-Kompleks ya da karmaşık (Gürültülü) eğitim verilerine karşı etkili olması,

4-Uyarlanabilirliğinin (uygulanabilirliği) kolay olması

dile getirilebilir.

En Yakın Komşu algoritması dezavantajları olarak;

1-İşlem hacmi ve işlem adımı fazla olduğundan dolayı yüksek donanıma ihtiyaç duymaktadır,
(maliyet)

2-Yüksek hacimli verilere karşı dirençli olsa da adım ve işlem sayısı fazla olduğundan dolayı zaman almaktadır,

3-Performans adına uygun algoritmanın bulunması kimi zaman dilimlerinde uzun sürmektedir,
(Uzaklık denklemi, parametreler vb.)

dile getirilebilir.