The k-Nearest Neighbors (k-NN) Algorithm

The k-Nearest Neighbors (k-NN) algorithm is a simple, intuitive, and widely used method for both classification and regression tasks in machine learning.

Overview of k-Nearest Neighbors (k-NN)

How k-NN Works

Data Preparation: Gather and preprocess your dataset. Ensure features are normalized or scaled if they vary widely in range.

Choose k: Decide on the number of nearest neighbors (k) to consider.

Compute Distance: Calculate the distance between the query point (the point for which you want to make a prediction) and all points in the training set. Common distance metrics include **Euclidean**, Manhattan, and Minkowski distances.

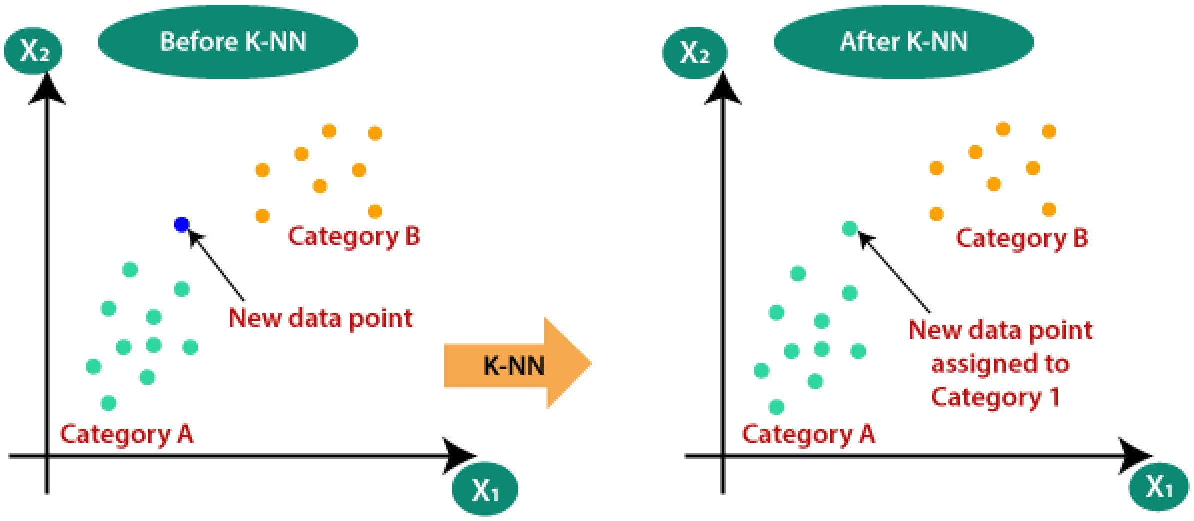
Identify Neighbors: Select the k points in the training set that are closest to the query point based on the computed distance.

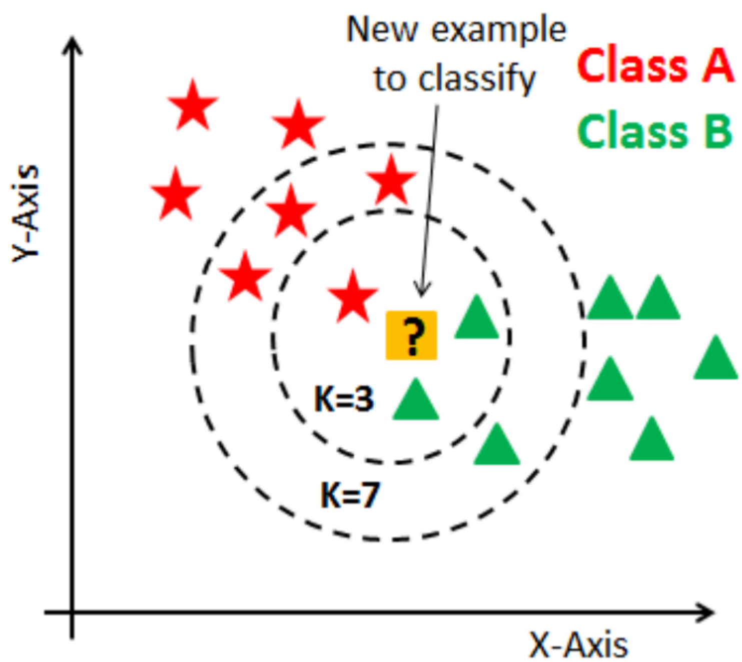
Make a Prediction: The query point is assigned to the class most common among its k nearest neighbors.

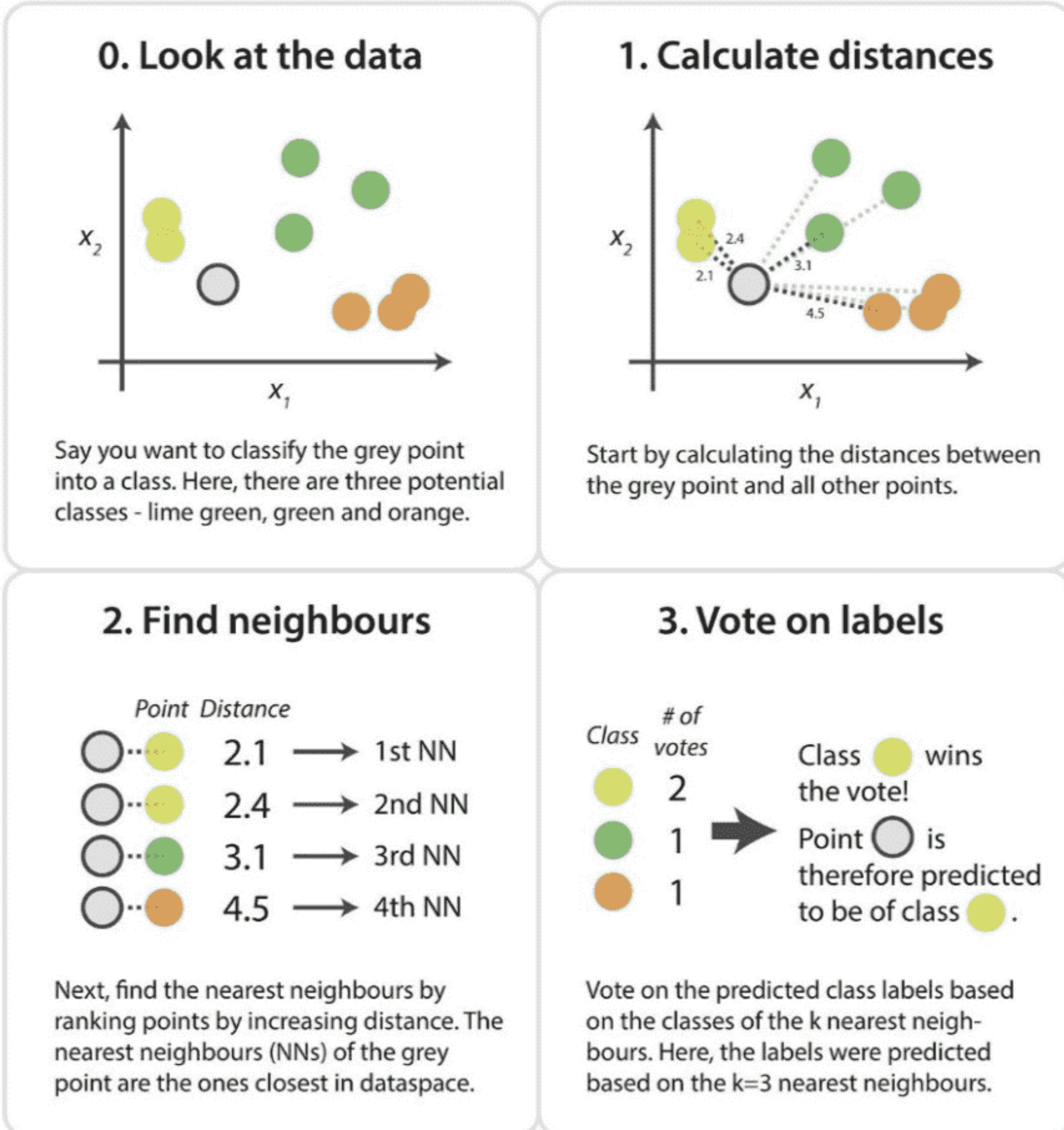
How to choose the value of k for KNN Algorithm?

The value of k is very crucial in the KNN algorithm to define the number of neighbors in the algorithm. The value of k in the k-nearest neighbors (k-NN) algorithm should be chosen based on the input data.

If the input data has more outliers or noise, a higher value of k would be better. It is recommended to choose an odd value for k to avoid ties in classification. Cross-validation methods can help in selecting the best k value for the given dataset.


Before K-NN
X₂
Category B
New data point
Category A
X₁
K-NN
After K-NN
X₂
Category B
New data point
assigned to
Category 1
Category A
X₁


New example
to classify
Class A
Class B
Y-Axis
K=3
K=7
X-Axis



Euclidean Distance Equation

$$\text{Euclidean distance} = \sqrt{(f_{11} - f_{12})^2 + (f_{21} - f_{22})^2}$$

where f_{11} = value of feature f_1 for data element d_1

f_{12} = value of feature f_1 for data element d_2

f_{21} = value of feature f_2 for data element d_1

f_{22} = value of feature f_2 for data element d_2

Algorithm Steps

For a given test instance x:

1. Calculate the distance between x and all points in the training set.
2. Sort the training points by distance from x.
3. Select the k nearest neighbors.
4. For classification, perform a majority vote among the k neighbors. For regression, average the k neighbors' values.

Choosing the Right k

Selecting an appropriate value for k is crucial:

Small k: Can be noisy and lead to overfitting.

Large k: May smooth out the predictions too much and underfit the data.

A common approach is to use cross-validation to determine the optimal k value.

Summary

k-Nearest Neighbors is a straightforward and versatile algorithm useful for both classification and regression. Its ease of implementation and lack of training phase make it an attractive choice for many problems, though its performance can be impacted by the size and dimensionality of the dataset. When using k-NN, careful consideration must be given to the choice of k and the distance metric to ensure optimal results.

diabetes.csv

In [1]:

```
#import libraries
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

In [2]:
```
data = pd.read_csv("diabetes.csv")
data
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33
...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63
764	2	122	70	27	0	36.8	0.340	27
765	5	121	72	23	112	26.2	0.245	30
766	1	126	60	0	0	30.1	0.349	47
767	1	93	70	31	0	30.4	0.315	23

768 rows × 9 columns

In [3]:
```
data.shape
```

Out[3]:
```
(768, 9)
```

In [4]:
```
data.isna().sum()
```

Out[4]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
```

```
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [6]:
```
#Segregating predictor variables
x = data.iloc[:, 0:8]
x
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33
...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63
764	2	122	70	27	0	36.8	0.340	27
765	5	121	72	23	112	26.2	0.245	30
766	1	126	60	0	0	30.1	0.349	47
767	1	93	70	31	0	30.4	0.315	23

768 rows × 8 columns

In [7]:
```
#Segregating the target/class variable
y = data['Outcome']
y
```

Out[7]:
```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

In [8]:
```
#split into training and test datasets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2,random_state=
```

In [9]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(614, 8)
(154, 8)
```

```
(614,)
(154,)
```

In [22]:

```
#kNN Classifier with k=27 means 27 closest neighbours are considered.
nn = KNeighborsClassifier(n_neighbors=15)
```

In [23]:

```
#Train the classifier with the training data
model = nn.fit(x_train, y_train)
```

In [24]:

```
prediction = model.predict(x_test)
```

In [25]:

```
prediction
```

Out[25]:

```
array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0],
      dtype=int64)
```

In [26]:

```
diff=pd.DataFrame({"Actual":y_test,"Prediction":prediction})
diff
```

Out[26]:

	Actual	Prediction
285	0	1
101	0	0
581	0	0
352	0	0
726	0	0
...	...	...
563	0	0
318	0	0
154	1	1
684	0	0
643	0	0

154 rows × 2 columns

In [17]:

```
#To Store the data of above dataframe to a csv file
diff.to_csv('diabetes_data.csv')
```

In [16]:

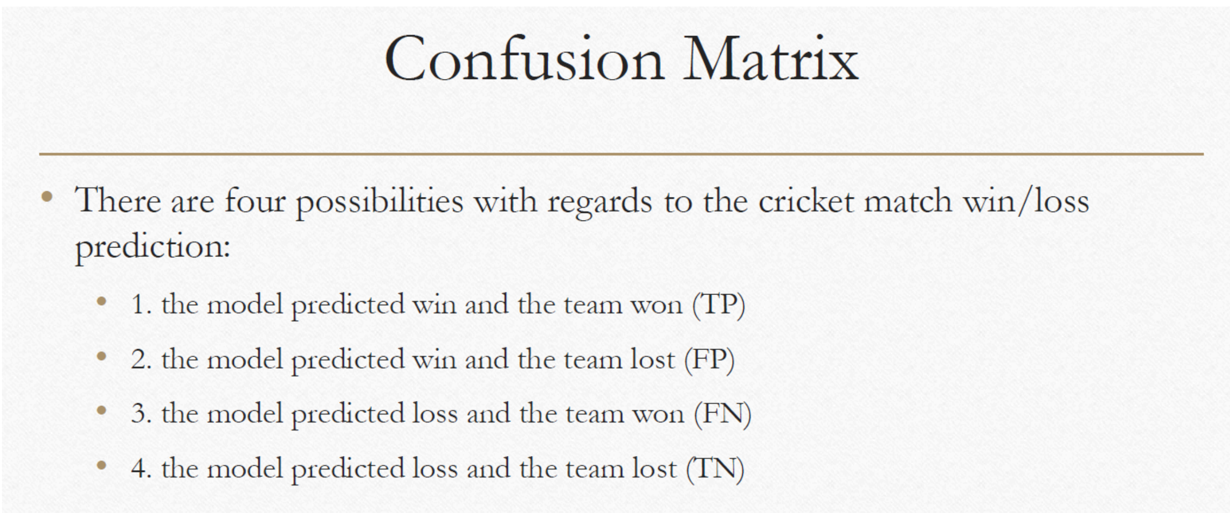
```
x_test
```

Out[16]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
285	7	136	74	26	135	26.0	0.647	51
101	1	151	60	0	0	26.1	0.179	22

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
581	6	109	60	27	0	25.0	0.206	27
352	3	61	82	28	0	34.4	0.243	46
726	1	116	78	29	180	36.1	0.496	25
...	...	...	...	...	...	...	...	...
563	6	99	60	19	54	26.9	0.497	32
318	3	115	66	39	140	38.1	0.150	28
154	8	188	78	0	0	47.9	0.137	43
684	5	136	82	0	0	0.0	0.640	69
643	4	90	0	0	0	28.0	0.610	31

154 rows × 8 columns



In [27]:
```
#Metric Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, prediction)
cm
```

Out[27]:
```
array([[89, 10],
       [23, 32]], dtype=int64)
```

In [146…
```
TN = cm[0][0]
FP = cm[0][1]
FN = cm[1][0]
TP = cm[1][1]
print(TP, FN, TN, FP)
```

```
28 27 88 11
```

Model Accuracy

$$\text{Model accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

	ACTUAL WIN	ACTUAL LOSS
Predicted Win	85	4
Predicted Loss	2	9

In context of the above confusion matrix, total count of TPs = 85, count of FPs = 4, count of FNs = 2 and count of TNs = 9.

$$\therefore \text{Model accuracy} = \frac{TP + TN}{TP + FP + FN + TN} = \frac{85 + 9}{85 + 4 + 2 + 9} = \frac{94}{100} = 94\%$$

Model Accuracy/Accuracy Score

In [147…

```
Model_Accuracy=(TP+TN)/(TP+TN+FN+FP)
print("Accuracy Score:",Model_Accuracy)
```

```
Accuracy Score: 0.7532467532467533
```

In [148…

```
from sklearn.metrics import accuracy_score
Accuracy=accuracy_score(y_test,prediction)
Accuracy
```

Out[148…

```
0.7532467532467533
```

Error Rate

$$\text{Error rate} = \frac{FP + FN}{TP + FP + FN + TN} = \frac{4+2}{85+4+2+9} = \frac{6}{100} = 6\%$$
$$= 1\text{–Model accuracy}$$

Error Rate

In [149…

```
Error_Rate=1-Model_Accuracy
print("Error Rate:", Error_Rate)
```

```
Error Rate: 0.24675324675324672
```

Sensitivity

- The sensitivity of a model measures the proportion of TP examples or positive cases which were correctly classified.

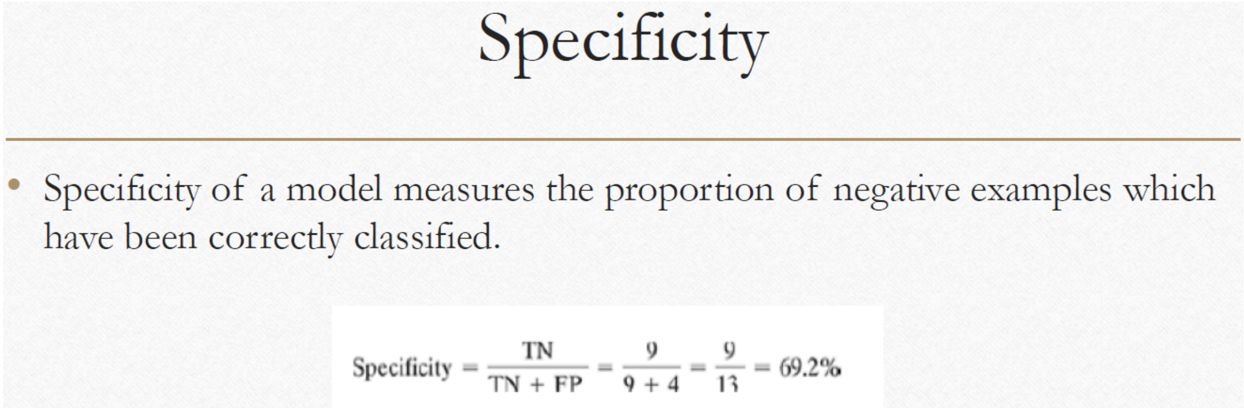
$$\text{Sensitivity} = \frac{TP}{TP + FN} = \frac{85}{85 + 2} = \frac{85}{87} = 97.7\%$$

Sensitivity

In [150…

```
Sensitivity= TP / (TP + FN)
print("Sensitivity:",Sensitivity)
```

```
Sensitivity: 0.509090909090909
```



Specificity

In [151…

```
Specificity= TN / (TN + FP)
print("Specificity:",Specificity)
```

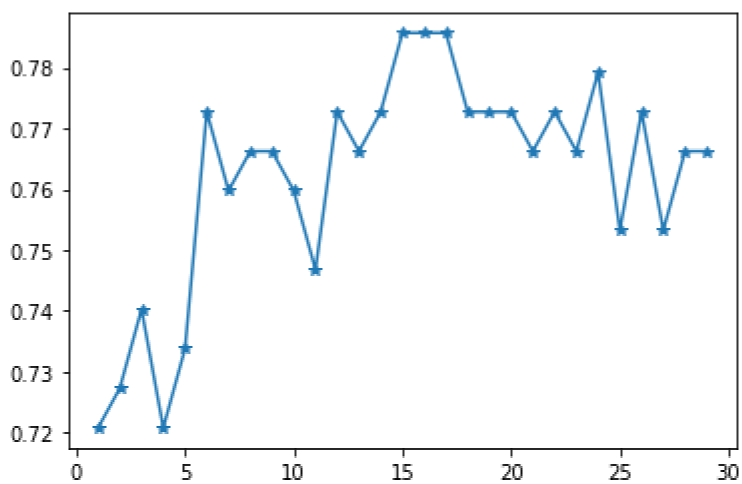
```
Specificity: 0.8888888888888888
```

To find the best value of k for highest accuracy_score.

In [28]:

```
k=[]
for i in range (1,30):
    nn = KNeighborsClassifier(n_neighbors=i)
    model = nn.fit(x_train, y_train)
    prediction = model.predict(x_test)
    from sklearn.metrics import accuracy_score
    k.append(accuracy_score(y_test,prediction))

import matplotlib.pyplot as plt
plt.plot(range(1,30),k,marker="*")
plt.show()
print(k)
```




```
[0.7207792207792207, 0.7272727272727273, 0.7402597402597403, 0.7207792207792207, 0.73376
62337662337, 0.7727272727272727, 0.7597402597402597, 0.7662337662337663, 0.7662337662337
663, 0.7597402597402597, 0.7467532467532467, 0.7727272727272727, 0.7662337662337663, 0.7
727272727272727, 0.7857142857142857, 0.7857142857142857, 0.7857142857142857, 0.772727272
7272727, 0.7727272727272727, 0.7727272727272727, 0.7662337662337663, 0.7727272727272727,
0.7662337662337663, 0.7792207792207793, 0.7532467532467533, 0.7727272727272727, 0.753246
7532467533, 0.7662337662337663, 0.7662337662337663]
```

tshirt.csv

In [153…

```
#import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

In [154…

```
df=pd.read_csv('tshirt.csv')
df
```

Out[154…

	Height	Weight	Size
0	158	58	M
1	158	59	M
2	158	63	M
3	160	59	M
4	160	60	M
5	163	60	M
6	163	61	M
7	160	64	L
8	163	64	L
9	165	61	L
10	165	62	L
11	165	65	L
12	168	62	L
13	168	63	L
14	168	66	L
15	170	63	L
16	170	64	L
17	170	68	L

In [155…

```
#Segregating predictor variables
x = df.iloc[:, 0:2]
y = df.iloc[:,2]
```

In [156…

```
#Alternate Method to take x & y (Segregating predictor variables)
x=df[['Height','Weight']]
y=df['Size']
```

In [157…

```
x.shape
```

Out[157…

```
(18, 2)
```

In [158…

```
y.shape
```

Out[158…

```
(18,)
```

In [159…

```
#split into training and test datasets
# x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2,random_stat
```

In [160…

```
# print(x_train.shape)
# print(x_test.shape)
# # print(y_train.shape)
# print(y_test.shape)
```

In [161…

```
#kNN Classifier with k=9 means 9 closest neighbours are considered
nn = KNeighborsClassifier(n_neighbors=9)
```

In [162…

```
#Train the classifier with the training data
model = nn.fit(x, y)
```

In [163…

```
prediction = model.predict(x)
```

In [164…

```
prediction
```

Out[164…

```
array(['M', 'M', 'M', 'M', 'M', 'M', 'L', 'M', 'L', 'L', 'L', 'L', 'L',
       'L', 'L', 'L', 'L', 'L'], dtype=object)
```

In [165…

```
diff=pd.DataFrame({'Actual':y,"Predicted":prediction})
diff
```

Out[165…

	Actual	Predicted
0	M	M
1	M	M
2	M	M
3	M	M
4	M	M
5	M	M
6	M	L
7	L	M
8	L	L
9	L	L
10	L	L
11	L	L
12	L	L
13	L	L
14	L	L
15	L	L

	Actual	Predicted
16	L	L
17	L	L

In [166…

```
#Metric Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y, prediction)
cm
```

Out[166…

```
array([[10,  1],
       [ 1,  6]], dtype=int64)
```

In [167…

```
TN=cm[0][0]
TP=cm[1][1]
FN=cm[1][0]
FP=cm[0][1]

print(TN,TP,FN,FP)
```

```
10 6 1 1
```

In [78]:

```
Model_Accuracy=(TP+TN)/(TP+TN+FN+FP)
print("Accuracy:",Model_Accuracy)
```

```
Accuracy: 0.8888888888888888
```

In [168…

```
from sklearn.metrics import accuracy_score
print("Accuracy Score: ",accuracy_score(y,prediction))
```

```
Accuracy Score:  0.8888888888888888
```

In [79]:

```
Error_Rate=1-Model_Accuracy
print("Error Rate:", Error_Rate)
```

```
Error Rate: 0.11111111111111116
```

In [80]:

```
Sensitivity= TP / (TP + FN)
print("Sensitivity:",Sensitivity)
```

```
Sensitivity: 0.8571428571428571
```

In [81]:

```
Specificity= TN / (TN + FP)
print("Specificity:",Specificity)
```

```
Specificity: 0.9090909090909091
```

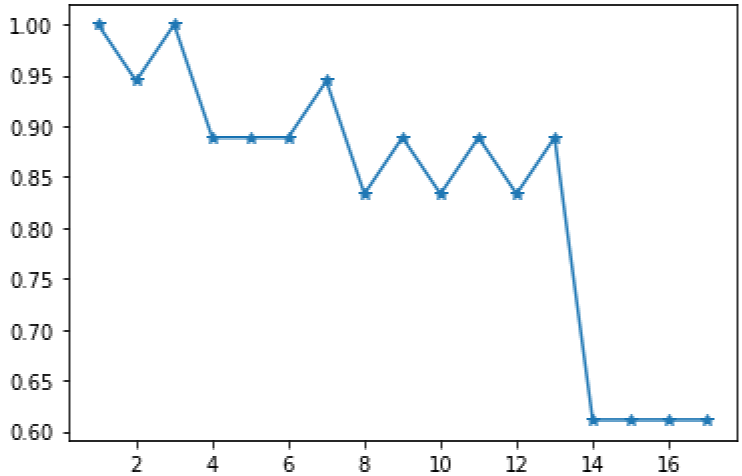
To find the best value of k for highest accuracy_score.

In [169…

```
k=[]
for i in range (1,18):
    nn = KNeighborsClassifier(n_neighbors=i)
    model = nn.fit(x,y)
    prediction = model.predict(x)
    from sklearn.metrics import accuracy_score
    k.append(accuracy_score(y,prediction))

import matplotlib.pyplot as plt
plt.plot(range(1,18),k,marker="*")
```

```
plt.show()
print(k)
```




```
[1.0, 0.9444444444444444, 1.0, 0.8888888888888888, 0.8888888888888888, 0.888888888888888
8, 0.9444444444444444, 0.8333333333333334, 0.8888888888888888, 0.8333333333333334, 0.888
8888888888888, 0.8333333333333334, 0.8888888888888888, 0.6111111111111112, 0.61111111111
11112, 0.6111111111111112, 0.6111111111111112]
```

AptitudeCommunication.csv

In [171…

```
import pandas as pd
df=pd.read_csv('AptitudeCommunication.csv')
df
```

Out[171…

	Name	Aptitude	Communication	Class
0	Karuna	2	5.0	Speaker
1	Bhavan	2	6.0	Speaker
2	Gaurav	7	6.0	Leader
3	Parul	7	2.5	Intel
4	Dinesh	8	6.0	Leader
5	Jani	4	7.0	Speaker
6	Bobby	5	3.0	Intel
7	Parimal	3	5.5	Speaker
8	Govind	8	3.0	Intel
9	Sushant	6	5.5	Leader
10	Gauri	6	4.0	Intel
11	Bharat	6	7.0	Leader
12	Rajvi	6	2.0	Intel
13	Pradip	9	7.0	Leader

In [172…

```
x=df[['Aptitude','Communication']]
y=df['Class']
```

In [173…
```
x.shape
```
Out[173…
```
(14, 2)
```
In [174…
```
y.shape
```
Out[174…
```
(14,)
```
In [180…
```
from sklearn.neighbors import KNeighborsClassifier

nn=KNeighborsClassifier(n_neighbors=5)
model=nn.fit(x,y)
y_pred=model.predict(x)
y_pred
```
Out[180…
```
array(['Speaker', 'Speaker', 'Leader', 'Intel', 'Leader', 'Speaker',
       'Intel', 'Speaker', 'Intel', 'Leader', 'Intel', 'Leader', 'Intel',
       'Leader'], dtype=object)
```
In [181…
```
diff=pd.DataFrame({"Actual":y,"Predicted":y_pred})
diff
```
Out[181…

	Actual	Predicted
0	Speaker	Speaker
1	Speaker	Speaker
2	Leader	Leader
3	Intel	Intel
4	Leader	Leader
5	Speaker	Speaker
6	Intel	Intel
7	Speaker	Speaker
8	Intel	Intel
9	Leader	Leader
10	Intel	Intel
11	Leader	Leader
12	Intel	Intel
13	Leader	Leader

In [182…
```
#Prediction for specific values of Aptitude & Communication
prediction=model.predict([[5,4.5]])
prediction
```
Out[182…
```
array(['Intel'], dtype=object)
```
In [183…
```
#Confusion Matrix
from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(y, y_pred)
cm
```

Out[183…

```
array([[5, 0, 0],
       [0, 5, 0],
       [0, 0, 4]], dtype=int64)
```

In [188…

```
from sklearn.metrics import accuracy_score
print("accuracy_score:", accuracy_score(y,y_pred))
```

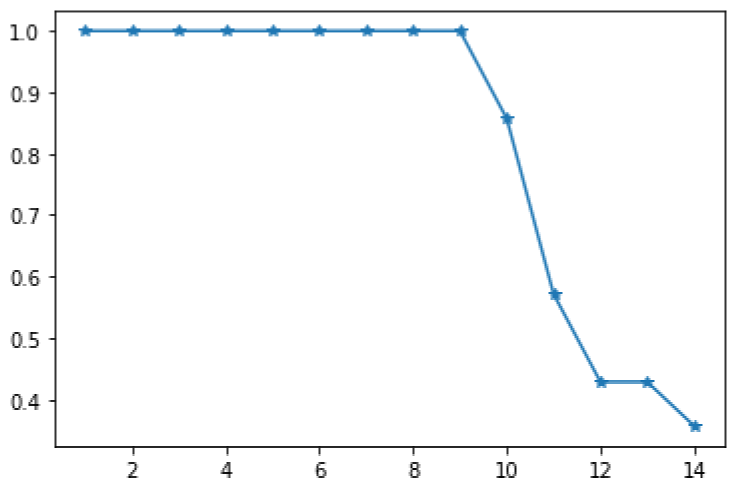
```
accuracy_score: 1.0
```

To find the best value of k for highest accuracy_score

In [190…

```
k=[]
for i in range (1,15):
    nn = KNeighborsClassifier(n_neighbors=i)
    model = nn.fit(x,y)
    prediction = model.predict(x)
    from sklearn.metrics import accuracy_score
    k.append(accuracy_score(y,prediction))

import matplotlib.pyplot as plt
plt.plot(range(1,15),k,marker="*")
plt.show()
print(k)
```




```
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.8571428571428571, 0.5714285714285714, 0.
42857142857142855, 0.42857142857142855, 0.35714285714285715]
```

In [ ]:

In [ ]: