KNN Machine Learning Algorithm

CPC 152 Project

Import the library and Understand the data

In [1]:

```
import numpy as np
import numpy.random as npr
import pandas as pd
```

In [61]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
```

In [62]:

```
data=pd.read_csv('C:/Users/User/Desktop/USM CS/SEMESTER 2/CPC152 - Foundations and Prog
data
```

Out[62]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

303 rows × 14 columns

In [63]:

```
data.head(10)
```

Out[63]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
5	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
6	56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
7	44	1	1	120	263	0	1	173	0	0.0	2	0	3	1
8	52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
9	57	1	2	150	168	0	1	174	0	1.6	2	0	2	1

In [64]:

```
data.tail(10)
```

Out[64]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
293	67	1	2	152	212	0	0	150	0	0.8	1	0	3	0
294	44	1	0	120	169	0	1	144	1	2.8	0	0	1	0
295	63	1	0	140	187	0	0	144	1	4.0	2	2	3	0
296	63	0	0	124	197	0	1	136	1	0.0	1	0	2	0
297	59	1	0	164	176	1	0	90	0	1.0	1	2	1	0
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

In [65]:

```
len(data)
```

Out[65]:

```
303
```

In [66]:

```
data.describe()
```

Out[66]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000

	age	sex	cp	trestbps	chol	fbs	restecg	thalach
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000

In [67]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [68]:

```
data.nunique()
```

Out[68]:

```
age          41
sex           2
cp            4
trestbps     49
chol        152
fbs           2
restecg       3
thalach      91
exang         2
oldpeak      40
slope         3
ca            5
thal          4
target        2
dtype: int64
```

In [69]:

```
 #data.dtypes()
```

In [70]:

```
#Clean the data set
```

In [71]:

```
data.apply(lambda x: sum(x.isnull()),axis=0)
```

Out[71]:

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

Split the dataframe into testdata and traindata

In [72]:

```
x=data.iloc[:,0:13]
y=data.iloc[:,13]
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=0,test_size=0.2)
```

In [73]:

```
x_train
```

Out[73]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
74	43	0	2	122	213	0	1	165	0	0.2	1	0	2
153	66	0	2	146	278	0	0	152	0	0.0	1	1	2
64	58	1	2	140	211	1	0	165	0	0.0	2	0	2
296	63	0	0	124	197	0	1	136	1	0.0	1	0	2
287	57	1	1	154	232	0	0	164	0	0.0	2	1	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
251	43	1	0	132	247	1	0	143	1	0.1	1	4	3
192	54	1	0	120	188	0	1	113	0	1.4	1	1	3
117	56	1	3	120	193	0	0	162	0	1.9	1	0	3
47	47	1	2	138	257	0	0	156	0	0.0	2	0	2
172	58	1	1	120	284	0	0	160	0	1.8	1	0	2

242 rows × 13 columns

In [74]:

```
x_test
```

Out[74]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
225	70	1	0	145	174	0	1	125	1	2.6	0	0	3
152	64	1	3	170	227	0	0	155	0	0.6	1	0	3
228	59	1	3	170	288	0	0	159	0	0.2	1	0	3
201	60	1	0	125	258	0	0	141	1	2.8	1	1	3
52	62	1	2	130	231	0	1	146	0	1.8	1	3	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...
146	44	0	2	118	242	0	1	149	0	0.3	1	1	2
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2
26	59	1	2	150	212	1	1	157	0	1.6	2	0	2
108	50	0	1	120	244	0	1	162	0	1.1	2	0	2
89	58	0	0	100	248	0	0	122	0	1.0	1	0	2

61 rows × 13 columns

In [75]:

```
y_train
```

Out[75]:

```
74     1
153    1
64     1
296    0
287    0
      ..
251    0
192    0
117    1
47     1
172    0
Name: target, Length: 242, dtype: int64
```

In [76]:

```
y_test
```

Out[76]:

```
225    0
152    1
228    0
201    0
52     1
      ..
146    1
302    0
26     1
108    1
89     1
Name: target, Length: 61, dtype: int64
```

K-Nearest Neighbour Algorithms

scaling of the data

In [77]:

```
#Feature Scaling
sc_X=StandardScaler()
x_train=sc_X.fit_transform(x_train)
x_test=sc_X.fit_transform(x_test)
```

Experiment 1

In [78]:

```
#Detemine the K value for the KNN
import math
math.sqrt(len(y_train))
#Therefore, the k value is 15, which is an odd number
```

Out[78]:

```
15.556349186104045
```

In [79]:

```
#Detemine the K value for the KNN
import math
math.sqrt(len(y_test))
#Therefore, the k value is 7
```

Out[79]:

```
7.810249675906654
```

In [80]:

```
#Define the model: Init K-NN
from sklearn.neighbors import KNeighborsClassifier
classifier=KNeighborsClassifier(n_neighbors=11,p=2,metric='euclidean')
classifier.fit(x_train,y_train)
```

Out[80]:

```
KNeighborsClassifier(metric='euclidean', n_neighbors=11)
```

In [81]:

```
#Generate the prediction result
y_prediction=classifier.predict(x_test)
y_prediction
```

Out[81]:

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

In [82]:

```
y_test
```

Out[82]:

```
225    0
152    1
228    0
201    0
52     1
      ..
146    1
302    0
26     1
108    1
89     1
Name: target, Length: 61, dtype: int64
```

In [83]:

```
# Find out the confusion Matrics
cm=confusion_matrix(y_test,y_prediction)
cm
```

Out[83]:

```
array([[21,  6],
       [ 3, 31]], dtype=int64)
```

In [86]:

```
#Find out the accracy value
print ("Accuracy : ", accuracy_score(y_test,y_prediction))
print ("F1 score : ", f1_score(y_test,y_prediction))
print ("Recall   : ", recall_score(y_test,y_prediction))
print ("Precision: ", precision_score(y_test,y_prediction))
```

```
Accuracy :  0.8524590163934426
F1 score :  0.8732394366197184
Recall   :  0.9117647058823529
Precision:  0.8378378378378378
```

Experiment 2

The same algorithm is repeated by changing with the smaller k value

In [87]:

```
classifier=KNeighborsClassifier(n_neighbors=7,p=2,metric='euclidean')
classifier.fit(x_train,y_train)
```

Out[87]:

```
KNeighborsClassifier(metric='euclidean', n_neighbors=7)
```

In [88]:

```
y_prediction2=classifier.predict(x_test)
y_prediction2
```

Out[88]:

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

In [89]:

```
# Find out the confusion Matrics
cm=confusion_matrix(y_test,y_prediction2)
cm
```

Out[89]:

```
array([[20,  7],
       [ 4, 30]], dtype=int64)
```

In [90]:

```
#Find out the accracy value
print ("Accuracy : ", accuracy_score(y_test,y_prediction2))
print ("F1 score : ", f1_score(y_test,y_prediction2))
print ("Recall   : ", recall_score(y_test,y_prediction2))
print ("Precision: ", precision_score(y_test,y_prediction2))
```

```
Accuracy :  0.819672131147541
F1 score :  0.8450704225352113
Recall   :  0.8823529411764706
Precision:  0.8108108108108109
```

Experiment 3

The same algorithm is repeated by changing with the larger k value

In [104…

```
classifier=KNeighborsClassifier(n_neighbors=20,p=2,metric='euclidean')
classifier.fit(x_train,y_train)
```

Out[104…

```
KNeighborsClassifier(metric='euclidean', n_neighbors=20)
```

In [105…

```
y_prediction3=classifier.predict(x_test)
y_prediction3
```

Out[105…

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

In [106…

```
# Find out the confusion Matrics
cm=confusion_matrix(y_test,y_prediction3)
cm
```

Out[106…

```
array([[19,  8],
       [ 2, 32]], dtype=int64)
```

In [107…

```
#Find out the accracy value
print ("Accuracy : ", accuracy_score(y_test,y_prediction3))
print ("F1 score : ", f1_score(y_test,y_prediction3))
print ("Recall   : ", recall_score(y_test,y_prediction3))
print ("Precision: ", precision_score(y_test,y_prediction3))
```

```
Accuracy :  0.8360655737704918
F1 score :  0.8648648648648648
Recall   :  0.9411764705882353
Precision:  0.8
```

Experiment 4

The same algorithm is repeated by changing with the different k value

In [109…

```
#Determine the optimum value of k
math.sqrt(len(data))
```

Out[109…

```
17.406895185529212
```

In [108…

```
classifier=KNeighborsClassifier(n_neighbors=17,p=2,metric='euclidean')
classifier.fit(x_train,y_train)
```

Out[108…

```
KNeighborsClassifier(metric='euclidean', n_neighbors=17)
```

In [110…

```
y_prediction4=classifier.predict(x_test)
y_prediction4
```

Out[110…

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

In [111…

```
# Find out the confusion Matrics
cm=confusion_matrix(y_test,y_prediction4)
cm
```

Out[111…

```
array([[19,  8],
       [ 2, 32]], dtype=int64)
```

In [112…

```
#Find out the accracy value
print ("Accuracy : ", accuracy_score(y_test,y_prediction4))
print ("F1 score : ", f1_score(y_test,y_prediction4))
print ("Recall   : ", recall_score(y_test,y_prediction4))
print ("Precision: ", precision_score(y_test,y_prediction4))
```

```
Accuracy :  0.8360655737704918
F1 score :  0.8648648648648648
Recall   :  0.9411764705882353
Precision:  0.8
```

Experiment 5

The same algorithm is repeated by changing with the different k value

In [131…

```
classifier=KNeighborsClassifier(n_neighbors=12,p=2,metric='euclidean')
classifier.fit(x_train,y_train)
```

Out[131…

```
KNeighborsClassifier(metric='euclidean', n_neighbors=12)
```

In [132…

```
y_prediction5=classifier.predict(x_test)
y_prediction5
```

Out[132…

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

In [133…

```
# Find out the confusion Matrics
cm=confusion_matrix(y_test,y_prediction5)
cm
```

Out[133…

```
array([[22,  5],
       [ 3, 31]], dtype=int64)
```

In [138…

```
#Find out the accracy value
print ("Accuracy : ", accuracy_score(y_test,y_prediction5))
print ("F1 score : ", f1_score(y_test,y_prediction5))
print ("Recall   : ", recall_score(y_test,y_prediction5))
print ("Precision: ", precision_score(y_test,y_prediction5))
```

```
Accuracy :  0.8688524590163934
F1 score :  0.8857142857142858
Recall   :  0.9117647058823529
Precision:  0.8611111111111112
```

Experiment 6

The same algorithm is repeated by changing with the different k value

In [135…

```
classifier=KNeighborsClassifier(n_neighbors=13,p=2,metric='euclidean')
classifier.fit(x_train,y_train)
```

Out[135…

```
KNeighborsClassifier(metric='euclidean', n_neighbors=13)
```

In [136…

```
y_prediction6=classifier.predict(x_test)
y_prediction6
```

Out[136…

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

In [137…

```
# Find out the confusion Matrics
cm=confusion_matrix(y_test,y_prediction6)
cm
```

Out[137…

```
array([[20,  7],
       [ 2, 32]], dtype=int64)
```

In [139…

```
#Find out the accracy value
print ("Accuracy : ", accuracy_score(y_test,y_prediction6))
print ("F1 score : ", f1_score(y_test,y_prediction6))
print ("Recall   : ", recall_score(y_test,y_prediction6))
print ("Precision: ", precision_score(y_test,y_prediction6))
```

```
Accuracy :  0.8524590163934426
F1 score :  0.8767123287671232
Recall   :  0.9411764705882353
Precision:  0.8205128205128205
```

Hence, the conclusion is the optimum k value for the data set is 12