knn(scikit)(20bkt0039)

importing libraries

In [13]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
```

loading dataset

In [14]:

```
dataset=pd.read_csv("D:\Downloads\TicTacToeEndgame.csv")
dataset.mean=dataset
```

In [15]:

```
dataset.head()
```

Out[15]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
0	x	x	x	x	o	o	x	o	o	positive
1	x	x	x	x	o	o	o	x	o	positive
2	x	x	x	x	o	o	o	o	x	positive
3	x	x	x	x	o	o	o	b	b	positive
4	x	x	x	x	o	o	b	o	b	positive

data preprocessing

In [16]:

```
# total number of observations
n= dataset.shape[0]
# number of features
n_features=dataset.shape[1]-1

dataset.replace(to_replace=['x'], value=1, inplace=True)
dataset.replace(to_replace=['o'], value=2, inplace=True)
dataset.replace(to_replace=['b'], value=3, inplace=True)
dataset["V10"].replace(to_replace=['positive'], value=1, inplace=True)
dataset["V10"].replace(to_replace=['negative'], value=0, inplace=True)
# number of succesfull outcomes
success=dataset[dataset['V10']==1].shape[0]
# number of losses
loss=dataset[dataset['V10']==0].shape[0]
print("total number of patients: {} ".format(n))
print("number of features: {}".format(n_features))
print("number of wins: {}".format(success))
print("number of loss: {}".format(loss))
dataset.head()
```

```
total number of patients: 958
number of features: 9
number of wins: 626
number of loss: 332
```

Out[16]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
0	1	1	1	1	2	2	1	2	2	1
1	1	1	1	1	2	2	2	1	2	1
2	1	1	1	1	2	2	2	2	1	1
3	1	1	1	1	2	2	2	3	3	1
4	1	1	1	1	2	2	3	2	3	1

In [17]:

```
# extracting feature columns
features_cols=list(dataset[0:9])

#show the list of columns
print("features columns:\n{}".format(features_cols))
```

```
features columns:
['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10']
```

In [18]:

```
#separate the data into features data and target data
x=dataset[features_cols]
y=dataset['V10'].values

print("feature values:")
x.head
```

```
feature values:
```

Out[18]:

```
<bound method NDFrame.head of      V1  V2  V3  V4  V5  V6  V7  V8  V9  V10
0     1   1   1   1   2   2   1   2   2    1
1     1   1   1   1   2   2   2   1   2    1
2     1   1   1   1   2   2   2   2   1    1
3     1   1   1   1   2   2   2   3   3    1
4     1   1   1   1   2   2   3   2   3    1
..   ..  ..  ..  ..  ..  ..  ..  ..  ..  ...
953   2   1   1   1   2   2   2   1   1    0
954   2   1   2   1   1   2   1   2   1    0
955   2   1   2   1   2   1   1   2   1    0
956   2   1   2   2   1   1   1   2   1    0
957   2   2   1   1   1   2   2   1   1    0

[958 rows x 10 columns]>
```

In [19]:

```
#split the data set into training and testing data
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=5)
print(x_train)
```

```
     V1  V2  V3  V4  V5  V6  V7  V8  V9  V10
120   1   2   1   1   3   3   1   2   2    1
370   2   2   1   1   1   1   3   3   2    1
89    1   1   2   2   1   2   2   1   1    1
500   3   1   2   1   1   3   2   1   2    1
932   3   3   1   2   2   2   3   1   1    0
..   ..  ..  ..  ..  ..  ..  ..  ..  ..  ...
400   2   2   1   3   1   3   1   3   3    1
118   1   2   1   1   3   2   1   2   3    1
701   1   2   3   2   2   1   1   2   1    0
206   1   2   3   1   3   3   1   3   2    1
867   2   3   2   1   1   2   1   1   2    0

[670 rows x 10 columns]
```

In [20]:

```
#dimension of training and testing data
print(x_train.shape)
print(x_test.shape)
```

```
(670, 10)
(288, 10)
```

In [21]:

```
# Normalization Step

from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()

scaler.fit(x_train)

x_train = scaler.transform(x_train)

print("----After Z-score Normalization on X_train--- --")

print (x_train)

scaler.fit(x_test)
x_test= scaler.transform(x_test)
print("----After Z-score Normalization on X_test-------")
print (x_test)
```

```
----After Z-score Normalization on X_train--- --
[[-0.95738078  0.18738292 -0.98501759 ...  0.17745201  0.28125644
   0.71106819]
 [ 0.33066164  0.18738292 -0.98501759 ...  1.44226953  0.28125644
   0.71106819]
 [-0.95738078 -1.06808266  0.31925471 ... -1.08736551 -1.00944093
   0.71106819]
 ...
 [-0.95738078  0.18738292  1.62352702 ...  0.17745201 -1.00944093
  -1.40633487]
 [-0.95738078  0.18738292  1.62352702 ...  1.44226953  0.28125644
   0.71106819]
 [ 0.33066164  1.4428485   0.31925471 ... -1.08736551  0.28125644
  -1.40633487]]
----After Z-score Normalization on X_test-------
[[ 0.1857525   1.36797819 -1.04777206 ...  1.36938234 -0.98890588
  -1.30061093]
 [-1.11904557 -1.12555167  0.21481519 ... -1.08019601  1.58851398
  -1.30061093]
 [ 1.49055057 -1.12555167  0.21481519 ... -1.08019601  1.58851398
  -1.30061093]
 ...
 [-1.11904557 -1.12555167 -1.04777206 ... -1.08019601  0.29980405
   0.76886944]
 [-1.11904557 -1.12555167  0.21481519 ...  1.36938234  0.29980405
  -1.30061093]
 [ 1.49055057 -1.12555167  0.21481519 ... -1.08019601  0.29980405
   0.76886944]]
```

knn

In [22]:

```
# Test the Model using  K Neighbors Classifier
#training and prediction through a K Neighbors Classifier

for k in range(3,19,2):
    knn= KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train, y_train) # Training
    y_predictions = knn.predict (x_test) # Testing
    # accuracy on x_test
    accuracy = accuracy_score (y_test,y_predictions,)
    print("Accuracy for K ="+str(k)+":", accuracy)
    # creating a confusion matrix
    cm = confusion_matrix(y_test,y_predictions)
    print(cm)
```

```
Accuracy for K =3: 0.9965277777777778
[[106   1]
 [  0 181]]
Accuracy for K =5: 0.9965277777777778
[[106   1]
 [  0 181]]
Accuracy for K =7: 0.9965277777777778
[[106   1]
 [  0 181]]
Accuracy for K =9: 0.9930555555555556
[[105   2]
 [  0 181]]
Accuracy for K =11: 1.0
[[107   0]
 [  0 181]]
Accuracy for K =13: 1.0
[[107   0]
 [  0 181]]
Accuracy for K =15: 1.0
[[107   0]
 [  0 181]]
Accuracy for K =17: 1.0
[[107   0]
 [  0 181]]
```