In [1]:

```
#KNN is a pseudo machine learning algorithm, as it doesn't contain any learnable parameters(weights, biases,
```

In [2]:

```
#KNN is a classifier -  Tells your test subject belongs to which class
```

In [3]:

```
#in knn k should be an  odd number
```

In [4]:

```
#we dont use knn now as there are better algorithms
```

In [5]:

```
import numpy as np
import matplotlib.pyplot as plt
```

In [6]:

```
X_data = []
Y_data = []

for x in range(500):
    points = np.random.randint(1,25,2)
    X_data.append(points)
    Y_data.append(np.ones(1))

for x in range(500):
    points = np.random.randint(27,50,2)
    X_data.append(points)
    Y_data.append(np.zeros(1))
```

In [7]:

```
type(X_data)
```

Out[7]:

```
list
```

In [8]:

```
X_data = np.array(X_data)
Y_data = np.array(Y_data)
```

In [9]:

```
X_data
```

Out[9]:

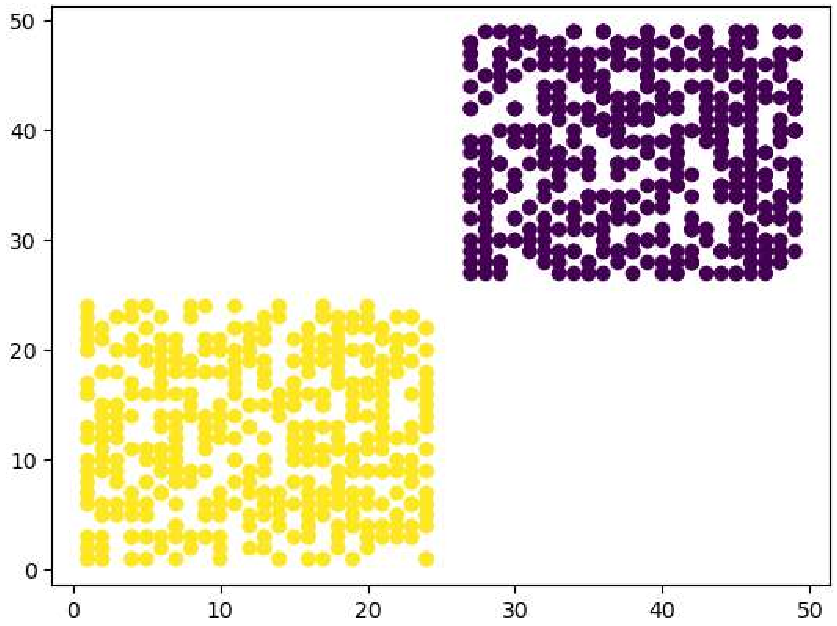
```
array([[ 3, 13],
       [ 1, 17],
       [15, 15],
       ...,
       [37, 37],
       [48, 40],
       [29, 28]])
```

In [10]:

```
plt.scatter(X_data[:,0],X_data[:,1], c = Y_data)
```

Out[10]:

```
<matplotlib.collections.PathCollection at 0x1ef36a1f130>
```



In [11]:

```
from sklearn.utils import shuffle
```

In [12]:

```
X_data, Y_data = shuffle(X_data, Y_data, random_state = 100)
```

In [13]:

```
split = 0.8
X_train = X_data[:int (X_data.shape[0]*split), :]
X_test = X_data[:int (X_data.shape[0]*split):, :]
Y_train = Y_data[:int (Y_data.shape[0]*split), :]
Y_test = Y_data[:int (Y_data.shape[0]*split):, :]
```

In [14]:

```
print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
```

```
(800, 2) (800, 2) (800, 1) (800, 1)
```

In [67]:

```
class KNN_classifier:
    def __init__(self,k = 5):
        self.k = k

    def initialize_data(self,X,Y):
        self.X = X
        self.Y = Y

    def distance_formula(self, X1, X2):
        return ((X1[0] - X2[0])**2 + (X1[1] - X2[1])**2)**0.5

    def predict(self, test_p):
        distance = []

        for i in range(self.X.shape[0]):
            distance.append((self.distance_formula(test_p,self.X[i]), i))

        distance = sorted(distance)
        distance = distance[:self.k]

        classes = []

        for dist, i in distance:
            classes.append(self.Y[i])

        all_classes, count = np.unique(classes, return_counts = True)

        max_count = np.argmax(count)

        print(f'Predicted class: {all_classes[max_count]}, probability:{count[max_count]/np.sum(count)}')

        return all_classes[max_count], count[max_count]/np.sum(count)
```

In [68]:

```
KNN_model = KNN_classifier(7)
```

In [69]:

```
KNN_model.initialize_data(X_train,Y_train)
```

In [70]:

```
KNN_model.predict(X_test[0])
```

```
Predicted class: 1.0, probability:1.0
```

Out[70]:

```
(1.0, 1.0)
```

In [71]:

```
Y_test[0]
```

Out[71]:

```
array([1.])
```

In [72]:

```
KNN_model.predict(np.array([26,26]))
```

```
Predicted class: 0.0, probability:1.0
```

Out[72]:

```
(0.0, 1.0)
```

In [73]:

```
corr = 0
for i in range(X_test.shape[0]):
    pred, prob = KNN_model.predict(X_test[i])
    if pred == Y_test[i]:
        corr +=1

print(corr/X_test.shape[0])
```

```
Predicted class: 0.0, probability:1.0
Predicted class: 0.0, probability:1.0
Predicted class: 1.0, probability:1.0
Predicted class: 1.0, probability:1.0
Predicted class: 0.0, probability:1.0
Predicted class: 1.0, probability:1.0
Predicted class: 0.0, probability:1.0
Predicted class: 0.0, probability:1.0
Predicted class: 0.0, probability:1.0
Predicted class: 0.0, probability:1.0
Predicted class: 1.0, probability:1.0
Predicted class: 0.0, probability:1.0
Predicted class: 1.0, probability:1.0
Predicted class: 1.0, probability:1.0
Predicted class: 1.0, probability:1.0
Predicted class: 1.0, probability:1.0
Predicted class: 0.0, probability:1.0
Predicted class: 1.0, probability:1.0
Predicted class: 1.0, probability:1.0
Predicted class: 1.0, probability:1.0
```

In [ ]:

In [ ]: