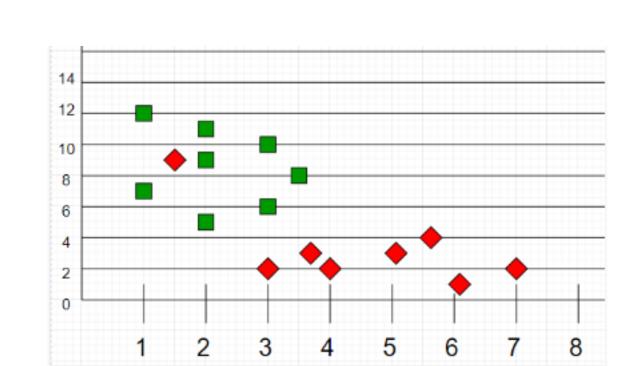
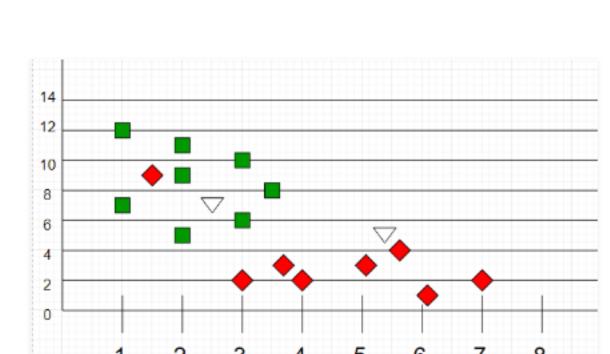
## **K-Nearest Neighbours**

KNN is one of the essential supervised classification algorithms. KNN is a non-parametric algorithm which means we have no idea about the distribution of our data.

Consider the following scatter plot for a dataset:



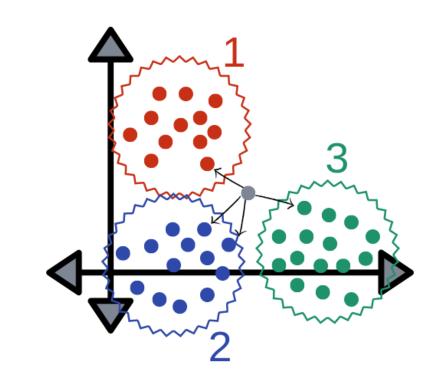
Consider the last points as the training set. Now it is time to apply the test set to our model.



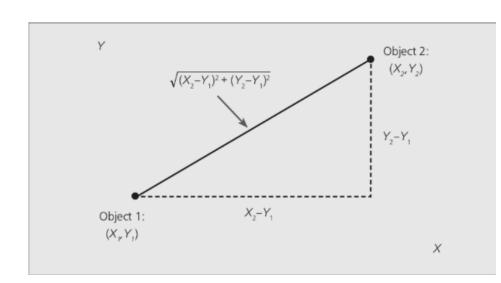
Our KNN model should classify the points into their suitable classes. As we can see on the visualization, separating our data from their class is so easy for human brains. The best method to do is finding the closest line between the data point and the selected neighbor point.

- Steps
  - Store the training set to an array. • Select an odd number for the number of selected points around the test point (K).
  - Calculate the distance for the test pint and each training points.

The whole process can be summarized with the image down below:



### **Euclidean Distance**



This is how we calculate the distance between 2 points. In mathematics, the Euclidean distance between two points in Euclidean space is the length of a line segment between the two points.

#### All the steps with extra details are included in the code files with comments.

import numpy as np import matplotlib.pyplot as plt from sklearn.datasets import load\_digits from sklearn.model\_selection import train\_test\_split from sklearn.metrics import accuracy\_score, confusion\_matrix %matplotlib inline

#### Return the dataset as 2 set of array.

features, labels = load\_digits(return\_X\_y = True)

print(features.shape) print(labels.shape)

(1797, 64) (1797,)

> class KnnModel: def \_\_init\_\_(self, k, x\_train, y\_train): self.k = kself.x\_train = x\_train self.y\_train = y\_train self.m, self.n = x\_train.shape def predict(self, x\_test): self.x\_test = x\_test self.m\_test, self.n = x\_test.shape y\_predict = np.zeros(self.m\_test) for i in range(self.m\_test): x = self.x\_test[i] neighbors = np.zeros(self.k) neighbors = self.find\_neighbors(x) y\_predict[i] = np.min(neighbors) return y\_predict def find\_neighbors(self, x): eucl\_distance = np.zeros(self.m) for i in range(self.m): d = self.calculate\_distance(x, self.x\_train[i]) eucl\_distance[i] = d index = eucl\_distance.argsort() y\_train\_sorted = self.y\_train[index] return y\_train\_sorted[:self.k] def calculate\_distance(self, x, x\_train): distance = np.sqrt(np.sum(np.square(x - x\_train))) **return** distance

# Validation set in KNN

In KNN algorithm, there is no training step because there is no model to build. Neither is there a validation measures model accuracy against the training data as a function of iteration count (training progress). Therefore, there is no need to split the data to 3 set of data.

x\_train, x\_test, y\_train, y\_test = train\_test\_split(features, labels, random\_state = 45, test\_size = 0.2)

model = KnnModel(3, x\_train, y\_train)

predicted\_values = model.predict(x\_test)

accuracy\_score(y\_test, predicted\_values)

Out[8]: 0.9805555555555555

In [9]: confusion\_matrix(y\_test, predicted\_values)

Out[9]: array([[27, 0, 0, 0, 0, 0, 0, 0, 0], [ 0, 33, 0, 0, 0, 0, 0, 0, 0], [ 0, 0, 35, 0, 0, 0, 0, 0, 0], [ 0, 0, 0, 41, 0, 0, 0, 0, 0, 0], [ 0, 0, 0, 0, 45, 0, 0, 0, 0, 0], [0, 0, 0, 0, 29, 0, 0, 0, 1], [ 1, 0, 0, 0, 0, 40, 0, 0, 0],

[ 0, 0, 0, 0, 0, 0, 39, 0, 0], [ 0, 0, 0, 0, 0, 0, 0, 32, 0], [ 0, 0, 0, 1, 1, 0, 0, 1, 2, 32]], dtype=int64)

In [10]: fig = plt.figure(figsize = (20, 20)) W = 10h = 10

> plt.gray() ax.matshow(x\_test[i].reshape(8, 8)) plt.xticks([]) plt.yticks([]) plt.subplots\_adjust(hspace = 1, wspace = 1) plt.title(f'y\_true: {y\_test[i]}\n y\_pred: {int(predicted\_values[i])}')

ax = fig.add\_subplot(w, h, i + 1)

y\_true: 8

y\_pred: 8

plt.show()

y\_true: 4

y\_pred: 4

y\_true: 4

y\_pred: 4

y\_true: 0

y\_pred: 0

y\_true: 5

y\_pred: 5

y\_true: 1

y\_pred: 1

y\_true: 3

y\_pred: 3

y\_true: 9

y\_pred: 8

y\_true: 9

y\_pred: 9

y\_true: 0 y\_pred: 0

for i in range(w \* h):

				y_predict	y_pred. 5	y_pred. 1	y_pred. s		
y_true: 4	y_true: 5	y_true: 0	y_true: 0	y_true: 6	y_true: 4	y_true: 6	y_true: 2	y_true: 2	y_true: 8
y_pred: 4	y_pred: 5	y_pred: 0	y_pred: 0	y_pred: 6	y_pred: 4	y_pred: 6	y_pred: 2	y_pred: 2	y_pred: 8
y_true: 9	y_true: 7	y_true: 7	y_true: 8	y_true: 1	y_true: 3	y_true: 9	y_true: 0	y_true: 7	y_true: 7
y_pred: 9	y_pred: 7	y_pred: 7	y_pred: 8	y_pred: 1	y_pred: 3	y_pred: 9	y_pred: 0	y_pred: 7	y_pred: 7
y_true: 6	y_true: 7	y_true: 4	y_true: 4	y_true: 5	y_true: 0	y_true: 6	y_true: 3	y_true: 2	y_true: 1
y_pred: 6	y_pred: 7	y_pred: 4	y_pred: 4	y_pred: 5	y_pred: 0	y_pred: 6	y_pred: 3	y_pred: 2	y_pred: 1
y_true: 0	y_true: 8	y_true: 7	y_true: 6	y_true: 9	y_true: 7	y_true: 8	y_true: 6	y_true: 7	y_true: 1
y_pred: 0	y_pred: 8	y_pred: 7	y_pred: 6	y_pred: 9	y_pred: 7	y_pred: 8	y_pred: 6	y_pred: 7	y_pred: 1
y_true: 8	y_true: 3	y_true: 4	y_true: 7	y_true: 4	y_true: 4	y_true: 2	y_true: 8	y_true: 8	y_true: 7
y_pred: 8	y_pred: 3	y_pred: 4	y_pred: 7	y_pred: 4	y_pred: 4	y_pred: 2	y_pred: 8	y_pred: 8	y_pred: 7
y_true: 4	y_true: 8	y_true: 5	y_true: 4	y_true: 6	y_true: 8	y_true: 6	y_true: 1	y_true: 5	y_true: 1
y_pred: 4	y_pred: 8	y_pred: 5	y_pred: 4	y_pred: 6	y_pred: 8	y_pred: 6	y_pred: 1	y_pred: 5	y_pred: 1
y_true: 7	y_true: 7	y_true: 9	y_true: 1	y_true: 2	y_true: 4	y_true: 9	y_true: 6	y_true: 4	y_true: 1
y_pred: 7	y_pred: 7	y_pred: 9	y_pred: 1	y_pred: 2	y_pred: 4	y_pred: 9	y_pred: 6	y_pred: 4	y_pred: 1
y_true: 7	y_true: 2	y_true: 0	y_true: 2	y_true: 3	y_true: 2	y_true: 8	y_true: 9	y_true: 9	y_true: 3
y_pred: 7	y_pred: 2	y_pred: 0	y_pred: 2	y_pred: 3	y_pred: 2	y_pred: 8	y_pred: 9	y_pred: 9	y_pred: 3
y_true: 1	y_true: 8	y_true: 3	y_true: 7	y_true: 2	y_true: 2	y_true: 6	y_true: 8	y_true: 0	y_true: 6
y_pred: 1	y_pred: 8	y_pred: 3	y_pred: 7	y_pred: 2	y_pred: 2	y_pred: 6	y_pred: 8	y_pred: 0	y_pred: 6