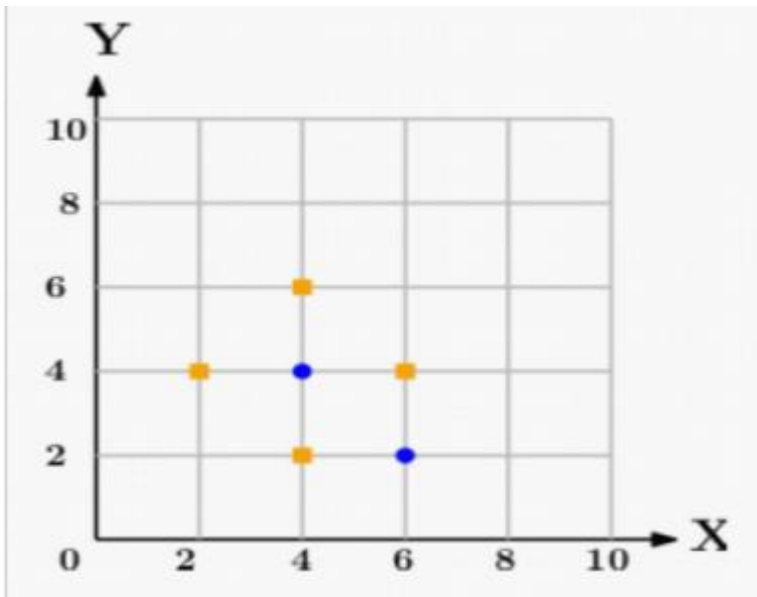


## Problem Statement

In the following diagram let blue circles indicate positive examples and orange square indicate negative examples. This assignment wants to use k-NN algorithm for classifying the points. If  $k=3$ , find the class of the point (6,6). Extend the same example for Distance-Weighted k-NN



## KNN Key points

- k nearest neighbours
- make prediction based on k number of nearest neighbours
- non parametric
- used for classification and regression
- distance metric - euclidean, manhattan
- lazy learning
- no training period
- Advantages
  1. intuitive algorithm
  2. can be used for classification and regression
- Disadvantages
  1. Time consuming for large dataset

2. Curse of dimensionality
  3. Sensitive to noisy data, missing values, outliers
- Application
    1. Recommendation Systems
  - Choosing the right value of K
    1. Choose odd value of k to avoid ties
    2. k can be taken as square root of N, where N is the number of samples in the dataset
    3. Choose K which gives least error

## Algorithm

1. Load the data
2.  $k :=$  chosen number of neighbors
3.  $p :=$  chosen minkowski order
4. collection  $:=$  empty list
5. For each instance in dataset
  - 5.1. Calculate the minkowski distance between the query and instance
  - 5.2. Add the distance and the target of the instance to collection
6. Sort the collection in ascending order by distance
7. Pick the first K entries from the sorted collection
8. Get the target of the selected K entries
9. if knn type is normal then
  - 9.1 return mode of the targets of K entries
10. if knn type is weighted then
  - 10.1 return target with the highest weighted sum

## Minkowski Distance

1. For Manhattan Distance,  $p = 1$
2. For Euclidean Distance,  $p = 2$

$$\left( \sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$$

## Weight for Distance weighted KNN

weight = 1/(distance + c)

### ▼ Source Code

```

1 from math import sqrt
2
3 def knn_classification(dataset, queries, k, distance_type='euclidean'):
4
5     print('knn_classification')
6
7     predictions = []
8
9     # For each query
10    for query in queries:
11
12        collection = []
13
14        # For each instance in dataset
15        for features, target in dataset:
16
17            # Calculate Distance
18            distance = 0
19            if distance_type == 'euclidean':
20                for instance_feature, query_feature in zip(features, query):
21                    distance = distance + ((instance_feature - query_feature)**2)
22            distance = sqrt(distance)
23            elif distance_type == 'manhattan':
24                for instance_feature, query_feature in zip(features, query):
25                    distance = distance + abs(instance_feature - query_feature)
26
27            '''
28            # Minkowski Distance
29            distance = 0
30            p = 2
31            if distance_type == 'manhattan':
32                p = 1

```

```

32     p=1
33     elif distance == 'euclidean':
34         p=2
35         for instance_feature, query_feature in zip(features,query):
36             distance = distance + ((instance_feature-query_feature)**p)
37         distance = distance**(1/p)
38     '''
39
40     # Add Target and Distance to Collection
41     collection.append([target,distance])
42
43     # Sort the collection in ascending order by distance
44     collection.sort(key = lambda collection: collection[1])
45
46     # Get the first k entries from the sorted collection
47     k_entries = collection[0:k]
48
49     # Get the target values of the k_entries
50     k_labels = [target for target,distance in k_entries]
51
52     # Get count of each target
53     target2count = {}
54     for target,distance in k_entries:
55         if target in target2count.keys():
56             target2count[target] = target2count[target] + 1
57         else:
58             target2count[target] = 1
59
60     # Prediction is the mode of k_labels i.e target of highest count
61     prediction = -1
62     max_count = -1
63     for target in target2count:
64         if target2count[target] > max_count:
65             prediction = target
66             max_count = target2count[target]
67
68     predictions.append(prediction)
69
70     return predictions
71
72 #####
73
74 dataset = [
75     ((4,2),1),
76     ((2,4),1),
77     ((6,4),1),
78     ((4,6),1),
79     ((6,2),0),
80     ((4,4),0)
81 ]
82
83 queries = [

```

```

84 (6,6)
85 ]
86
87 k = 3
88
89 #####
90
91 predictions = knn_classification(k=3,dataset=dataset,queries=queries)
92 for query, prediction in zip(queries, predictions):
93     print('Query = {query}'.format(query=query))
94     print('Prediction = {prediction}'.format(prediction=prediction))
95     print()

```

```

knn_classification
Query = (6, 6)
Prediction = 1

```

```

1 from math import sqrt
2
3 def distance_weighted_knn_classification(dataset,queries,k,distance_type='euclidean')
4
5     print('distance_weighted_knn_classification')
6
7     predictions = []
8
9     # For each query
10    for query in queries:
11
12        collection = []
13
14        # For each instance in dataset
15        for features, target in dataset:
16
17            # Calculate Distance
18            distance = 0
19            if distance_type == 'euclidean':
20                for instance_feature, query_feature in zip(features,query):
21                    distance = distance + ((instance_feature-query_feature)**2)
22            distance = sqrt(distance)
23            elif distance_type == 'manhattan':
24                for instance_feature, query_feature in zip(features,query):
25                    distance = distance + abs(instance_feature-query_feature)
26
27            # Add Target and Distance to Collection
28            collection.append([target,distance])
29
30            # Sort the collection in ascending order by distance
31            collection.sort(key = lambda collection: collection[1])
32
33            # Get the first k entries from the sorted collection

```

```

34     k_entries = collection[0:k]
35
36     # compute weighted Sum of each target
37     target2weight = {}
38     c = 0.0001
39     for target,distance in k_entries:
40         weight = 1/(distance + c)
41         if target in target2weight.keys():
42             target2weight[target] = target2weight[target] + weight
43         else:
44             target2weight[target] = weight
45
46     # Prediction is the target value with maximum weighted sum
47     prediction = -1
48     max_weighted_sum = -1
49     for target in target2weight:
50         if target2weight[target] > max_weighted_sum:
51             prediction = target
52             max_weighted_sum = target2weight[target]
53
54     predictions.append(prediction)
55
56     return predictions
57
58 #####
59
60 dataset = [
61     ((4,2),1),
62     ((2,4),1),
63     ((6,4),1),
64     ((4,6),1),
65     ((6,2),0),
66     ((4,4),0)
67 ]
68
69 queries = [
70     (6,6)
71 ]
72
73 k = 3
74
75 #####
76
77 predictions = distance_weighted_knn_classification(k=3,dataset=dataset,queries=que
78 for query, prediction in zip(queries, predictions):
79     print('Query = {query}'.format(query=query))
80     print('Prediction = {prediction}'.format(prediction=prediction))
81     print()

```

distance\_weighted\_knn\_classification  
Query = (6, 6)

Prediction = 1

```

1 #####
2
3 dataset = [
4     ((4,2),1),
5     ((2,4),1),
6     ((6,4),1),
7     ((4,6),1),
8     ((6,2),0),
9     ((4,4),0)
10 ]
11
12 queries = [
13     (6,6)
14 ]
15
16 k = 3
17
18 #####
19
20 predictions = knn_classification(k=3,dataset=dataset,queries=queries)
21 for query, prediction in zip(queries, predictions):
22     print('Query = {query}'.format(query=query))
23     print('Prediction = {prediction}'.format(prediction=prediction))
24     print()

```

knn\_classification  
Query = (6, 6)  
Prediction = 1

```

1 predictions = knn_classification(k=3,dataset=dataset,queries=queries)
2 for query, prediction in zip(queries, predictions):
3     print('Query = {query}'.format(query=query))
4     print('Prediction = {prediction}'.format(prediction=prediction))
5     print()

```

knn\_classification  
Query = (6, 6)  
Prediction = 1

```

1 predictions = distance_weighted_knn_classification(k=3,dataset=dataset,queries=que
2 for query, prediction in zip(queries, predictions):
3     print('Query = {query}'.format(query=query))
4     print('Prediction = {prediction}'.format(prediction=prediction))
5     print()

```

distance\_weighted\_knn\_classification  
Query = (6, 6)

Prediction = 1

