```
from math import sqrt

def knn_classification(dataset,queries,k,distance_type='euclidean'):

  print('knn_classification')

  predictions = []

  # For each query
  for query in queries:

    collection = []

    # For each instance in dataset
    for features, target in dataset:

      # Calculate Distance
      distance = 0
      if distance_type == 'euclidean':
        for instance_feature, query_feature in zip(features,query):
          distance = distance + ((instance_feature-query_feature)**2)
        distance = sqrt(distance)
      elif distance_type == 'manhattan':
        for instance_feature, query_feature in zip(features,query):
          distance = distance + abs(instance_feature-query_feature)

      '''
      # Minkowski Distance
      distance = 0
      p = 2
      if distance == 'manhattan':
        p=1
      elif distance == 'euclidean':
        p=2
      for instance_feature, query_feature in zip(features,query):
        distance = distance + ((instance_feature-query_feature)**p)
      distance = distance**(1/p)
      '''

      # Add Target and Distance to Collection
      collection.append([target,distance])

    # Sort the collection in ascending order by distance
    collection.sort(key = lambda collection: collection[1])

    # Get the first k entries from the sorted collection
    k_entries = collection[0:k]

    # Get the target values of the k_entries
    k_labels = [target for target,distance in k_entries]

    # Get count of each target
    target2count = {}
    for target,distance in k_entries:
```

```
      for target,distance in k_entries:
        if target in target2count.keys():
          target2count[target] = target2count[target] + 1
        else:
          target2count[target] = 1

      # Prediction is the mode of k_labels i.e target of highest count
      prediction = -1
      max_count = -1
      for target in target2count:
        if target2count[target] > max_count:
          prediction = target
          max_count = target2count[target]

      predictions.append(prediction)

    return predictions

##################################################################

dataset = [
  ((4,2),1),
  ((2,4),1),
  ((6,4),1),
  ((4,6),1),
  ((6,2),0),
  ((4,4),0)
]

queries = [
  (6,6)
]

k = 3

##################################################################

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

```
knn_classification
Query = (6, 6)
Prediction = 1
```

```
from math import sqrt

def distance_weighted_knn_classification(dataset,queries,k,distance_type='euclidean

  print('distance_weighted_knn_classification')

  predictions = []
```

```
    # For each query
    for query in queries:

      collection = []

      # For each instance in dataset
      for features, target in dataset:

        # Calculate Distance
        distance = 0
        if distance_type == 'euclidean':
          for instance_feature, query_feature in zip(features,query):
            distance = distance + ((instance_feature-query_feature)**2)
          distance = sqrt(distance)
        elif distance_type == 'manhattan':
          for instance_feature, query_feature in zip(features,query):
            distance = distance + abs(instance_feature-query_feature)

        # Add Target and Distance to Collection
        collection.append([target,distance])

      # Sort the collection in ascending order by distance
      collection.sort(key = lambda collection: collection[1])

      # Get the first k entries from the sorted collection
      k_entries = collection[0:k]

      # compute weighted Sum of each target
      target2weight = {}
      c = 0.0001
      for target,distance in k_entries:
        weight = 1/(distance + c)
        if target in target2weight.keys():
          target2weight[target] = target2weight[target] + weight
        else:
          target2weight[target] = weight

      # Prediction is the target value with maximum weighted sum
      prediction = -1
      max_weighted_sum = -1
      for target in target2weight:
        if target2weight[target] > max_weighted_sum:
          prediction = target
          max_weighted_sum = target2weight[target]

      predictions.append(prediction)

    return predictions

  ##############################################################

  dataset = [
    ((4,2),1),
    ((2,4),1),
    ((6,4),1)
```

```
    ((6,4),1),
    ((4,6),1),
    ((6,2),0),
    ((4,4),0)
  ]

  queries = [
    (6,6)
  ]

  k = 3

  ################################################################

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

```
    distance_weighted_knn_classification
    Query = (6, 6)
    Prediction = 1
```

```
  ################################################################

  dataset = [
    ((4,2),1),
    ((2,4),1),
    ((6,4),1),
    ((4,6),1),
    ((6,2),0),
    ((4,4),0)
  ]

  queries = [
    (6,6)
  ]

  k = 3

  ################################################################

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

```
    knn_classification
    Query = (6, 6)
    Prediction = 1
```

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

```
   knn_classification
   Query = (6, 6)
   Prediction = 1
```

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

```
   distance_weighted_knn_classification
   Query = (6, 6)
   Prediction = 1
```