```
import numpy as np


# Classification
data = np.array([[.3, .8], [-.3, 1.6], [.9, 0], [1, 1]])
classes = ['A', 'B', 'B', 'A']

def manhattan_distance(x,y):
  return np.absolute(x[0] - y[0]) + np.absolute(x[1] - y[1])


k = 3
new_x = np.array([.5, .2])
distances = []

for i, x in enumerate(data):
  distances.append((manhattan_distance(new_x, x), classes[i]))

distances
```

```
[(0.8, 'A'), (2.2, 'B'), (0.6000000000000001, 'B'), (1.3, 'A')]
```

```
distances.sort()
neighbors = distances[:k]

votes = {'A' : 0, 'B' : 0}
for n in neighbors:
  votes[n[1]] += 1

votes
```

```
{'A': 2, 'B': 1}
```

With no distance weighting, the output class would be A.

```
weighted_votes = {'A' : 0, 'B' : 0}

for n in neighbors:
  weighted_votes[n[1]] += 1/n[0]**2

weighted_votes
```

```
    {'A': 2.1542159763313604, 'B': 2.7777777777777772}
```

With distance weighting, the output class would be B.

```
# Regression problem.

reg_labels = [.6, -.3, .8, 1.2]
reg_distances = []

for i, x in enumerate(data):
  reg_distances.append((manhattan_distance(new_x, x), reg_labels[i]))

reg_distances
```

```
    [(0.8, 0.6), (2.2, -0.3), (0.6000000000000001, 0.8), (1.3, 1.2)]
```

```
reg_distances.sort()
reg_neighbors = reg_distances[:k]

values = []
weights = []
for i, val in reg_neighbors:
  weight = 1/i**2
  weights.append(weight)
  values.append(val*weight)

output = np.array(values).sum()/np.array(weights).sum()
output
```

```
    0.7846282024578213
```

The regression value with distance weighting is 0.7846