Title: Implementation of nearest neighbor algorithm

Objectives:

1. Understanding nearest neighbors
2. Supervised Technique
3. Easy to implement, thus great choice to start with to study more complex methods
4. Understanding different distance measures

Methodology:

Nearest neighbor algorithm is a pretty simple algorithm. It finds out the nearest neighbor of a given point and concludes that the class in which the nearest neighbor belongs to is the desired class of the unknown pattern also.

Now, the question is how to determine the distance to find out the nearest neighbor? There are basically many distance measure algorithms. Some popular distance measures will be discussed here.

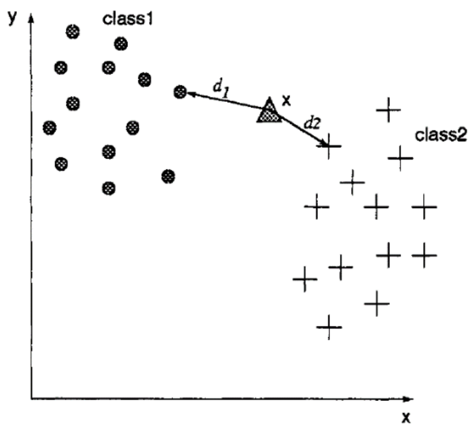

Figure: Nearest Neighbor Illustration

Nearest neighbor methods pose the problem of finding a reliable way of measuring the distance from one class sample to another. Obviously, we need to specify a distance metric that will allow us to measure the similarity of pattern samples in geometric pattern space. In practice, several methods are used.

Hamming distance measure: The most basic measure, and one that is widely used because of its simplicity, is Hamming distance measure. For two vectors

$$X = (x_1, x_2, \ldots)$$
$$Y = (y_1, y_2, \ldots)$$

The Hamming distance is found by evaluating the difference between each component of one vector with the corresponding component of the other, and summing these differences to provide an absolute value for the variation between the two vectors. The measure is defined by:

$$H = \sum |x_i - y_i|$$

The Hamming distance is often used to compare binary vectors. It is perhaps obvious that in this case the Hamming distance provides a value for the number of bits that are different between two vectors. In actual fact Hamming distance measure for binary data can be performed simply by the exclusive-OR function since

$$|x_i - y_i| \text{ is equivalent to } x_i \; XOR \; y_i$$

Euclidean Distance Measure: One of the most common metrices used is the Euclidean Distance measure. Consider an example in a rectangular coordinate system where we have two vectors (X and Y) that we wish to find the distance between them ($d(X, Y)$).

The shortest distance, is the Euclidean distance which is defined by:

$$d(X,Y)_{euc} = \sqrt{\left(\sum_{i=1}^{n} (X_i - Y_i)^2\right)}$$

where n is the dimensionality of the vector.

For the two-dimensional example, this gives us:

$$d(X,Y)_{euc} = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

There is nothing too strange about that, of course, as it is simply Pythagoras's theorem for the sides of a triangle. A special case given for binary vectors where the metric is then equivalent to the square root of the Hamming distance.

The Euclidean matric is widely used mainly because it is simple to calculate. For binary input vectors the matric reduces to a special case which is mathematically equivalent to the square root of the Hamming distance.

City Block Distance (Manhattan): A simplified version of the Euclidean distance measure is the city block measure. This method performs the Euclidean measure without calculating the squared or square root functions. Thus

$$D_{cb} = \sum_{n} |X_i - Y_i|$$

The effect of this, apart from the obvious one that it is much faster to compute than the Euclidean, is that points of equal distance from a vector lie on a square boundary about the vector, as opposed to a circular boundary for the Euclidean.

For the city block distance, anything falling on the square boundary will yield the same distance value. This does introduce some error into the measure, but this is accepted as a compromise between accuracy and speed of calculation.

Square Distance: Simplifying the Euclidean distance measure still further – but consequently adding more error – we have the square distance. With this measure the distance between two vectors is defined as the maximum of the differences between element of the two:

$$D_{sq} = MAX|X_i - Y_i|$$

Thus, different types of distance measures can be described. Before moving towards implementation, let's look at the flowchart.

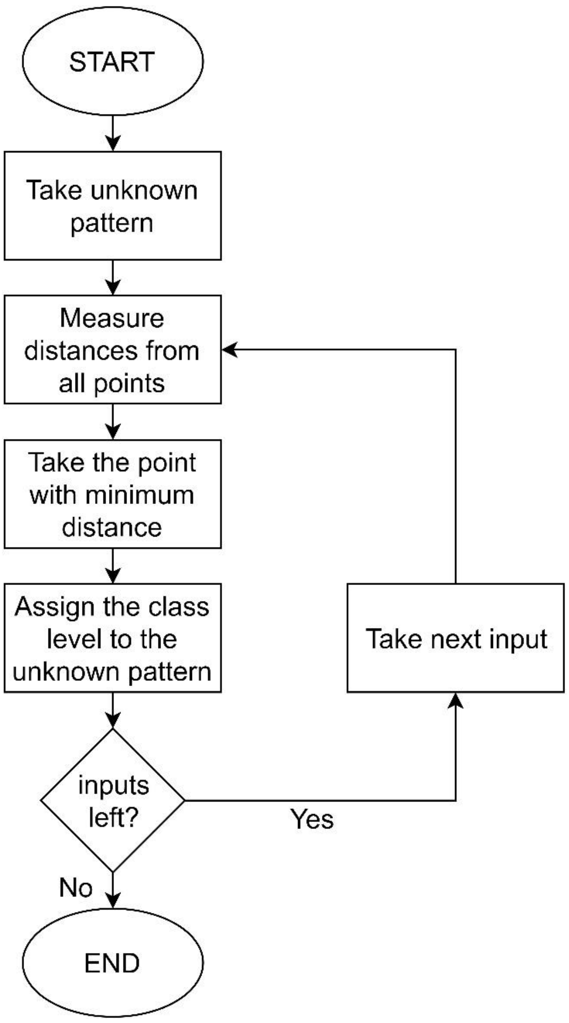
START
Take unknown pattern
Measure distances from all points
Take the point with minimum distance
Assign the class level to the unknown pattern
inputs left?
Yes
Take next input
No
END

Figure: Flowchart for Nearest Neighbor Algorithm

Implementation:

Implementation of Nearest Neighbor Algorithm is pretty easy. Here, three the algorithm has been applied by three distance measures:

1. Euclidean Distance Measure
2. Manhattan Distance Measure
3. Square Distance Measure

Each of the measures has given the outputs which has been shown also. Train and test sets has been predefined.

Code:

```
"""
Title: Nearest Neighbour Algorithm
Author: Azmain Yakin Srizon
"""
"""
Class 1: y=x*2
Class 2: y=x*5
"""

import math

# Declaring train data
train_data = [[1,2,1],[2,4,1],
              [3,6,1],[4,8,1],
              [5,10,1],[1,5,2],
              [2,10,2],[3,15,2],
              [4,20,2],[5,25,2]]

# Declaring test data
test_data = [[6,12],[7,35],
             [8,40],[9,18]]

# Calculating Euclidean distance
print('Euclidean Distance:')
for i in test_data:
    minimum_distance = 99999999999999
    selected_node = 0
    for j in train_data:
        distance = math.sqrt((j[0]-i[0])*(j[0]-i[0])+(j[1]-i[1])*(j[1]-i[1]))
        if distance < minimum_distance:
            minimum_distance = distance
            selected_node = j[2]
    print('Input: ' + str(i) + ', Predicted class: ' + str(selected_node))

# Calculating Mahattan Distance
print('\nManhattan Distance:')
for i in test_data:
    minimum_distance = 99999999999999
    selected_node = 0
    for j in train_data:
        distance = abs(j[0]-i[0])+abs(j[1]-i[1])
        if distance < minimum_distance:
            minimum_distance = distance
            selected_node = j[2]
    print('Input: ' + str(i) + ', Predicted class: ' + str(selected_node))

# Calculating Square Distance
print('\nSquare Distance:')
for i in test_data:
    minimum_distance = 99999999999999
    selected_node = 0
    for j in train_data:
        distance = max(abs(j[0]-i[0]),abs(j[1]-i[1]))
        if distance < minimum_distance:
            minimum_distance = distance
            selected_node = j[2]
    print('Input: ' + str(i) + ', Predicted class: ' + str(selected_node))
```

Output:

```
Euclidean Distance:
Input: [6, 12], Predicted class: 1
Input: [7, 35], Predicted class: 2
Input: [8, 40], Predicted class: 2
Input: [9, 18], Predicted class: 2

Manhattan Distance:
Input: [6, 12], Predicted class: 1
Input: [7, 35], Predicted class: 2
Input: [8, 40], Predicted class: 2
Input: [9, 18], Predicted class: 2

Square Distance:
Input: [6, 12], Predicted class: 1
Input: [7, 35], Predicted class: 2
Input: [8, 40], Predicted class: 2
Input: [9, 18], Predicted class: 2
```

Performance Analysis:

Although nearest neighbor algorithm is a very short and easy to implement algorithm but it has some analysis also. For the three distance measures, the output was the same. Hence the accuracy is somehow same. Here, there are two classes defined.

The first class is,

$$y = 2 \times x$$

And the second class is,

$$y = 5 \times x$$

But all three distance measures have given 3 correct outputs. Hence, they've an accuracy of 75%. But it can vary for different datasets. Different measures can give different output if the samples are designed in that manner. And it can not be said if the accuracy will be better or worse as nearest neighbor only focuses on the nearest value. It has no intension to find out any pattern or something like that.

So, this point gives us a realization that this algorithm does have some limitations and can't be used everywhere. The limitations are:

1. May not work for all datasets
2. Doesn't focus on patterns
3. Different distance measures may give different results

Apart from these limitations, Nearest Neighbor Algorithm is one of the fundamental algorithms of Neural Network and it'll help to implement more complex algorithms like K-Nearest Neighbors and other Neural Networks algorithms.