Nearest neighbor based classifiers

Nearest Neighbor (NN) rule classifies a sample based on the category of it's nearest neighbor. The NN based classifiers use some or all of patterns available in training set to classify a test pattern. The logic of classification involves finding the similarity between the test pattern and neighboring patterns in the training set.

Nearest neighbor algorithm – The NN algorithm assigns the class label of it's closest neighbor to a test pattern.

If there are 'n' training patterns, $(x_1, \theta_1); (x_2, \theta_2), \ldots\ldots, (x_n, \theta_n)$ where x_i is the dimension d and θ_i is the class label of the i^{th} pattern.

If P is the test pattern, then if

$d(P, X_K) = \min \{ d(P, X_i) \}$, where $i = 1, 2, \ldots\ldots, n$

Pattern P is assigned to the class θ_K associated with X_K.

Let the training set consist of the following patterns -

$X_1 = (0{\cdot}8, 0{\cdot}8, 1)$; $X_2 = (1{\cdot}0, 1{\cdot}0, 1)$; $X_3 = (1{\cdot}2, 0{\cdot}8, 1)$

$X_4 = (0{\cdot}8, 1{\cdot}2, 1)$; $X_5 = (1{\cdot}2, 1{\cdot}2, 1)$; $X_6 = (4{\cdot}0, 3{\cdot}0, 2)$

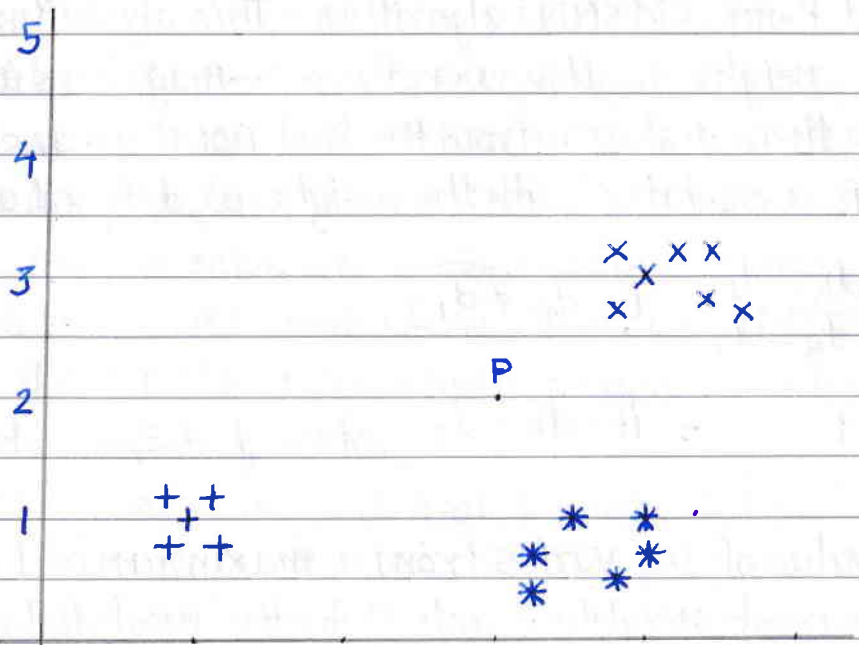
$X_7 = (3{\cdot}8, 2{\cdot}8, 2)$; $X_8 = (4{\cdot}8, 2{\cdot}8, 2)$; $X_9 = (3{\cdot}8, 3{\cdot}2, 2)$

$X_{10} = (4{\cdot}2, 3{\cdot}2, 2)$; $X_{11} = (4{\cdot}4, 2{\cdot}8, 2)$; $X_{12} = (4{\cdot}4, 3{\cdot}2, 2)$

$X_{13} = (3{\cdot}2, 0{\cdot}4, 3)$; $X_{14} = (3{\cdot}2, 0{\cdot}7, 3)$; $X_{15} = (3{\cdot}8, 0{\cdot}5, 3)$

$X_{16} = (3{\cdot}5, 1{\cdot}0, 3)$; $X_{17} = (4{\cdot}0, 1{\cdot}0, 3)$; $X_{18} = (4{\cdot}0, 0{\cdot}7, 3)$

Test pattern is $P = (3{\cdot}0, 2{\cdot}0, ?)$.



Computing Euclidean distance, e.g.

$d(X_1, P) = \sqrt{(0.8-3.0)^2 + (0.8-2.0)^2} = 2.51$ and so on.

It can be verified that $d(X_{16}, P) = 1.12$, which is the smallest distance. Since $X_{16} \in$ class 3, point P will be classified to belong to class 3.

K-nearest neighbor algorithm - Here, instead of finding just 1 nearest neighbor, 'k' neighbors are found. The majority class of these K nearest neighbors is the class label assigned to the new pattern. Typically 'k' is chosen to be an odd number to break the ties, and arrive at a majority.

Consider the previous example. Based on the distance computed, the 5 nearest neighbors are -

$X_{16} \in$ Class 3
$X_7 \in$ Class 2
$X_{14} \in$ Class 3
$X_6 \in$ Class 2
$X_{17} \in$ Class 3

Hence point P will be classified to belong to class 3 based on majority.

This method reduces the error in classification when training patterns are noisy, or the chosen training data is challenging and contains outliers. For large data sets, k can be larger to reduce the error. The value of k can be determined to meet the requirement through experimentation.

Modified K-nn (MKNN) algorithm - This algorithm also considers k-neighbors. However, these k-neighbors are weighted according to their distance from the test point. Each of the 'k' neighbors is associated with the weight w_j defined as

$$w_j = \frac{d_k - d_j}{d_k - d_1} \quad \text{if } d_k \neq d_1$$

$$1 \quad \text{if } d_k = d_1, \text{ where } j = 1, 2, \ldots\ldots, k.$$

The value of w_j varies from a maximum of 1 for the 'nearest' nearest neighbor, upto 0 for the 'most distant' nearest

The test pattern P is assigned to that class, for which the weights of the representatives among the k-nearest neighbors sums to the largest value.

Since MKNN applies a weighted majority role instead of simple majority rule, it means that outlier patterns have lesser effect on classification.

Consider again, $P = (3.0, 2.0, ?)$. For the five nearest points, the distances from P are, $d(P, X_{16}) = 1.12$; $d(P, X_7) = 1.13$; $d(P, X_{14}) = 1.32$; $d(P, X_6) = 1.41$; $d(P, X_{17}) = 1.41$.

The values of w will be,

$w_{16} = 1$ $\quad X_{16} \in$ class 3

$w_7 = \frac{1.41 - 1.13}{1.41 - 1.12} = 0.97$ $\quad X_7 \in$ class 2

$w_{14} = \frac{1.41 - 1.32}{1.41 - 1.12} = 0.31$ $\quad X_{14} \in$ class 3

$w_6 = 0$ $\quad X_6 \in$ class 2

$w_{17} = 0$ $\quad X_{17} \in$ class 3

$\therefore$ Sum Class 1 = 0

Sum Class 2 = $0.97 + 0 = 0.97$

Sum Class 3 = $1 + 0.31 + 0 = 1.31$

$\therefore$ Point P $\in$ Class 3.

r-near algorithm - If the nearest neighbor of the point of interest is very far away, then there is no relevance of such point in decision making. Hence, instead of looking at nearest neighbors, only the neighbors within some distance 'r' of the point of interest can be considered. The algorithm steps are -

i) Given the point P, determine the subdata in hypersphere of radius 'r' centered at P.

$B_r(P) = \{ X_i \in X$, such that $\| P - X_i \| \leq r \}$

ii) If $B_r(p)$ is empty, then output the majority class of the entire data set.

iii) If $B_r(p)$ is not empty, output the majority class of the data points in it.

This algorithm can be used to identify outliers. If a pattern does not have any similarity with the patterns within the chosen radius, it can be identified as an outlier.

The nearest neighbor classification method is very time consuming, especially when the number of training patterns is large. To overcome this, many efficient algorithms have been proposed. Some of those use a pre-processing step, which requires additional computing time, but it needs to be done only once.

Branch and bound algorithm – If the data are stored in an ordered fashion, all the points need not be required to be considered for finding the nearest neighbor. If the data is stored in a tree like data structure, the search for the nearest neighbor can be performed efficiently by not having to search down the branches where one can obtain a lower bound for the distance that is higher than the current best value found.

The data is clustered into representative groups S_j, where each group has the smallest possible radius. Each of such clusters has center μ_j and radius γ_j.

After searching one of the clusters, point X^* is found, which is the nearest neighbor from the cluster to the point of interest P. This distance is say $d = d(P, X_j^*)$. Then lower bounds can be computed for the distances in the other clusters.

For a point $X_k \in S_j$, by triangular inequality

$$d(P, \mu_j) \leq d(P, X_k) + d(X_k, \mu_j) \leq d(P, X_j) + \gamma_j.$$

Hence, a lower bound b_j can be computed for the distance between any $X_k \in S_j$ and P as

$$\min_{X_k \in S_j} d(P, X_k) \geq b_j = d(P, \mu_j) - \gamma_j$$

Clusters which satisfy the inequality $b_j \geq d$ need not be searched. The following steps can be followed –

i) Cluster the data points into L sets S_i, $i = 1, 2, \ldots L$. These are the first level clusters. For each cluster, maintain the centre u_i and the radius r_i. Within each cluster S_i, cluster the data into L sub-clusters $S_{i,j}$, $j = 1, 2, \ldots, L$. Maintain the centre $u_{i,j}$ and the radius $r_{i,j}$. These are second level clusters. Continue this type of recursive clustering until there are clusters of one point.

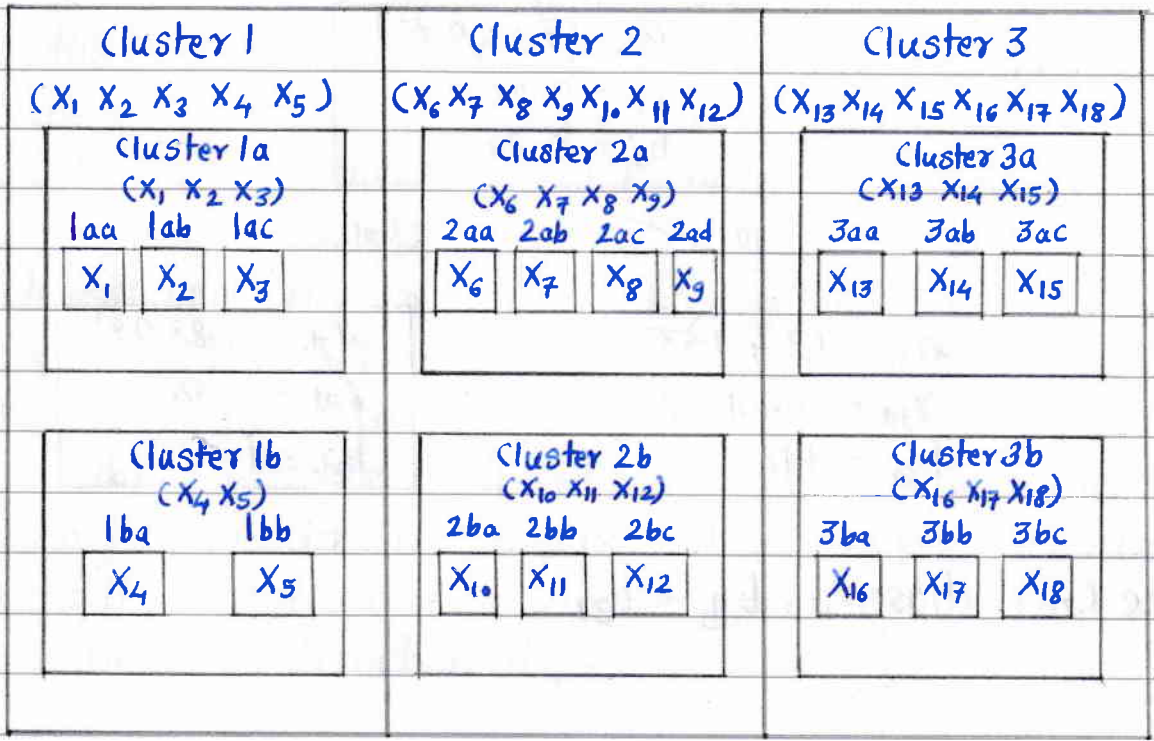
ii) To find the nearest neighbor to a new point P,

a. Branch step – First compute b_j. Then find the nearest neighbor in the cluster with the smallest b_j. If the clusters have sub clusters, this step is done recursively.

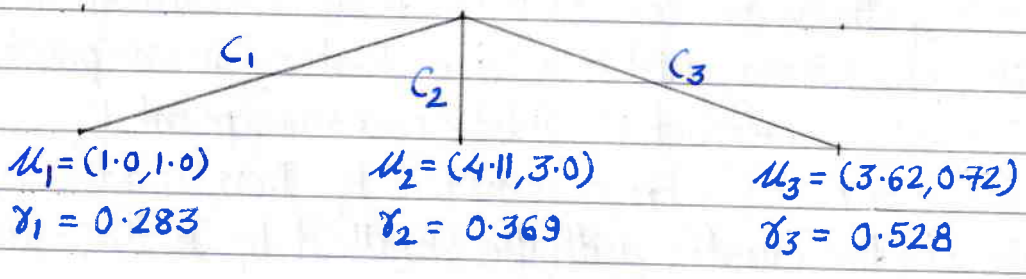
b. Bound step – If that cluster does not satisfy the bound, look into the cluster with the next highest b_j. Otherwise stop and output the result.

It can be shown that on an average, there is considerable improvement over the standard NN algorithm.

Consider the data for the 18 points mentioned earlier. Let the points of class 1 be taken as a cluster & similarly points of the other 2 classes as the other two clusters. Further cluster 1 is sub-clustered as cluster 1a and 1b & similarly clusters 2a, 2b and 3a, 3b are made. At next level, each point is taken to be a sub-cluster.



It can be seen that centre of cluster 1 is (1·0, 1·0). Now consider X_1 = (0·8, 0·8). Calculating the radius as $\sqrt{(1-0.8)^2+(1-0.8)^2}$ = 0·283. Similarly calculating the value of the radius for other elements of cluster 1, and choosing maximum amongst them, we get the radius r_1 = 0·283. Similar computations are made for the other two clusters.



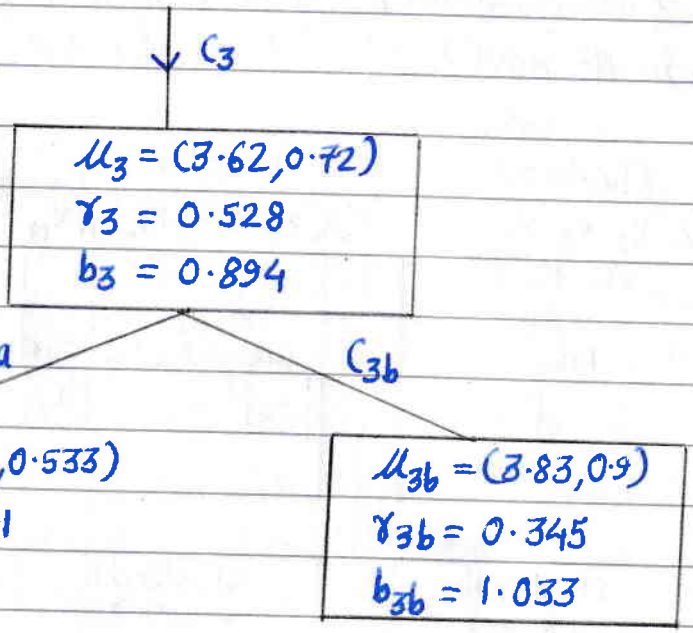
The test point is P (3·0, 2·0).
At the first level, b_j for each cluster j can be found.

$b_1 = d(P, \mu_1) - r_1 = 1.953$

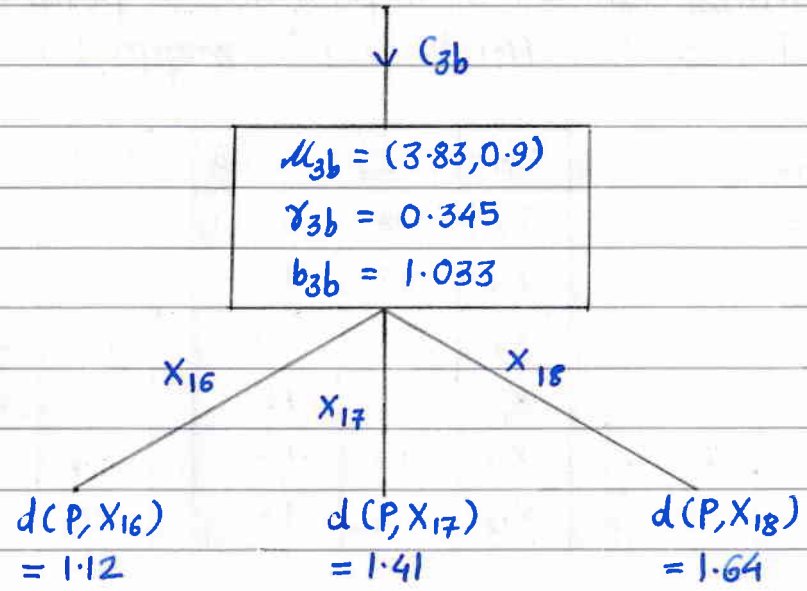
$b_2 = d(P, \mu_2) - r_2 = 1.125$

$b_3 = d(P, \mu_3) - r_3 = 0.894$ ..... smallest

Since b_3 is smallest, the sub-clusters of cluster 3 are searched.



Hence C_{3b} is chosen as $b_{3b} < b_{3a}$



The second sub-cluster of cluster 3 is searched and X_{16} is the point found to be closest to point P. The bound 'd' is calculated as the distance from P to X_{16}, which is 1.12.

Since b_1 and b_2 are greater than 'd', clusters 1 and 2 need not be searched and X_{16} is considered as the nearest neighbor of P.

Nearest neighbor search by projection

For a 1 dimensional space, if the point set is ordered, the nearest neighbor of point P can be found by locating position of P' in the order. Finding the successor and predecessor, and taking the closest, gives the nearest neighbor.

In the 2-D case, the points are sorted in increasing order using the first co-ordinate value. For finding nearest neighbor of P, locate position of P' in the sorted list of the first co-ordinate value. Then nearest neighbors of P are searched. We examine the point that is closer to P in the x-distance first and remember the closest point to P so far observed, say point Q. The search will be pruned as soon as it is known that any other point is further away from P than Q is.

This process is modified to project the points onto both the x and y-axes. The first step is on the x axis, the second on the y-axis, third on x-axis & so on. The search stops, when the search in either direction stops.

Example- Training data set of 18 points. Test point P=(3,2). The entire set of x-coordinates can be mapped as follows-

Point	X co-ord.	Difference
X_1	0·8	2·2
X_2	1	2
X_3	1·2	1·8
X_4	0·8	2·2
X_5	1·2	1·8
X_6	4	1

Point	X co-ord.	Difference
X_7	3·8	0·8
X_8	4·2	1·2
X_9	3·8	0·8
X_{10}	4·2	1·2
X_{11}	4·4	1·4
X_{12}	4·4	1·4

Point	X co-ord.	Difference
X_{13}	3·2	0·2
X_{14}	3·2	0·2
X_{15}	3·8	0·8
X_{16}	3·5	0·5
X_{17}	4	1
X_{18}	4	1

These points are rank ordered, and their actual distance from P is found. The point closest to P so far is kept in memory.

Hence the rank ordering is - $X_{13}, X_{14}, X_{16}, X_7, X_9, X_{15}, X_6, X_{17}, X_{18}, X_8, X_{10}, X_{11}, X_{12}, X_3, X_5, X_2, X_1, X_4$.

Starting from X_{13}, $d(P, X_{13}) = 1.612$ is calculated. X_{13} is kept in memory. After checking for X_{14}, $d(P, X_{14}) = 1.315$ is found and X_{14} is stored in memory. For X_{16}, $d(P, X_{16}) = 1.118$ and hence X_{16} is stored in memory. Continuing further, checking each time that difference $< d(P, X_{16})$, various distances are calculated, which continue to be $> d(P, X_{16})$ and hence X_{16} continues to be stored in memory. The next point X_8 has a distance in x-direction of 1·2, which is larger than the actual distance of the closest point X_{16}. Then it is not necessary to check any more points.

The entire set of y-coordinates can be mapped as follows-

Point	Y co-ord.	Difference
X_1	0·8	1·2
X_2	1	1
X_3	0·8	1·2
X_4	1·2	0·8
X_5	1·2	0·8
X_6	3	1

Point	Y co-ord.	Difference
X_7	2·8	0·8
X_8	2·8	0·8
X_9	3·2	1·2
X_{10}	3·2	1·2
X_{11}	2·8	0·8
X_{12}	3·2	1·2

Point	Y co-ord.	Difference
X_{13}	0·4	1·6
X_{14}	0·7	1·3
X_{15}	0·5	1·5
X_{16}	1	1
X_{17}	1	1
X_{18}	0·7	1·3

For the y-dimension, the rank ordering is - $X_4, X_5, X_7, X_8, X_{11}, X_2, X_6, X_{16}, X_{17}, X_1, X_3, X_9, X_{10}, X_{12}, X_{14}, X_{18}, X_{15}, X_{13}$.

Similar activity like along the x-dimension is carried out. $d(P, X_{16}) = 1.118$ is the minimum so far and hence X_{16} is stored. $d(P, X_{17}) = 1.414$, hence X_{16} stays stored in memory. The next point X_1 has a distance in y-direction of 1.2, which is larger than the actual distance of the closest point X_{16}. Hence it is not required to check any more points.

Since X_{16} emerges as the data point stored along x-direction as well as y-direction, after a single pass only X_{16} is identified as the nearest neighbor.