18CSE751 – Introduction to Machine Learning
Lecture 16: Nearest Neighbour Classifier

Dr.Vani Vasudevan

Professor –CSE, NMIT

K- NEAREST NEIGHBORS CLASSIFICATION

- **K Nearest Neighbors** is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., Distance functions). KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique.
- Also, KNN is classified under **instance based learning** which sometimes referred as **lazy learning** . This is because the process is delayed until a new instance must be classified.

KNN ALGORITHM

- A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If K = 1, then the case is simply assigned to the class of its nearest neighbor.

Distance functions

Euclidean
$$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Manhattan
$$\sum_{i=1}^{k}|x_i - y_i|$$

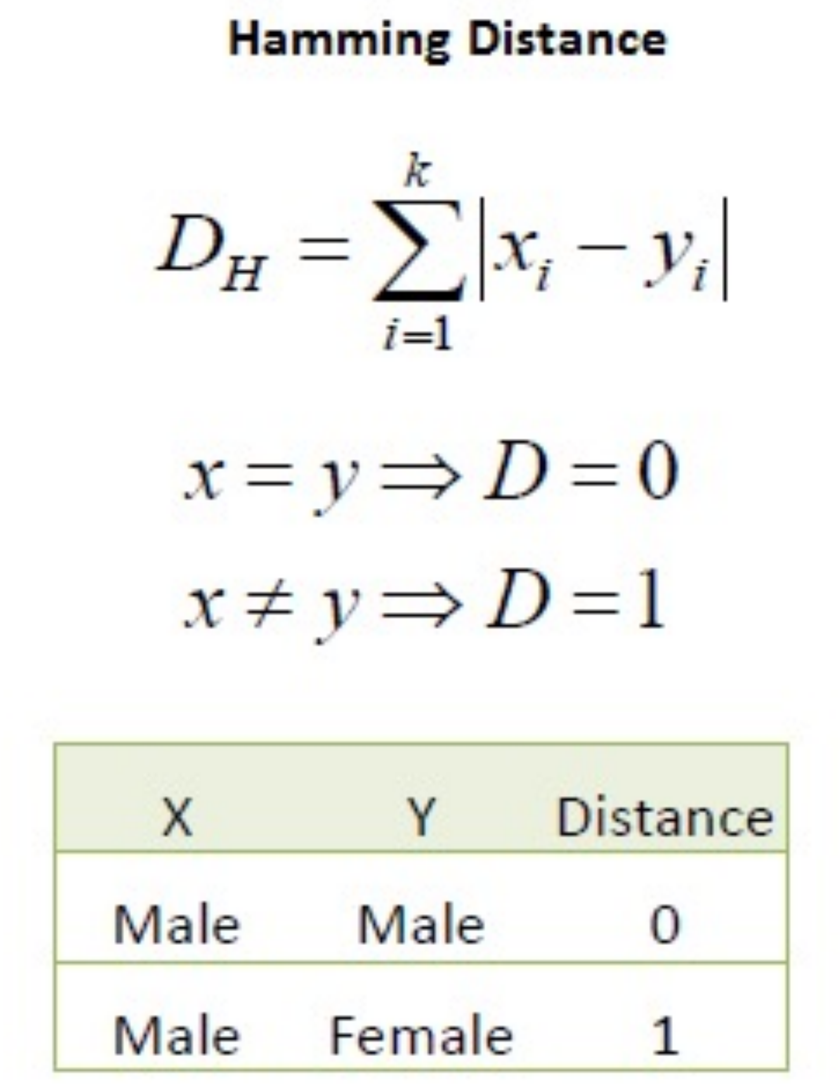
Minkowski
$$\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$$

KNN ALGORITHM

- It should also be noted that all three distance measures are only valid for continuous variables.
- In the instance of categorical variables, the hamming distance must be used.
- It also brings up the issue of standardization of the numerical variables between 0 and 1 when there is a mixture of numerical and categorical variables in the dataset.

KNN ALGORITHM

- Choosing the optimal value for K is best done by first inspecting the data. In general, a large K value is more precise as it reduces the overall noise but there is no guarantee. Cross-validation is another way to retrospectively determine a good K value by using an independent dataset to validate the K value. Historically, the optimal K for most datasets has been between 3-10. That produces much better results than 1NN.



Hamming Distance

$$D_H = \sum_{i=1}^{k} |x_i - y_i|$$

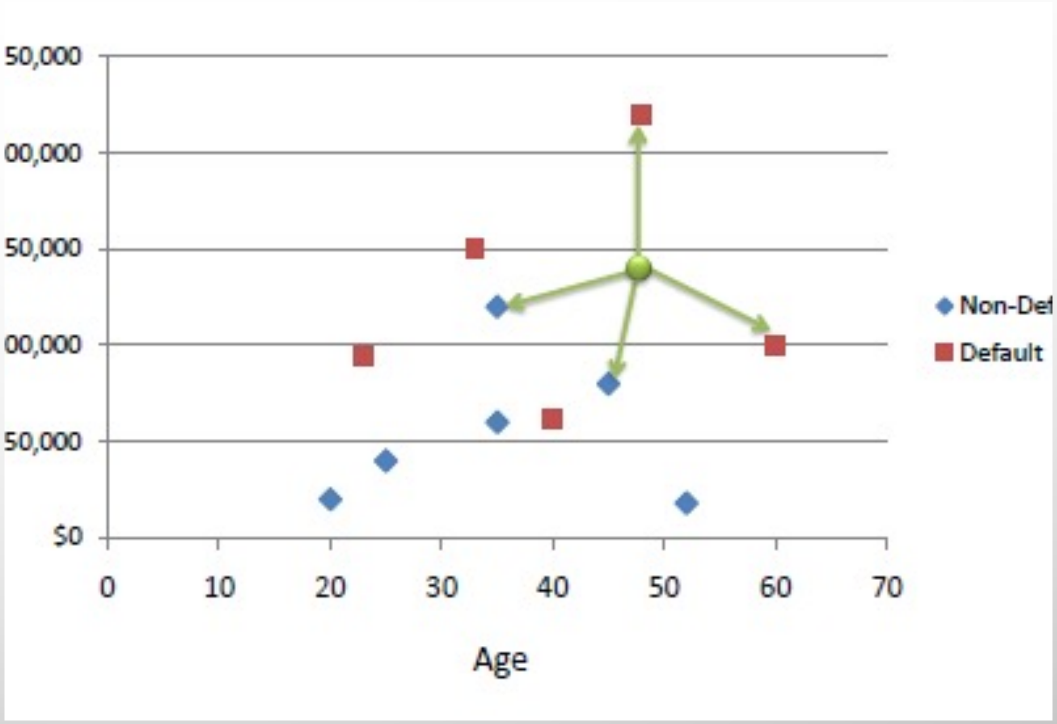
$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

X	Y	Distance
Male	Male	0
Male	Female	1

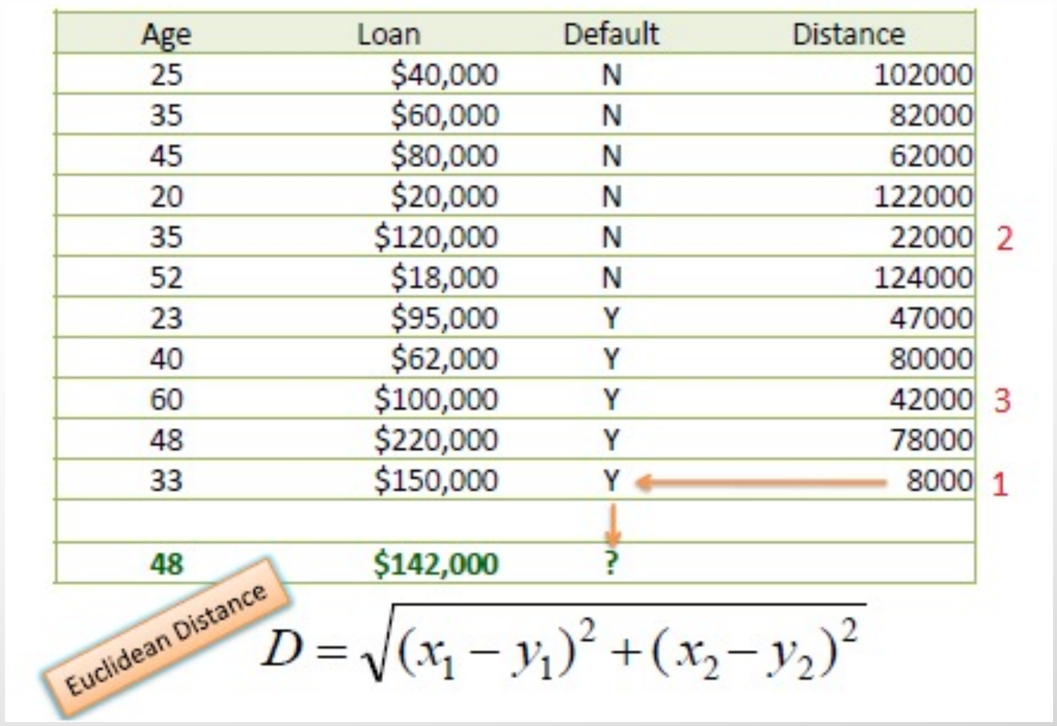
EXAMPLE

- CONSIDER THE FOLLOWING DATA CONCERNING CREDIT DEFAULT. AGE AND LOAN ARE TWO NUMERICAL VARIABLES (PREDICTORS) AND DEFAULT IS THE TARGET.



EXAMPLE

- WE CAN NOW USE THE TRAINING SET TO CLASSIFY AN UNKNOWN CASE (AGE=48 AND LOAN=$142,000) USING EUCLIDEAN DISTANCE. IF K=1 THEN THE NEAREST NEIGHBOR IS THE LAST CASE IN THE TRAINING SET WITH DEFAULT=Y.
- D = SQRT[(48-33)^2 + (142000-150000)^2] = 8000.01 >> DEFAULT=Y



Age	Loan	Default	Distance	
25	$40,000	N	102000	
35	$60,000	N	82000	
45	$80,000	N	62000	
20	$20,000	N	122000	
35	$120,000	N	22000	2
52	$18,000	N	124000	
23	$95,000	Y	47000	
40	$62,000	Y	80000	
60	$100,000	Y	42000	3
48	$220,000	Y	78000	
33	$150,000	Y	8000	1
48	**$142,000**	**?**		

Euclidean Distance

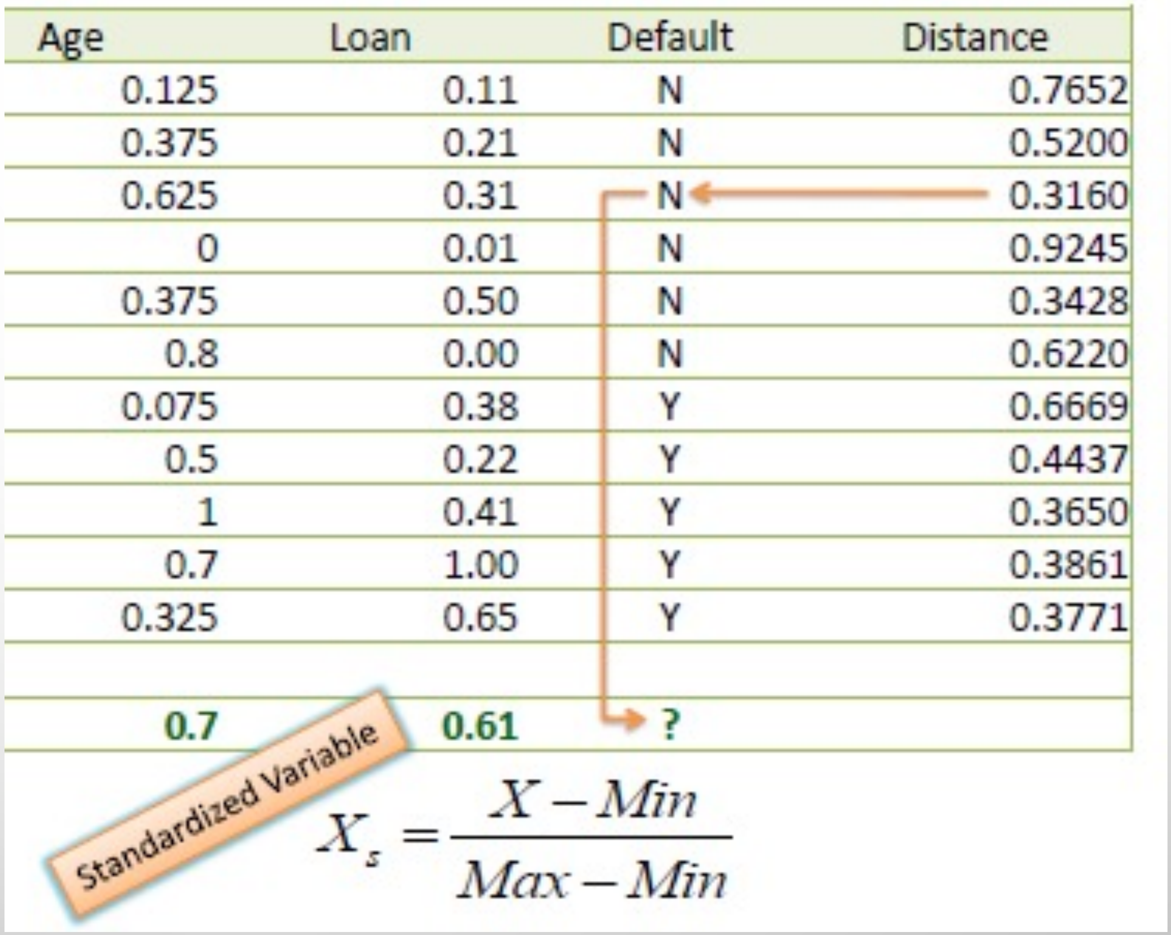
$$D = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

- With K=3, there are two Default=Y and one Default=N out of three closest neighbors. The prediction for the unknown case is again Default=Y.

EXAMPLE

Standardized distance

- One major drawback in calculating distance measures directly from the training set is in the case where variables have different measurement scales or there is a mixture of numerical and categorical variables.
- For example, if one variable is based on annual income in dollars, and the other is based on age in years then income will have a much higher influence on the distance calculated. One solution is to standardize the training set as shown aside.
- Using the standardized distance on the same training set, the unknown case returned a different neighbor which is not a good sign of robustness.



Age	Loan	Default	Distance
0.125	0.11	N	0.7652
0.375	0.21	N	0.5200
0.625	0.31	N	0.3160
0	0.01	N	0.9245
0.375	0.50	N	0.3428
0.8	0.00	N	0.6220
0.075	0.38	Y	0.6669
0.5	0.22	Y	0.4437
1	0.41	Y	0.3650
0.7	1.00	Y	0.3861
0.325	0.65	Y	0.3771
0.7	0.61	?	

Standardized Variable

$$X_s = \frac{X - Min}{Max - Min}$$

KNN

- The algorithm assumes all instances correspond to points in the n-dimensional space $\mathfrak{R}^n$.
- The nearest neighbors of an instance are defined in terms of the standard euclidean distance. More precisely, let an arbitrary instance x be described by the
- Feature vector $\langle a_1(x), a_2(x), \ldots a_n(x) \rangle$

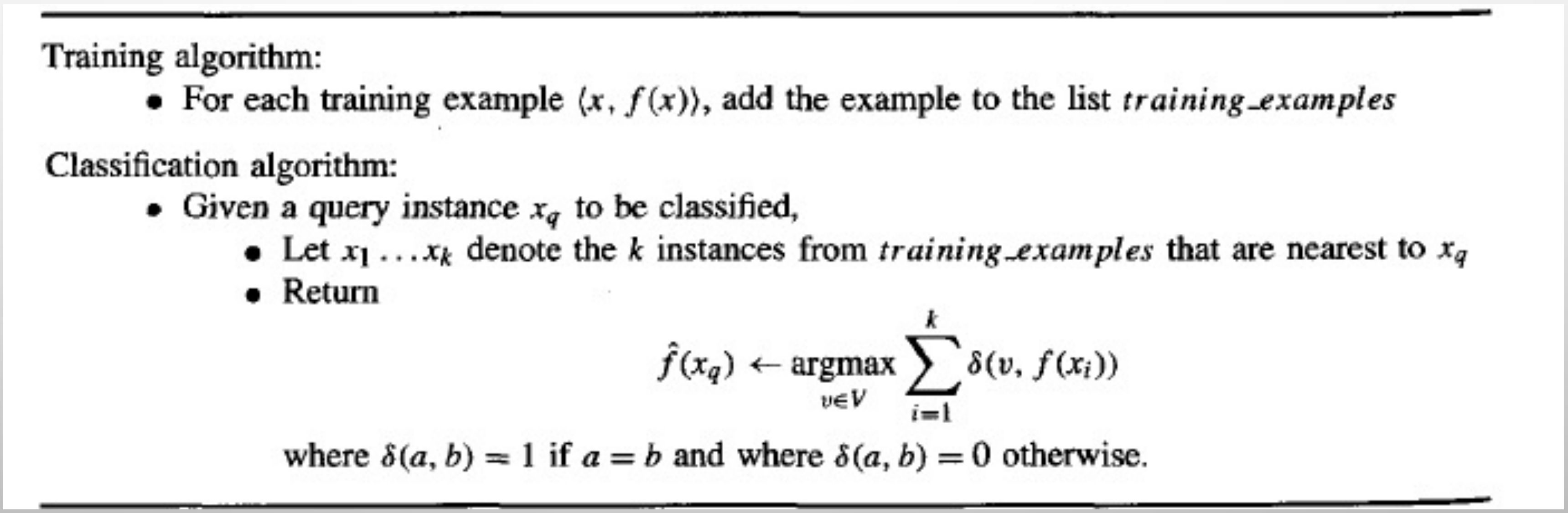
where $a_r(x)$ denotes the value of the r^{th} attribute of instance x. Then the distance between two instances x_i and x_j is defined to be $d(xi, x_j)$, where

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

KNN

- In Nearest-Neighbor learning the target function is either discrete-valued or real-valued.

1. Learning discrete-valued target functions of the form f : $\Re^n \rightarrow V$, where V is the finite set {v_1, . . . v_s}. The K-Nearest-Neighbor algorithm for approximating a discrete-valued target function is given in table.



```
Training algorithm:
    • For each training example ⟨x, f(x)⟩, add the example to the list training_examples

Classification algorithm:
    • Given a query instance $x_q$ to be classified,
        • Let $x_1 \ldots x_k$ denote the k instances from training_examples that are nearest to $x_q$
        • Return
              $\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$
          where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.
```

KNN

Voronoi diagram

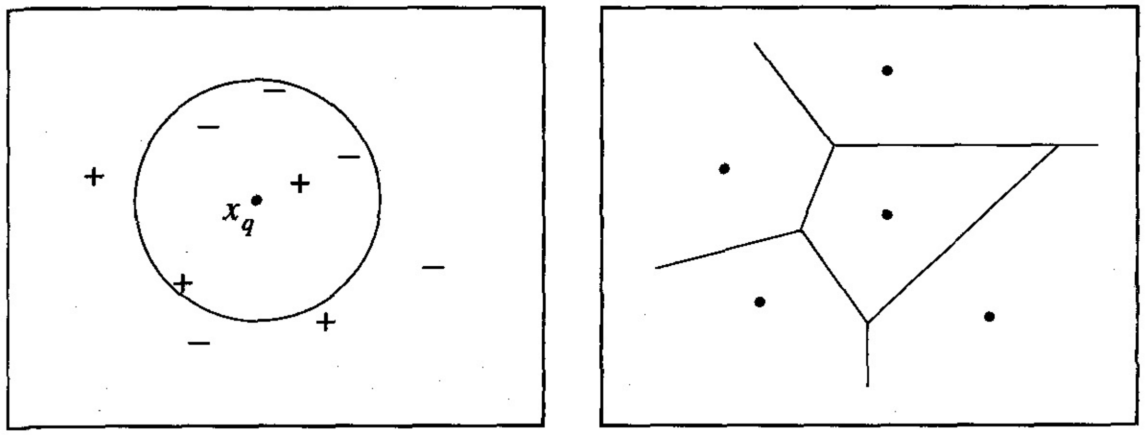


FIGURE 8.1
k-NEAREST NEIGHBOR. A set of positive and negative training examples is shown on the left, along with a query instance x_q to be classified. The 1-NEAREST NEIGHBOR algorithm classifies x_q positive, whereas 5-NEAREST NEIGHBOR classifies it as negative. On the right is the decision surface induced by the 1-NEAREST NEIGHBOR algorithm for a typical set of training examples. The convex polygon surrounding each training example indicates the region of instance space closest to that point (i.e., the instances for which the 1-NEAREST NEIGHBOR algorithm will assign the classification belonging to that training example).

KNN

2. The k-nearest NEIGHBOR algorithm is easily adapted to approximating continuous-valued target functions.

- **Calculate the mean value of the k nearest training examples rather than calculate their most common value.** More precisely, to approximate a real-valued target function f: : $\Re^n \rightarrow \Re$
- The final line of the above algorithm is replaced with $\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$

DISTANCE-WEIGHTED NEAREST NEIGHBOR ALGORITHM…

- Weight the contribution of each of the k neighbors according to their distance to the query point x_q , giving greater weight to closer neighbors.
- For example, in the algorithm of which approximates discrete-valued target functions, **weight the vote of each neighbor according to the inverse square of its distance from x_q**
- This can be accomplished by replacing the final line of the algorithm by

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$$

where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

DISTANCE-WEIGHTED NEAREST NEIGHBOR ALGORITHM…

Case 1: Where the query point x_q, exactly matches one of the training instances x_i and the denominator $d(x_q, x_i)^2$ is therefore zero, we assign $\hat{f}(x_q)$ to be $f(x_i)$

Case 2: If there are several such training examples, we assign the **majority classification** among them.

DISTANCE-WEIGHTED NEAREST NEIGHBOR ALGORITHM…

- We can distance-weight the instances for real-valued target functions in a similar fashion, replacing the final line of the algorithm in this case by

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

- Denominator in above equation is a constant that normalizes the contributions of the various weights (e.g, It assures that if f $(x_i) =$ c for all training examples, then f$(x_q) \rightarrow$ c as well).

DISTANCE-WEIGHTED NEAREST NEIGHBOR ALGORITHM…

- All the variants of the k-nearest NEIGHBOR algorithm **consider only the k nearest neighbors to classify the query point.**
- Once we add distance weighting, Its fine to allow all training examples to have an influence on the classification of the x_q, because **very distant examples will have very little effect on $\hat{f}(x_q)$**
- The only disadvantage of considering all examples is that the classifier will run more slowly.

DISTANCE-WEIGHTED NEAREST NEIGHBOR ALGORITHM…

- If all training examples are considered when classifying a new query instance, we call the algorithm a **global method**.
- If only the nearest training examples are considered, we call it a **local method**.
- When the rule in the below equation is applied as a global method, using all training examples, it is known as **shepard's method** (shepard 1968).

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

REMARKS ON K-NEAREST NEIGHBOR ALGORITHM

- The distance-weighted k-nearest neighbor algorithm is a **highly effective inductive inference method for many practical problems.**
- It is **robust to noisy training data**
- **quite effective** when it is provided a **sufficiently large set of training data**.
- The **weighted average of the k neighbors nearest to the query point can smooth out the impact of isolated noisy training examples**

WHAT IS THE INDUCTIVE BIAS OF K-NEAREST NEIGHBOR?

The inductive bias corresponds to an assumption that the classification of an instance x_q, will be most like the classification of other instances that are nearby in euclidean distance.

- Practical issue 1: **The distance between instances is calculated based on all attributes of the instance** (i.e., on all axes in the euclidean space containing the instances). It contrasts with methods such as rule and decision tree learning systems that select only a subset of the instance attributes when forming the hypothesis.

ISSUES WITH K-NEAREST NEIGHBOR…

- Approach to overcoming this problem is to weight each attribute differently when calculating the distance between two instances.
- Solution :
- This corresponds to stretching the axes in the Euclidean space, shortening the axes that correspond to less relevant attributes, and lengthening the axes that correspond to more relevant attributes. The amount by which each axis should be stretched can be determined automatically using **a cross-validation** approach.
- 1. To stretch (multiply) the j^{th} axis by some factor z_j, where the values $z_1 \ldots z_n$, are chosen to minimize the true classification error of the learning algorithm.
- 2. True error can be estimated using cross validation.

ISSUES WITH K-NEAREST NEIGHBOR…

Algorithm 1:

a. Select a random subset of the available data to use as training examples, then determine the values of $z_1 \ldots z_n$, that lead to the minimum error in classifying the remaining examples.

b. By repeating this process multiple times the estimate for these weighting factors can be made more accurate.

c. This process of stretching the axes in order to optimize the performance KNN provides a mechanism for suppressing the impact of irrelevant attributes

- Alternative : completely eliminate the least relevant attributes from the instance space. This is equivalent to setting some of the z_i scaling factors to zero.

ISSUES WITH K-NEAREST NEIGHBOR…

Algorithm 2:

Moore and lee (1994) discuss efficient cross-validation methods for selecting relevant subsets of the attributes for knn algorithms **leave-one-out cross validation,**

- Set of m training instances is repeatedly divided into a training set of size m - 1 and test set of size 1, in all possible ways.

ISSUES WITH K-NEAREST NEIGHBOR…

Practical issue 2: **Efficient memory indexing**.

- KNN algorithm delays all processing until a new query is received, significant computation can be required to process each new query.
- Various methods have been developed for indexing the stored training examples so that the nearest neighbors can be identified more efficiently at some additional cost in memory.
- One such indexing method is the kd-tree (bentley 1975; friedman et al. 1977), in which instances are stored at the leaves of a tree, with nearby instances stored at the same or nearby nodes. The internal nodes of the tree sort the new query x_q, to the relevant leaf by testing selected attributes of x_q.

ISSUES SUMMARIZED

1. **The cost of classifying new instances can be high.** This is because nearly all computation takes place at classification time rather than when the training examples are first encountered. Therefore, techniques for efficiently indexing training examples are a significant practical issue in reducing the computation required at query time.

2. Nearest Neighbor approaches consider all attributes of the instances when attempting to retrieve similar training examples from memory. **If the target concept depends on only a few of the many available attributes, then the instances that are truly most "similar" may well be a large distance apart**

UNIT IV

DECISION TREES : LEARNING WITH TREES, USING DECISION TREES, UNIVARIATE TREES, CLASSIFICATION TREES, REGRESSION TREES, PRUNING, RULE EXTRACTION FROM TREES, LEARNING RULES FROM DATA, MULTIVARIATE TREES, ID3, EXAMPLES **SUPPORT VECTOR MACHINES** : OPTIMAL SEPARATION, KERNELS, SVM ALGORITHM, MULTICLASS CLASSIFICATION, SVM REGRESSION (T2-CHAPTER-9,13;T1-CHAPTERS 8,12)

REFERENCES

T1: STEPHAN MARSLAND, **MACHINE LEARNING, AN ALGORITHMIC PERSPECTIVE,** CRC PRESS SECOND EDITION, 2015.

T2: ETHEM ALPAYDIN, **INTRODUCTION TO MACHINE LEARNING**, 2ND ED., PHI LEARNING PVT. LTD., 2013

R1: TOM M. MITCHELL, **MACHINE LEARNING**, MCGRAW-HILL EDUCATION (INDIAN EDITION), 2013