CHAPTER 2

Principles and Theories of the Study

This chapter delineates the essential basic theories used in this study. First, section 2.1 describes the definition of K-nearest neighbor. Section 2.2 describes the fuzzy membership functions and section 2.3 describes the fuzzy K-nearest neighbor. Section 2.4 describes the String grammar and section 2.5 describes the fuzzy K-nearest neighbor. The last section explains the String Grammar Fuzzy K-nearest neighbor.

2.1 K-Nearest neighbor

K-nearest neighbor was proposed by Fix, E. and Hodges, J. [1, 18]. It is a method for non- parametric estimation solution for data classification. The principle is similar to the extended window to find an equal to the number of K value, for example $K = 3$ as shown in figure 2.1, the window will extend until we found the three nearest neighbors. After that, the voting scheme is utilized in the classification.

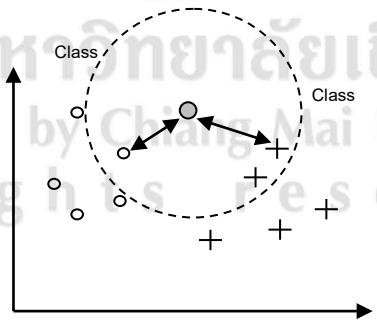
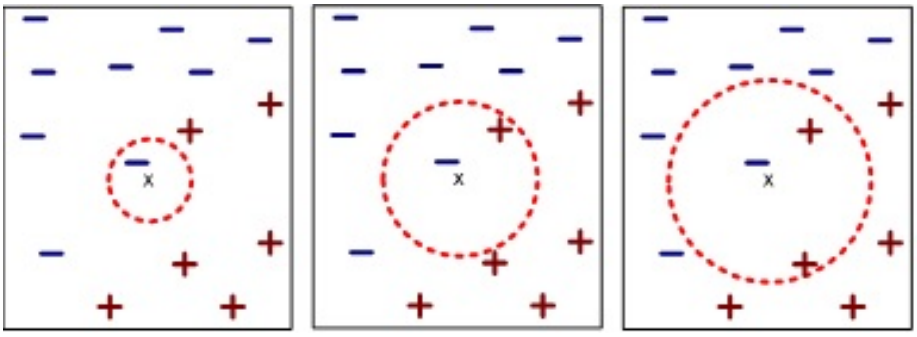


Figure 2.1 An example of a circle where it will be extended indefinitely until the three nearest samples are found.



(a) 1-nearest neighbor (b) 2-nearest neighbors (c) 3-nearest neighbors

Figure 2.2 Example of *K-nearest* neighbor (K-NN)

Figure 2.2 shows the test data point as "×". For K = 1, 2, 3, we construct a circle with "×" is a center. We extend the circle's size according to the value of k. We then count samples in each class, for example in 1-nearest neighbor, the nearest sample is in class "-", hence, the test data will be in class "-". In 2-NN case, there is a tie between class "+" and class "-". But, the sample from class "-" is closer than the one from class "+". Hence, the test data will be in class "-". Again in 3-NN case, the majority in class "+", hence, the test data are in class "+".

The algorithm of K-NN [19] is as follows:

- Compute distance between a test data point to every train data points using Euclidean distance method.
- Sort all distances by increasing order.
- Select the set of k closest distance.
- The class label of test data is assigned based on the majority vote of its nearest neighbors

2.2 Fuzzy membership functions

2.2.1 Fuzzy Sets

Zadeh, L. A. [20] proposed fuzzy sets theory. Given a universe U, crisp set A of U is commonly defined by specifying the objects of the universe that are members of *A*. An equivalent way of defining A is to specify the characteristic function of *A*, $\mathrm{U}_A : \mathrm{U} \to \{0, 1\}$ for all $x \in \mathrm{U}$ where

$$\mathrm{U}_A(x) = \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases} \tag{2.1}$$

Nevertheless, fuzzy sets are obtained by generalizing the concept of a characteristic function to a membership function u: U → [0, 1].

The advantage of the fuzzy sets is that the degree of membership in a set can be specified to any number between 0 and 1. This can be especially advantageous in pattern recognition and classification method whereas objects are not clearly members of one class or another.

Using crisp techniques, an ambiguous object will be assigned to one class only, which a precision of the assignment is not warranted. On the other hand, fuzzy techniques will specify to what degree the object belongs to each class.

For a fuzzy K-nearest neighbor [8]. Let $\{\vec{x}_1,...,\vec{x}_n\}$ be a set of sample vectors. The *c* fuzzy partitions of these vectors specify the degrees of membership of each vector in each of *c* classes. It is denoted by the *c* × *n* matrix U where, $u_{ik} = u_i(\vec{x}_k)$ for *i* = 1,…,*c*, and *k* = 1, …, *n*, is the degree of membership of $\vec{x}_k$ in class *i*. The following properties must be true for U to be a fuzzy *c* partition:

$$\sum_{i=1}^{c} u_{ik} = 1 \tag{2.2}$$

$$0 < \sum_{k=1}^{n} u_{ik} \le n$$

$$u_{ik} \in [0,1]$$

2.3 Fuzzy K-Nearest Neighbor

Keller, J. M. and Hunt, D. J. [9] proposed the formulation of a method for assigning fuzzy membership values given a set of labeled sample vectors. The method is to transform the crisp partition of the vectors (defined by the labels) into a fuzzy partition. The transformation process was designed so that the membership value of a vector for the class to which it belongs to. The membership values have several properties as follows.

1) It should be 1.0 if the vector is equal to the mean of its class.

2) It should be 0.5 if the vector is equal to the mean of the other class.

3) It should be near 0.5 if the vector is equidistant from the two means.

4) It should never be less than 0.5.

5) As a vector gets closer to its mean and farther from the other mean, the membership value should approach 1.0 exponentially.

6) It should depend on relative distances from the means of the classes rather than absolute distances.

The following method of assigning fuzzy membership values satisfies the above conditions.

For $\overline{x_k}$ in class 1:

$$u_{1k} = 0.5 + \frac{\exp(f(d_2 - d_1)/d) - \exp(-f)}{2(\exp(f) - \exp(-f))}, \tag{2.3}$$

and

$$u_{2k} = 1 - u_{1k} \ . \tag{2.4}$$

For $\overline{x_k}$ in class 2:

$$u_{1k} = 1 - u_{2k} \ , \text{ and} \tag{2.5}$$

$$u_{2k} = 0.5 + \frac{\exp(f(d_1 - d_2)/d) - \exp(-f)}{2(\exp(f) - \exp(-f))} \tag{2.6}$$

where d_1 and d_2 are the distance from the testing sample vector to the center of class 1 and class 2, respectively. d is the distance between the two centers of each class and f is a parameter used for controlling the rate at which memberships decrease toward 0.5, normally $f > 0$.

We can assign fuzzy membership value [8] to the labeled samples with each sample $\vec{x}$ in class i using a K-nearest neighbor rule, and then the membership values in each class can be calculated as follows:

$$u_j = \begin{cases} 0.51 + 0.49(n_j / K), & if\ j = i \\ 0.49(n_j / K), & if\ j \neq i \end{cases} \tag{2.7}$$

where n_j is the number of the neighbors found which belong to the class j.

K is K-nearest neighbor value.

The fuzzy algorithm is similar to the crisp version in the sense that it must also search the labeled sample set for the K-nearest neighbors. Beyond obtaining these K samples, the procedures differ considerably.

For a fuzzy K-nearest neighbor [8]. Let $W = \{\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n}\}$ be a set of n labeled samples. Also, let $u_i(\vec{x})$ be the assigned membership of the vector $\vec{x}$ (to be computed) as

$$u_i(\vec{x}) = \frac{\sum_{j=1}^{K} u_{ij} \left(1 / \left\| \vec{x} - \vec{x}_j \right\| \right)^{2/m-1}}{\sum_{j=1}^{K} \left(1 / \left\| \vec{x} - \vec{x}_j \right\| \right)^{2/m-1}} \tag{2.8}$$

where u_{ij} is the membership in the i^{th} class of the j^{th} vector of the labeled sample set.

A fuzzy K-nearest neighbor algorithm is as follows:

BEGIN

Input x, of unknown classification.

Set K, $1 \leq K \leq n$.

Initialize $i = 1$

DO UNTIL (K-nearest neighbors of $\vec{x}$ are found)

Compute distance from $\vec{x}$ to $\vec{x}_i$

IF ($i \leq K$) THEN

Include $\vec{x}_i$ in the set of K-nearest neighbors

ELSE IF ($\vec{x}_i$ closer to $\vec{x}$ than any previous nearest neighbor)

THEN

Delete the farthest of the K-nearest neighbors

Include $\vec{x}_i$ in the set of K-nearest neighbors.

END IF

END DO UNTIL

Initialize $i = 1$.

FOR $i = 1$ to c

Compute $u_i(\vec{x})$ using (2.8).

Increment i.

END

As seen in equation 2.8, the assigned memberships of $\vec{x}$ using the inverse distance serves more weight to a vector's membership if it is closer and less weight if it is further from the vector under consideration. The labeled samples can be assigned class memberships in several ways. In [8] shows that the variable m determines how heavily the distance is weighted when calculating each neighbor's contribution to the membership value. If m is two, then the contribution of each neighboring point is weighted by the reciprocal of its distance from the point being classified. As m

increases, the neighbors are more evenly weighted, and their relative distances from the point being classified have less effect. As *m* approaches one, the closer neighbors are weighted far more heavily than those farther away, which has the effect of reducing the number of points that contribute to the membership value of the point being classified.

2.4 String grammar

Phrase structure grammar was originally introduced by Chomsky, N. [21].

A phrase structure grammar G is a 4-tuple

$$G = (V_N, V_T, P, S) \tag{2.9}$$

where

V_N is finite set of nonterminal symbols

V_T is finite set of terminal symbols, $V_N \cup V_T = V, V_N \cup V_T = \lambda$

S is start symbol, $S \in V_N$

P is finite set of productions or rewriting rules of the form $\alpha \to \beta$, $\alpha, \beta \in V^*$, $\alpha = \lambda$; V* is the set of all finite length strings of symbols from V, including λ , the null string, $V+ = V^* - \{\lambda\}$.

V (alphabet) is a finite (non-empty set of symbols)

λ is an empty string

Let G = (V_N, V_T, P, S) be a grammar. If every production in P is of the form A→ aB, or A →a, A, B $\in V_N$, a $\in V_T$, then the grammar G is the finite-state or the regular grammar.

Phrase structure grammars have been used to describe patterns in syntactic pattern recognition [12]. Each pattern is represented by a string of primitives corresponding to a sentence in a language (tree or graph in high dimensional grammars). All strings which belong to the same class are generated by one grammar.

2.5 String Grammar Nearest Neighbor

String grammar nearest neighbor [22] is the method which is used for computing the closely distance between the testing sample string and all training sample strings.

For string grammar nearest neighbor, the Levenshtein distance [12, 22-28] is used as a distance metric.

We identify a string of object *i* (st_i) to the closest string of object *j* (st_j) as follows,

$$\mathbf{st}_i \text{ is in } j^{th} \text{ object if } d\left(\mathbf{st}_i, \mathbf{st}_j\right) = \min_{1 \le k \le TN} \left(d\left(\mathbf{st}_i, \mathbf{st}_k\right)\right) \tag{2.10}$$

where *TN* is the number of objects in the training dataset. The distance between string i and string k is Levenshtein distance.

2.6 String Grammar K-Nearest Neighbor

String grammar *K*-nearest neighbor [22] is the method which is used for computing the *K* closely distance between the testing sample string and all training sample strings. For string grammar nearest neighbor, the Levenshtein distance is used as a distance matric.

Levenshtein distance [12, 22-28] is the distance used for measuring the difference between two string metric sequences. Informally, the Levenshtein distance between two words is equal to the number of single-characters which requires to change one word into others.

The Levenshtein distance between two strings is defined as the minimum number of edits needed to transform one string into the other, with the allowable editing operations being insertion, deletion, or substitution of a single character. It is named after Vladimir Levenshtein, who considered this distance in 1965. It is closely related to pairwise string alignments.

The algorithm of Levenshtein distance between two strings *a* and *b* :

$$Lev_{a,b}\left(i,j\right) = \begin{cases} \max\left(i,j\right) & ,if\ \min\left(i,j\right)=0 \\ \min\begin{cases} Lev_{a,b}\left(i-1,j\right)+1 \\ Lev_{a,b}\left(i,j-1\right)+1 \\ Lev_{a,b}\left(i-1,j-1\right)+\left[a_i \neq b_j\right] \end{cases} & ,otherwise \end{cases} \tag{2.11}$$

where *i* is a character index of string *a* and *j* is a character index of string *b*.

Note that the first, second, and third elements in the minimum corresponds to deletion (from *a* to *b*), insertion, and substitution, respectively.

We identify a string of the object as follows :

1 Compute the distance between the test string with every train string using Levenshtein distance.

2 Sort all distances in an increasing order.

3 Select the set of *K* closet distances.

4 The class label of the test string is assigned based on a majority vote of its nearest neighbor

2.7 String Grammar Fuzzy K-Nearest Neighbor

Let a training dataset, which consists of the strings of objects in all classes, be $\mathbf{X} = \left\{ \mathbf{x}_1^1, ..., \mathbf{x}_{N_1}^1, \mathbf{x}_1^2, ..., \mathbf{x}_{N_2}^2, ..., \mathbf{x}_1^C, ..., \mathbf{x}_{N_C}^C, \right\}$ where $\mathbf{x}_i^j$ is a training string of object *i* in class *j*, and N_j is the number of training strings of objects in class *j*. Each string ($\mathbf{x}_i^j$) is a sequence of symbols (primitives). For example, $\mathbf{x}_i^j = \left(x_{i1}^j x_{i2}^j ... x_{il}^j \right)$ is a string with length *l* where each x_{ir}^j is a member of a set $\sum$ defined symbols or primitives ($x_{ir}^j \in \Sigma$ for $i = 1, ..., N_j$, $j = 1, ..., C$, and $r = 1, ..., l$). Please be noted that the number of the strings of all training objects is still *N*. For a test string of object *i*, the Levenshtein distances between strings $\mathbf{x}$ and $\mathbf{x}_i^j$, $Lev\left(\mathbf{x}, \mathbf{x}_i^j\right)$, for $1 \le j \le C$ and $1 \le i \le N_j$ are calculated.

Next, the Levenshtein distance will be using on the equation of membership function, i.e., equation 3.1.

The original algorithms of each equation for our modified algorithms as follows:

1. New possibilistic clustering

From new possibilistic clustering algorithm [24] the membership function as

$$u_{ij} = \exp\left(-\frac{m\sqrt{c}\left\|\vec{x}_j - \vec{a}_i\right\|^2}{\beta} \right) \tag{2.12}$$

where $i = 1,\ldots,C$, $j=1,\ldots,n$

$$\vec{a}_i = \frac{\sum_{j=1}^{n} \mu_{ij}^m \vec{x}_j}{\sum_{j=1}^{n} \mu_{ij}^m} \quad \text{(set of } C \text{ cluster center)}$$

$$\beta = \frac{\sum_{j=1}^{n} \left\| x_j - \bar{x} \right\|^2}{n} \quad \text{with} \quad \bar{x} = \frac{\sum_{j=1}^{n} x_j}{n}$$

2. Fuzzy *K*-nearest neighbor

From a fuzzy *K*-nearest neighbor [8]. Let $W = \left\{\vec{x_1}, \vec{x_2},\ldots, \vec{x_n}\right\}$ be a set of n labeled samples. Also, let $u_i\left(\vec{x}\right)$ be the assigned membership of the vector $\vec{x}$ (to be computed) as

$$u_i\left(\vec{x}\right) = \frac{\sum_{j=1}^{K} u_{ij} \left(1 \Big/ \left\|\vec{x} - \vec{x}_j\right\| \right)^{2/m-1}}{\sum_{j=1}^{K} \left(1 \Big/ \left\|\vec{x} - \vec{x}_j\right\| \right)^{2/m-1}} \tag{2.13}$$

where u_{ij} is the membership in the i^{th} class of the j^{th} vector of the labeled sample set.

3. Possibilistic entropy based clustering

From possibilistic entropy based clustering algorithm [29] the membership function as

$$u_{ij} = e^{\beta_i d_{ij}^2} \tag{2.14}$$

where β_i is computed

$$\beta_i = (1+\alpha)\frac{\sum_{j=1}^{N} u_{ij}\left(d_{ij}\right)^2}{\sum_{j=1}^{N} u_{ij}\left(d_{ij}\right)^4}$$

where $\alpha \geq 0.5$ and d_{ij} is Euclidean distance between the i^{th} data point and the j^{th} cluster centers

4. Fuzzy C-Means clustering

From vector fuzzy C-Means clustering algorithm [30] the membership function as

$$u_{ij} = \frac{1}{\sum_{p=1}^{C} \frac{d_{ij}}{d_{ip}}} \tag{2.15}$$

where d_{ij} is Euclidean distance between the i^{th} data point and the j^{th} cluster centers

5. New fuzzy entropy clustering

From new fuzzy entropy clustering algorithm [31] the membership function as

$$u_{ij} = \sum_{p=1}^{C} e^{-\frac{1}{\gamma}\left(\|x_j - v_p\|^2 - \|x_j - v_i\|^2\right)} \tag{2.16}$$

where $\gamma = 1$ and $v_i = \frac{\sum_{j=1}^{N} u_{ji}^m x_j}{\sum_{j=1}^{N} u_{ji}^m}$

6. Rule generation based clustering

From rule generation based clustering algorithm [32] the membership function as

$$u_{ij} = \exp\left(-\frac{\left(x_j - a_i\right)^2}{2(\sigma)^2}\right) \tag{2.17}$$

where a_i is a set of C cluster center

σ is the standard deviation

7. Possibilistic clustering

From possibilistic clustering algorithm [33] the membership function as

$$u_{ij} = \frac{1}{1+\left(\frac{d_{ij}^2}{\eta_i}\right)^{\frac{1}{m-1}}} \tag{2.18}$$

where $\eta_i = P\frac{\sum_{j=1}^{N} u_{ij}^m d_{ij}^2}{\sum_{j=1}^{K} u_{ia}^m}$ and $P = 1$