3-2020

How to Describe Conditions Like 2-out-of-5 in Fuzzy Logic: a Neural Approach

Olga Kosheleva

Vladik Kreinovich

Nguyen Hoang Phuong

Comments:
Technical Report: UTEP-CS-20-19
To appear in *Journal of Advanced Computational Intelligence and Intelligent Informatics (JACIII)*

How to Describe Conditions Like 2-out-of-5 in Fuzzy Logic: a Neural Approach

Olga Kosheleva[1], Vladik Kreinovich[2], and Hoang Phuong Nguyen[3]
[1]Department of Teacher Education
[2]Department of Computer Science
University of Texas at El Paso
500 W. University
El Paso, TX 79968, USA
olgak@utep.edu,vladik@utep.edu
[3]Division Informatics, Math-Informatics Faculty
Thang Long University
Nghiem Xuan Yem Road
Hoang Mai District
Hanoi, Vietnam, nhphuong2008@gmail.com

Abstract

In many medical applications, we diagnose a disease and/or apply a certain remedy if, e.g., two out of five conditions are satisfied. In the fuzzy case, i.e., when we only have certain degrees of confidence that each of n statement is satisfied, how do we estimate the degree of confidence that k out of n conditions are satisfied? In principle, we can get this estimate if we use the usual methodology of applying fuzzy techniques: we represent the desired statement in terms of "and" and "or", and use fuzzy analogues of these logical operations. The problem with this approach is that for large n, it requires too many computations. In this paper, we derive the fastest-to-compute alternative formula. In this derivation, we use the ideas from neural networks.

1 Formulation of the Problem

Need for fuzzy logic in medical applications. To diagnose a patient, to come up with appropriate cure, it is often important to perform many tests. Each tests results in numerous numerical values that describe the state of the patient: even a routine blood test returns several pages of numbers. These numbers are important and need to be taken into account, but, of course, medical doctors do not just operate based on these numbers, they also use their experience and intuition – if they operated by numbers only, it would have been easy

to replace them with a computer program.

Some medical doctors have more experience and a better intuition, they are more successful in curing the corresponding diseases. Other medical doctors have not yet acquired this experience. It is therefore desirable to incorporate the experience of top medical doctors into a computer-based system so as to help beginning medical doctors make good decisions.

Most top medical doctors are willing and eager to share their knowledge. The problem is that this knowledge does not usually come in terms of numbers – which would then be easy to incorporate in a computer system, this knowledge usually comes in terms of words from natural language, words which are not easy for a computer to understand. For example, a medical doctor may say that a certain treatment is appropriate when the fever is high – without giving a precise definition of what "high" means.

This situation – and similar situations with experts in other fields – motivated Lotfi Zadeh to come up with *fuzzy logic*; see, e.g., [2, 3, 8, 10, 11, 12]. In this approach, for every statement in which an expert is not 100% confident, the expert supplies his/her degree of confidence in this statement – estimated, e.g., by a number on a scale from 0 to 10. Since different experts may use different scale, a reasonable idea is to make the corresponding numbers compatible by reducing them to the interval $[0, 1]$; e.g.:

- 7 on a 0-to-10 scale will be represented by a number $7/10 = 0.7$,
- 3 on a 0-to-5 scale will be represented by a number $3/5 = 0.6$, etc.

In this case:

- if we are absolutely confident that a statement is true, then our degree of confidence will be 1, and
- if we are absolutely confident that a statement is not true, then our degree of confidence will be 0.

Logical operations in fuzzy logic and how we use them. In medicine, decisions are rarely based on one simple opinion or one simple rule. Usually, several expert statements $S_1, \ldots, S_n$ need to be taken into account. In such a case, the practitioner's confidence in this decision is the confidence that all these statements are true, i.e., equivalently, that the combined statement $S_1 \,\&\, \ldots \,\&\, S_n$ is true.

In the ideal world, we should elicit the corresponding degrees of confidence from the experts, but the problem is that based on N statement, we can make $2^N - 1$ combinations corresponding to all possible non-empty subsets of the set $\{S_1, \ldots, S_N\}$, and for a reasonable large N, like several dozen, this number is astronomical. There is no way to ask the expert to estimate the degree of confidence in billions of possible combinations.

Since we cannot elicit the degree of confidence in a complex statement like $A \,\&\, B$ from the expert, we must be able to evaluate it based on the known

degrees of confidence a and b in the statements A and B. An algorithm $f_{\&}(a,b)$ that transforms the degrees of confidence a and b into an estimate for a degree of confidence in $A\,\&\,B$ is known as an *"and"-operation*, or, for historic reasons, a *t-norm*.

There are many possible "and"-operations. Often, a reasonable choice is to follow Zadeh himself and select the simplest possible "and"-operation

$$f_{\&}(a,b) = \min(a,b).$$

Similarly, to estimate the degree of confidence in statements of the type $A \vee B$, we need to have an "or"-operation $f_{\vee}(a,b)$, and for the negation $\neg A$, a negation-operation $f_{\neg}(a)$.

Sometimes, we may want to describe the degree of confidence in a more complex statement, e.g., a statement of the type $(S_1\,\&\,S_2\,\&\,S_3) \vee (S_4\,\&\,S_5)$ that corresponds, e.g., to the case when we have two different situations $S_1\,\&\,S_2\,\&\,S_3$ and $S_4\,\&\,S_5$ under which a given medicine is recommended. In this case, a reasonable idea is:

- to first apply "and"-operations and get the degrees of confidence for $S_1\,\&\,S_2\,\&\,S_3$ and $S_4\,\&\,S_5$, and then
- apply an "or"-operation to combine these two estimates.

In general, if we have a complex statement, a usual practice is to represent this complex statement in an equivalent form that uses "and", "or", and "not", and then sequentially apply the corresponding operations of fuzzy logic.

How can we describe conditions like 2-out-of-5: a challenge. Medicine is filled with statement of the "2-out-of-5" type: if a patient has 2 out of the given 5 symptoms, this means that, most probably, the patient has the corresponding disease. It could be 8 out of 10 etc., but it is a known fact that for many diseases, some of the symptoms may not occur in some patients.

Since such statements are ubiquitous, it is desirable to be able to describe them in fuzzy logic; this need was emphasized, e.g., in [1, 4, 13]. The problem is that for such statements, the above-described usual fuzzy techniques do not work well.

Indeed, in principle, if we have five statements $S_1, \ldots, S_5$, then we can naturally describe the 2-out-of-5 statement in terms of "and" and "or", by explicitly listing all possible pairs of statements, i.e., as

$$(S_1\,\&\,S_2) \vee (S_1\,\&\,S_3) \vee (S_1\,\&\,S_4) \vee (S_1\,\&\,S_5) \vee (S_2\,\&\,S_3) \vee$$

$$(S_3\,\&\,S_4) \vee (S_2\,\&\,S_5) \vee (S_3\,\&\,S_4) \vee (S_3\,\&\,S_5) \vee (S_4\,\&\,S_5).$$

This statement is long and difficult-to-compute, but still doable. However, when we get more symptoms, the resulting "and"-"or"-statement becomes astronomically large, not practically doable.

We need a simpler way to estimate our degree of confidence in such statements. This is what we will do in this paper.

2 Analysis of the Problem

What do we want. We want to have, for each $k < n$, a function $f_{k,n}(a_1, \ldots, a_n)$ that would transform our degrees of confidence a_i in individual statements S_i into our estimate for the degree of confidence that k out of n statements are true.

When we are absolutely sure about each statement, i.e., when each value a_i is equal to 0 or 1, then we should have:

- $f_{k,n}(a_1, \ldots, a_n) = 1$ if at least k values a_i are equal to 1, and
- $f_{k,n}(a_1, \ldots, a_n) = 0$ if fewer than k values a_i are equal to 1.

Also, the very fact that we are simply counting the conditions – and not taking some of them with a larger weight – means that we treat all n conditions as equally important. Thus, it makes sense to require that the value of the desired function will not change if we simply change the order of values. In precise terms, for any permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ and for all possible values $a_1, \ldots, a_n$, we should have

$$f_{k,n}(a_{\pi(1)}, \ldots, a_{\pi(n)}) = f_{k,n}(a_1, \ldots, a_n).$$

If our confidence in one (or more) of the statements increases, then clearly our degree of confidence in the desired statement – that at least k of the statements are true – should also increase (or at least not decrease). Thus, the desired function $f_{k,n}(a_1, \ldots, a_n)$ should be monotonic in all a_i:

$$\text{if } a_i \le a_i' \text{ for all } i, \text{ then } f_{k,n}(a_1, \ldots, a_n) \le f_{k,n}(a_1', \ldots, a_n').$$

What we mean by simplicity. To describe the desired "simple" operation, let us first clarify what we mean by simplicity. The problem with the traditional approach to such statement is that they take too long to compute – sometimes, unrealistically long. So, when we talk about simplicity, we mean the need to make computations faster.

When we estimate the computation time of an algorithm, we need to take into account that nowadays, even the cheapest PC has several processors working in parallel. Thus, it makes sense to count the time needed to compute the corresponding value in parallel.

So, we are looking for parallel algorithms in which we have a sequence of "layers" – computational blocks working in parallel. The fewer layers we have and the faster the computations on each layer, the faster the overall computations. So, to speed up the overall computations, we need to have the smallest possible number of layers and the fastest-to-compute blocks.

Which blocks are the fastest to compute? In mathematical terms, each block computes a function of its inputs. Functions can be linear and nonlinear.

Of course, linear functions are faster to compute, so they will be our first example of fast-to-compute blocks. We will denote such blocks by L, short of "linear".

In many practical situations, it is not sufficient to have linear functions – and later on, we will show that this is exactly one of such situations. Thus, we also need some non-linear blocks. Different non-linear blocks require different computation time. Usually, the more inputs we have, the longer it takes to process them. The fastest-to-compute are the blocks that process only one input. So, in addition to linear blocks, we will also consider non-linear blocks that take one input a and compute some non-linear function $s(a)$ of this input. These blocks will be denoted by NL, short of "non-linear".

Can we have just one layer? If we have only one layer, then this layer must be formed either by NL blocks or by L blocks. We cannot have NL blocks, because they compute a function of one variable, and we need a function of n variables.

Can we have L blocks – i.e., can the desired function be linear? In other words, can we have a function of the type

$$f_{k,n}(a_1, \ldots, a_n) = w_0 + \sum_{i=1}^{n} w_i \cdot a_i$$

for some coefficients w_i? Not really. Indeed, if k values a_i are equal to 1 and the result are equal to 0, then we should get the value 1:

$$f_{k,n}(1, \ldots, 1\,(k \text{ times}), 0, \ldots, 0) = w_0 + \sum_{i=1}^{k} w_i = 1.$$

On the other hand, if we have $k+1$ values equal to 1 and all the other equal to 0, then we should also get the value 1:

$$f_{k,n}(1, \ldots, 1\,(k+1 \text{ times}), 0, \ldots, 0) = w_0 + \sum_{i=1}^{k+1} w_i = 1.$$

Subtracting these two equalities, we conclude that

$$\sum_{i=1}^{k+1} w_i - \sum_{i=1}^{k} w_i = w_{k+1} = 0.$$

Similarly, we can prove that all the weights $w_1, \ldots, w_n$ should be equal to 0 – thus, the function should not depend on the inputs at all and be a constant, which cannot be, since:

- for some inputs, the desired function is equal to 1, while
- for other inputs, the desired function is equal to 0.

So, we cannot have one layer, we need at least two layers.

We cannot have L-L or NL-NL configurations. It is easy to see that we cannot have two consequent layers of the same type. Indeed, if we have two consequent linear layers, this means that:

- first, we apply some linear transformation to the inputs, and then,
- we apply a second linear transformation to the results of the first transformation.

It is well known that a composition of two linear transformations is linear – and we already know that linear functions are not sufficient.

Similarly, if we have two NL layers, then:

- on the first layer, each input a_i is transformed into a value $b_i = s_i(a_i)$ for some function $s_i(a)$ of one variable, and then,
- on the second layer, we apply a non-linear function $t_i(a)$ to the results b_i of the first layer.

As a result, we get $t_i(b_i) = t_i(s_i(a_i))$, which is equivalent to applying a single function $t_i(s_i(a))$ to the inputs a_i. And we already know that this way, we cannot build the desired function.

So, if we have two layers, then these layers must be different: either NL-L or L-NL. Let us analyze these two cases one by one.

Can we have NL-L? In this case:

- first, we apply some non-linear function $s_i(a)$ to each input a_i, and then,
- we form a linear combination of the results $s_i(a_i)$.

In other words, we will have

$$f_{k,n}(a_1, \ldots, a_n) = w_0 + \sum_{i=1}^{n} w_i \cdot s_i(a_i)$$

for some coefficients w_i.

In this case, if k values a_i are equal to 1 and all other values a_i are equal to 0, then we should get the value 1:

$$f_{k,n}(1, \ldots, 1\,(k \text{ times}), 0, \ldots, 0) = w_0 + \sum_{i=1}^{k} w_i \cdot s_i(1) + \sum_{j=k+1}^{n} w_j \cdot s_i(0) = 1.$$

On the other hand, if we have $k+1$ values equal to 1 and all the other equal to 0, then we should also get the value 1:

$$f_{k,n}(1, \ldots, 1\,(k+1 \text{ times}), 0, \ldots, 0) = w_0 + \sum_{i=1}^{k+1} w_i \cdot s_i(1) + \sum_{j=k+2}^{n} w_j \cdot s_i(0) = 1.$$

Subtracting these two equalities, we conclude that $s_{k+1}(1) - s_{k+1}(0) = 0$, i.e., that $s_{k+1}(1) = s_{k+1}(0)$.

Similarly, we can prove that all i from 1 to n, we will have $s_i(1) = s_i(0)$. So, if each a_i is equal to 0 or to 1, the value

$$f_{k,n}(a_1, \ldots, a_n) = w_0 + \sum_{i=1}^{n} w_i \cdot s_i(a_i)$$

should be the same, no matter how many of these values are 0s and how many are 1s. But this cannot be, since:

- for some combinations of 0 and 1 inputs, the desired function is equal to 1, while
- for other combinations of 0 and 1 inputs, the desired function is equal to 0.

So, we cannot have a NL-L configuration either. The only remaining case is the case of L-NL configuration.

Can we have L-NL? In this case:

- we first form a linear combination of the inputs $w_0 + \sum\limits_{i=1}^{n} w_i \cdot a_i$, and then
- we apply a non-linear functions $s(a)$ to this linear combination, resulting in

$$f_{k,n}(a_1, \ldots, a_n) = s\left(w_0 + \sum_{i=1}^{n} w_i \cdot a_i\right).$$

The requirement that the value should not change under any permutation means that all the weights $w_1, \ldots, w_n$ must be equal to each other: $w_1 = \ldots = w_n$. If we denote the common value of the coefficients w_i by w, then the above formula takes the form

$$f_{k,n}(a_1, \ldots, a_n) = s\left(w_0 + \sum_{i=1}^{n} w \cdot a_i\right) = s\left(w_0 + w \cdot \sum_{i=1}^{n} a_i\right).$$

Instead of a function $s(a)$, we can use a different function $t(a) \stackrel{\text{def}}{=} s(w_0 + w \cdot a)$. In terms of this new function, the above expression gets the following simplified form:

$$f_{k,n}(a_1, \ldots, a_n) = t\left(\sum_{i=1}^{n} a_i\right).$$

What can we conclude about the function $t(a)$? We want the value $f_{k,n}(a_1, \ldots, a_n)$ to be between 0 and 1, so we should have $t(a) \in [0, 1]$ for all a. Monotonicity implies that the function $t(a)$ is monotonic: if $a \le a'$, then $t(a) \le t(a')$. Also:

- If at least k values a_i are equal to 1, then we must get 1. In this case, the sum $\sum\limits_{i=1}^{n} a_i$ is equal to k or larger. So, for any value $a \geq k$ we should have

$$t(a) = 1.$$

- On the other hand, if fewer than k values a_i are equal to 1, and other values are equal to 0, then we should get 0. In this case, the sum $\sum\limits_{i=1}^{n} a_i$ is equal to $k-1$ (or to a smaller number). Thus, we should have $t(k-1) = 0$ – and, by monotonicity, we should have $t(a) = 0$ for all $a \leq k-1$.

Now, we are ready to present our final result.

3 Main Result: The Simplest Possible Fuzzy Analogue of the 2-out-of-5-Type Operations

General conclusion. Suppose that we know the degrees of confidence $a_1, \ldots, a_n$ in n statements $S_1, \ldots, S_n$, and for some integer $k < n$, we want to estimate our degree of confidence $f_{k,n}(a_1, \ldots, a_n)$ that at least k of these statements are true. In this case, we should take

$$f_{k,n}(a_1, \ldots, a_n) = t\left(\sum_{i=1}^{n} a_i\right),$$

where:

- for $a \leq k-1$, we have $t(a) = 0$;
- for $a \geq k$, we have $t(a) = 1$, and
- for a between $k-1$ and k, the function $t(a)$ increases from 0 to 1.

Which of these operations should we choose? All we need to do is to choose the values of the function $t(a)$ for the interval $[k-1, k]$; for all other values, its values are determined already. As we have mentioned earlier, the simplest possible functions are linear, so we should take $t(a) = c_0 + c_1 \cdot a$. From the conditions that $t(k-1) = 0$ and $t(k) = 1$, we conclude that $c_0 = -(k-1)$ and $c_1 = 1$. Thus, on this interval, we should take $t(a) = a - (k-1)$. Thus, we arrive at the following operation $f_{k,n}(a_1, \ldots, a_n)$:

- when $\sum\limits_{i=1}^{k} a_i \leq k-1$, we have $f_{k,n}(a_1, \ldots, a_n) = 0$;
- when $k-1 \leq \sum\limits_{i=1}^{k} a_i \leq k$, we have $f_{k,n}(a_1, \ldots, a_n) = \sum\limits_{i=1}^{n} a_i - (k-1)$; and

- when $k \le \sum\limits_{i=1}^{k} a_i$, we have $f_{k,n}(a_1, \ldots, a_n) = 1$.

These three cases can be covered by a single formula

$$f_{k,n}(a_1, \ldots, a_n) = \min\left[\max\left(0, \sum_{i=1}^{n} a_i - (k-1)\right), 1\right].$$

Comment. It is worth mentioning that for $k = n$, we get the exact same expression as when we repeatedly apply the "and"-operation

$$f_{\&}(a, b) = \max(a + b - 1, 0)$$

to the values a_i:

- first, we compute the estimate a_{12} for the degree of confidence in

  $$A_{12} \stackrel{\text{def}}{=} A_1 \,\&\, A_2$$

  as $a_{12} = f_{\&}(a_1, a_2)$;

- then, we compute the estimate a_{123} for the degree of confidence in

  $$A_{123} \stackrel{\text{def}}{=} A_1 \,\&\, A_2 \,\&\, A_3 \equiv A_{12} \,\&\, A_3$$

  as $a_{123} = f_{\&}(a_{12}, a_3)$,

- $\ldots$,

- finally, we compute the estimate $a_{1\ldots n}$ for the degree of confidence in

  $$A_{1\ldots n} \stackrel{\text{def}}{=} A_1 \,\&\, \ldots \,\&\, A_n \equiv A_{1\ldots n-1} \,\&\, A_n$$

  as $a_{123} = f_{\&}(a_{1\ldots n-1}, a_n)$.

Why neural networks? All this may sound reasonable, but why do we mention neural approach in the title? The explanation is simple: the same arguments have been used to explain the success of traditional neural networks; see, e.g., [5, 6, 7]. The main difference is that in a neural network, we want to be able to approximate any function, in which case we cannot have a 2-layer arrangement, the simplest we get is a 3-layer arrangement L-NL-L:

- first, we apply some linear transformations to the inputs $x_1, \ldots, x_n$, resulting in values $z_k = w_{k0} + \sum\limits_{i=1}^{n} w_{ki} \cdot x_i$;

- then, we apply a non-linear transformation to z_k, resulting in $y_k = s_k(z_k)$, and

- finally, we apply a linear transformation to the results y_k, getting

$$y = \sum_{k=1}^{K} W_k \cdot y_k + W_0,$$

where K is the overall number of neurons on the first (L) layer.

As a result, we get the usual formula describing how, for a traditional neural network, its output y depends on the inputs $x_1, \ldots, x_n$:

$$y = \sum_{k=1}^{K} W_k \cdot s_k \left(w_{k0} + \sum_{i=1}^{n} w_{ki} \cdot x_i \right) + W_0.$$

Other possible applications of this idea. We can use the same idea to describe how to generalize the number of elements in a set (also known as the set's cardinality) to fuzzy sets, in which we have n elements with degree of membership $a_1, \ldots, a_n$. In this case, one linear layer is sufficient, and, by taking into account that when all a_i are 0 or 1, we should get the actual cardinality, we get Zadeh's formula $\sum\limits_{i=1}^{n} a_i$.

Acknowledgments

This work was supported in part by the US National Science Foundation grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science) and HRD-1242122 (Cyber-ShARE Center of Excellence).

The authors are thankful to all the participants of the International Conference on Artificial Intelligence and Computational Intelligence AICI'2020 (Hanoi, Vietnam, January 4–6, 2020) for valuable discussions.

References

[1] K.-P. Adlassnig, "Fuzzy methods in medical research and patient care", *Abstracts of the International Conference on Artificial Intelligence and Computational Intelligence AICI'2020*, Hanoi, Vietnam, January 4–6, 2020, pp. 8–9.

[2] R. Belohlavek, J. W. Dauben, and G. J. Klir, *Fuzzy Logic and Mathematics: A Historical Perspective*, Oxford University Press, New York, 2017.

[3] G. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic*, Prentice Hall, Upper Saddle River, New Jersey, 1995.

[4] W. Koller, A. Rappelsberger, B. Willinger, G. Kleinoscheg, and K.-P. Adlassnig, "Artificial Intelligence in infection control—healthcare institutions need intelligent information and communication technologies for surveillance and benchmarking", In: V. Kreinovich and Nguyen Hoang Phuong (eds.), *Soft Computing for Biomedical Applications and Related Topics*, Springer Verlag, to appear.

[5] V. Kreinovich, "From traditional neural networks to deep learning: towards mathematical foundations of empirical successes", In: S. N. Shahbazova, J. Kacprzyk, V. Emilia Balas, and V. Kreinovich (eds.), *Proceedings of the World Conference on Soft Computing*, Baku, Azerbaijan, May 29–31, 2018.

[6] V. Kreinovich and A. Bernat, "Parallel algorithms for interval computations: an introduction", *Interval Computations*, 1994, No. 3, pp. 6–62.

[7] V. Kreinovich and O. Kosheleva, "Deep learning (partly) demystified", *Proceedings of the 2020 4th International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence ISMSI'2020*, Thimpu, Bhutan, March 21–22, 2020, to appear.

[8] J. M. Mendel, *Uncertain Rule-Based Fuzzy Systems: Introduction and New Directions*, Springer, Cham, Switzerland, 2017.

[9] H. T. Nguyen, V. Kreinovich, and P. Wojciechowski, "Strict Archimedean t-norms and t-conorms as universal approximators", *International Journal of Approximate Reasoning*, 1998, Vol. 18, Nos. 3–4, pp. 239–249.

[10] H. T. Nguyen, C. L. Walker, and E. A. Walker, *A First Course in Fuzzy Logic*, Chapman and Hall/CRC, Boca Raton, Florida, 2019.

[11] V. Novák, I. Perfilieva, and J. Močkoř, *Mathematical Principles of Fuzzy Logic*, Kluwer, Boston, Dordrecht, 1999.

[12] L. A. Zadeh, "Fuzzy sets", *Information and Control*, 1965, Vol. 8, pp. 338–353.

[13] J. Zeckl, M. Wastian, D. Brunmeir, A. Rappelsberger, S. B. Arseniev, and K.-P. Adlassnig, "From machine learning to knowledge-based decision support—a predictive-model-markup-language-to-Arden-syntax transformer for decision trees", In: V. Kreinovich and Nguyen Hoang Phuong (eds.), *Soft Computing for Biomedical Applications and Related Topics*, Springer Verlag, to appear.