Abstract

The theory of fuzzy measures has a great potential for real world applications, but limited by the lack of suitable identifying methods. This research proposes a bisection algorithm based on Sugeno’s λ-fuzzy measures. The proposed method is simple enough to suit the practical applications for the required data is similar to the traditional weighted-sum method. The computing complexity of this method is O(n), and it is efficient to meet the huge computations in practice.

Keywords: Fuzzy measure; bisection algorithm; computing complexity

1. Introduction

The theory of fuzzy measures has a great potential for applications of subjective evaluation, information fusion, multiple criteria decision making [Wang and Klir, 1992; Grabisch, 1995]. However, this potential has not been fully utilized due to the lack of identifying methods for constructing fuzzy measure from empirical data [Yuan and Klir, 1996]. The crux is the amount of required coefficients growing exponentially with problem size n (roughly 2^n). The existed identifying methods are based on either learning data, or semantic estimations, or both, but this problem is not yet solved in a fully satisfactory way [Grabisch, 1995].

Sugeno proposed a λ-fuzzy measure satisfying the λ-additive axiom [Sugeno and Terano, 1977; Wang and Klir, 1992]. The λ-fuzzy measure reduces the difficulty of identification effectively, and has plenty applications recently, including pattern recognition, speaker verification, and public attitude analyzing. Some studies estimate this single parameter of λ-fuzzy measure from learning data by the soft-computing methods like genetic algorithm [Lee and Leekwang, 1995], neural networks [Wang and Wang, 1997]. But collecting subjective evaluations of each information source by questionnaire is an easier approach, and this approach reduces the identifying problem to an n-1 degree polynomial (see function G in fig. 1 and explanation of section 2 for detail). There are many available methods [Wierzchon, 1983], and the Keller and Osborns’ Newton’s method seems the simplest among them for practical uses.

However, the Newton’s method is sensitive to initial solution, and a feasible initial solution is not easy to locate in the polynomial of figure 1. A bad initial solution of G(λ)≈1 will lead the positive or negative infinite; and G(λ)<1 when S<1, G(λ)>1 when S>1 will mislead the searching sequence back to the trivial solution λ=0. Besides, an over-estimated initial solution causes a slow converging sequence. The last, the Newton’s method requires the first-order differentiation having computing complexity O(n^2).

This research proposes a simple method based on bisection search and a linear transformation of traditional one (see function H in fig. 2 and explanation of section 2 for detail). The properties of this method are listed below. (1) The input is simple as the traditional weighted-sum method, and the required data is n only. (2) The executing time is short in practice and increases linearly with problem size only. A analysis of computing complexity O(n) is given in section 4. (3) The robustness is guaranteed and discussed in section 5. (4) The implementation is easy, and the source code of an executable program is opened in the appendix.
2. \(\lambda\)-Fuzzy Measures

In this section, some notations and required properties for our method are given. Most of them have been discussed deeply in some pioneering studies. We refer the reader to Leszczyński et al. [1986], Wang and Klir [1992].

**Definition 2.1.** Let \(X=\{x_1, x_2, \ldots, x_n\}\) be a nonempty finite set, \(P\) is the power set of \(X\). A (regular) \(\lambda\)-fuzzy measure \(\mu\) defined on \((X, P)\) is a set function satisfying the conditions:

1. \(\mu(\emptyset)=0\), \(\mu(X)=1\), where \(\emptyset\) is the empty set; (boundary conditions)
2. If \(A, B \in P\) and \(A \cup B = \emptyset\) then \(\mu(A \cup B)=\mu(A)+\mu(B)+\lambda \mu(A)\mu(B), \lambda \in (-1, +\infty)\); (monotonicity)

**Proposition 2.2.** Denote \(g_i=\mu(x_i)\), the fuzzy measure \(\mu\) satisfies the bounding conditions:

1. \(g_i \in [0, 1]\) for all \(i\);
2. if there exists \(g_i=1\), then \(g_j=0\) for any \(j \neq i\);
3. if \(g_i<1\) for all \(i\), then there are at least two of them being positive.

Extending definition 2.1, we obtain an equation for identifying parameter \(\lambda\):

\[
G(\lambda) = \begin{cases} 
\prod_{i=1}^{n} (1+\lambda g_i) = 1+\lambda & \lambda \neq 0 \\
1 & \lambda = 0 
\end{cases}
\]

**Proposition 2.3.** Denote \(S = \sum g_i\), if \(\lambda \in (-1, +\infty)\) then \(G(\lambda)>0\), \(G'(\lambda)>0\), and \(G(0)=S\) for \(n \geq 2\) (lemma 2.4 and corollary 2.5 of Leszczyński et al. 1986).

**Theorem 2.4.** The parameter \(\lambda\) can be determined uniquely from \(G(\lambda)\) (theorem 3.6 of Wang and Klir, 1992):

\[
\begin{align*}
\lambda &> 0 \quad \text{when } S < 1 \\
\lambda &= 0 \quad \text{when } S = 1 \\
-1 < \lambda < 0 & \quad \text{when } S > 1
\end{align*}
\]

**Proposition 2.5.** Let \(H(\lambda)=G(\lambda)-1-\lambda=0\).

1. When \(S>1\), if exist \(u, v\), \(-1<u<v<0\), and \(H(u)>0, H(v)<0\), then there exists a unique \(w\), such that \(H(w)=0\) and \(u<w<v\).
2. When \(S<1\), if exist \(u, v\), \(u>v>0\), and \(H(u)>0, H(v)<0\), then there exists a unique \(w\), such that \(H(w)=0\) and \(u<w<v\).

[Proof] From the continuity of function \(H\) and uniqueness of theorem 2.4, if we can obtain a closed range having the two endpoints with different signs, then a solution exists uniquely from intermediate theory (see fig. 2).

**Proposition 2.6.** If there exists a small \(\epsilon\), and \(c_0\) such that \((1-c_0)(1-g)\in(1-\epsilon, 1)\), then \(H(1-c_0)-H(1)=\epsilon(1-g)>0\) when all \(c_0<2\), then return error.

**3. The Algorithm**

Fig. 3 shows the proposed algorithm. There are three primary steps consisting of validating the input data, determining a closed range of \(\lambda\), and conducting a bisection search.

**Validate the input data**

1-1. If \(n<2\), then return error
1-2. If any \(g_i<0\) or \(g_i>1\) for \(i=1..n\), then return error
1-3. If any \(g_i<\epsilon^{1/2}\) for \(i=1..n\), then let \(g_i=0\)
1-4. Let \(c_0\) be the count of \(g_i=0\), \(c_1\) be the count of \(g_i=1\), for \(i=1..n\)
1-5. If \(c_1>1\), then return error
1-6. If \(c_1=1\) and \(c_0<n-1\) then return error
1-7. If \(n-c_0<2\), then return error

**Determine a closed range of \(\lambda\) (S is the summation of \(g_i\))**

2-1. If \(|S-1|<\epsilon\), then return 0
2-2. If \(S>1\), then \(\{\text{let } u=-1, v=-\epsilon, \text{and go to step } 3\}\)
2-3. If \(S<1\), then find a \(k\geq1\) and \(H(2^k)>0\);
    Let \(k=0\); repeat let \(k=k+1\) until \(H(2^k)>0\)
    Let \(v=\epsilon, u=2^k\)
4. Computing Complexity

There are two explicit loops and one implicit loop in this method. The first explicit loop is to find the range of $\lambda$ when $S<1$ (step 2-3). The second explicit loop is to conduct a bisection search within a closed range (step 3). The implicit loop is to compute function $H$ each explicit loop. The following discussions begin with the implicit loop.

1) THE IMPLICIT LOOP OF COMPUTING FUNCTION $H$

By definition, $H(\lambda) = (1+\lambda * g_1) * (1+\lambda * g_2) * ... *(1+\lambda * g_n) - 1 - \lambda$. Let $c_1$ be the fixed overhead, and roughly $c_2$ be the executing time of each input; then, we can denote the executing time of function $H$ as $O_1(n)$ below.

$$O_1(n) = c_1 + c_2n$$

2) THE EXPLICIT LOOP OF FINDING A CLOSED RANGE

If $S<1$, this loop can be stated as finding a $k$ and $H(\lambda_k)>0$. If $S>1$, then $k=0$. Whatever $S$ is, the value of $k$ is predefined as a specific set of $g_i$ is given. We denote the executing time by $O_2(n)$ below. There are two constants in $O_2(n)$, say $c_3$, $c_4$, where $c_3$ is the fixed parts, and $c_4$ is the executing time in the loop excluding the computing time of function $H$.

$$O_2(n) = c_3 + (c_4 + c_1 + c_2n) * (k + 1)$$

3) THE EXPLICIT LOOP OF CONDUCTING A BISECTION SEARCH

There are 2 normal conditions to halt this loop, including the absolute width of searching range $|(u-v)|$ is smaller than $e_2$, or the distance of $w$ under the precisions $d=0.0001$, $e=10^{-12}$. The sequence of $n$ is 2, 3, 4, 5, 6, 7, 8, then, 16, 32, at last, 64. The values of $S$ are discussed in two ways. (1) If $S<1$, then $S$ is distributed uniformly in 4 absolutely range: $0.8 \pm 0.2$, $0.4 \pm 0.1$, $0.2 \pm 0.05$, and $0.1 \pm 0.025$. (2) If $S>1$, then $S$ is distributed uniformly in 4 relative range: $1+\lambda * (0.8 \pm 0.2)$, $1+\lambda * (0.4 \pm 0.1)$, $1+\lambda * (0.2 \pm 0.05)$, and $1+\lambda * (0.1 \pm 0.025)$.

5. Experimental Analysis

1) THE EXPERIMENTAL DATA

Thousands of testing data are generated randomly in this section, and controlled by $n$ (problem size) and $S$ (summation of input). In a random data set, the value of $S$ is randomly determined within a range firstly, and then another process distributes this summation to each input ($g_i$) randomly again. For example, in the combination of $n=2$ and $S=0.8 \pm 0.2$, we get the value of $S$ randomly from $(0.6, 1)$ at first. and then, we distribute this value of $S$ to $g_1$, $g_2$ randomly.

Table 1 shows the results of each combination $n$ and $S$ under the precisions $d=0.0001$, $e=10^{-12}$. The sequence of $n$ is 2, 3, 4, 5, 6, 7, 8, then, 16, 32, at last, 64. The values of $S$ are discussed in two ways. (1) If $S<1$, then $S$ is distributed uniformly in 4 absolutely range: $0.8 \pm 0.2$, $0.4 \pm 0.1$, $0.2 \pm 0.05$, and $0.1 \pm 0.025$. (2) If $S>1$, then $S$ is distributed uniformly in 4 relative range: $1+\lambda * (0.8 \pm 0.2)$, $1+\lambda * (0.4 \pm 0.1)$, $1+\lambda * (0.2 \pm 0.05)$, and $1+\lambda * (0.1 \pm 0.025)$.

<table>
<thead>
<tr>
<th>$\lambda^2$</th>
<th>$S=1\pm(e_1)$</th>
<th>$S=1\pm(e_2)$</th>
<th>$S=1\pm(e_3)$</th>
<th>$S=1\pm(e_4)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^{-12}$</td>
<td>0.00001</td>
<td>0.00002</td>
<td>0.00003</td>
<td>0.00004</td>
</tr>
<tr>
<td></td>
<td>0.0001</td>
<td>0.001</td>
<td>0.01</td>
<td>0.1</td>
</tr>
<tr>
<td>$S=1\pm(e_1)$</td>
<td>0.00001</td>
<td>0.00002</td>
<td>0.00003</td>
<td>0.00004</td>
</tr>
<tr>
<td>$S=1\pm(e_2)$</td>
<td>0.0001</td>
<td>0.001</td>
<td>0.01</td>
<td>0.1</td>
</tr>
<tr>
<td>$S=1\pm(e_3)$</td>
<td>0.00000</td>
<td>0.00001</td>
<td>0.00002</td>
<td>0.00003</td>
</tr>
<tr>
<td>$S=1\pm(e_4)$</td>
<td>0.00000</td>
<td>0.00001</td>
<td>0.00002</td>
<td>0.00003</td>
</tr>
</tbody>
</table>

Table 1. The $\lambda^2$ of fixed data set, and averages of $\lambda^2$, $H^1$, $|H|$ under various combinations of $S$ and $n$ in 30 random tests.
Each combination is tested 30 times randomly, and the average of $\lambda$, |H|, and H# are reported. Beside the random data, the value of $\lambda$ of a fixed data set is given. The purpose of fixed data is verification. Each input of fixed data has the same value; and the sum of fixed data is equal to the middle point of random range.

(2) THE ANALYSIS OF RELIABILITY

This section discusses the reliability in two ways. The first is to compare the $\lambda$ of known data set with the computing result outside the method. The other is to verify the results satisfying the identification equation $H(\lambda)=0$ or not. The required precision of $\lambda$ is $d=0.0001$ in Table 1.

There are eight combinations of $n=2$ in the first row of Table 1, and four numbers in each combination; the first of the four numbers is the $\lambda$ of fixed data set. Usually, this algorithm is too complicated to $n=2$; but it is easy to verify the results manually. For example, the contents of the fixed data are $S=1.8$, and $g_1=g_2=0.9$ in the combination of $n=2$ and $S=1+(n-1)*(0.8+0.2)$, and $S=0.1$, and $g_1=g_2=0.05$ in $n=2$ and $S=0.1+0.025$. We can compare the values in the Table 1 with the “true” values below; and both the results are bounded in $-0.98761+0.0001$, and $359.99994+0.0001 (d=0.0001)$. $0.9+0.9+\lambda*0.9*0.9=1$ $\lambda=-0.8/0.81=-0.9876$ $0.05+0.05+\lambda*0.05*0.05=1$ $\lambda=0.9/0.0025=360$

In another way, we can compute the values of function H; and both errors, $H_1=g_1=0.9(-0.98761)=-0.00035$, $H_2=g_2=0.05(359.99994)=0.000054$, are relatively small. The last number of each combination is the average of |H|. All the values in the Table 1 are very small, from 0.000002 to 0.000089; that is, the values of $\lambda$ are quite reliable.

6. Conclusions

A bisection algorithm given in this paper has been used successfully to identify $\lambda$-fuzzy measure. The amount of required data is small, and as same as the problem size, $n$. The execution of the method is short in practice, and increases with $n$. An analysis of computing complexity $O(n)$ is given, and the reliability is shown by thousands of samples. The robustness is discussed in two way, the data exceeding the ability of computing systems are screened in the validating step, and the rests are demonstrated reliable through four types of specially designed data. The implementation of this method is easy and effective.

7. References