Theoretical principles and implementation issues of fuzzy GUHA association rules

Martin Ralbovský

KIZI FIS VŠE

@ KEG 21.5.2009

Preliminaries

The GUHA method

- Method of exploratory data analysis
- Automatic verification of hypotheses
- Hypotheses viewed as formulas of logical calculus
- Statistical aspects
- Implementation use *bit string approach*

My thesis

Application of the “fuzzy paradigm” to the GUHA method

Aspects of concern:

- Association rules
- Fuzzy data
- Comparison to *mainstream* (active area)
- Implementation

Fuzzy paradigm:

- Fuzzy set theory
- Fuzzy logic

Content of the presentation

1. What is a fuzzy association rule?
2. Fast implementation of fuzzy bit strings

What is an association rule?

The *mainstream* (Agrawal, apriori, itemset…) look

- set theory
- couple of “bound” itemsets (A -> B)
- support / confidence

The GUHA look

- observational logic
- association rule is a formula
- generalized quantifier

What is a fuzzy association rule?

The mainstream look

- no broadly accepted precise definition
- different authors use different definitions

The GUHA look

- the past approaches to make GUHA fuzzy did not concentrate on association rules
- yet do be done

Theoretical models of fuzzy association rules

- Theoretical apparatus answering the question "What is a fuzzy association rule"
- Fuzzy set theoretic – based
- Fuzzy logic based
- Five theoretical models identified in the literature, all of them based on fuzzy set theory

Linguistic terms model

- Most simple form
- Antecedent and consequent contain only 1 item

old_person -> high_blood_pressure

- How is the market basket analysis motivation applied?

Quantitative derived model

- Quantitative association rules
- Variables X and Y defined on completely ordered domains

$$A \Rightarrow B : X \in A = [x_1, x_2] \Rightarrow Y \in B = [y_1, y_2]$$

- Intervals are replaced by fuzzy sets

- What if we do not have completely ordered domains?
- How can we do market basket analysis?

Kuok's model

- The database contains attributes (columns)
- For each attribute, an associated set of fuzzy sets is defined
- X and Y are sets of attributes

$$\text{If} \quad X = \{x_1, x_2, \ldots, x_p\} \quad \text{is} \quad A = \{f_1, f_2, \ldots f_p\}$$

$$\text{then} \quad Y = \{y_1, y_2, \ldots, y_q\} \quad \text{is} \quad B = \{g_1, g_2, \ldots g_q\}$$

$$\text{where } f_i \in \{\text{attributes related to } x_i\} \text{ and } g_j \in \{\text{attributes related to } y_j\}.$$

- What is *is* ?
- How should the conjunction of attributes be interpreted – crisp/fuzzy?

Fuzzy transaction-based model

- Set of items I, fuzzy transaction τ is a nonempty fuzzy subset of I.
- For given item, τ(i) notes degree of membership of item i in transaction τ
- Degree of inclusion of itemset I_0 in a fuzzy transaction

$$\widetilde{\tau}(I_0) = \min_{i \in I_0} \widetilde{\tau}(i)$$

- Fuzzy association rule A -> C holds if

$$\forall \widetilde{\tau} \in T : \widetilde{\tau}(A) < \widetilde{\tau}(C).$$

- One transaction spoils the others
- All transactions need to support the rule

Gradual rules model

- The model provides an alternative look on fuzzy association rules
- Association rule can be viewed as a set of elementary fuzzy implications enhanced with probabilities

My approach

- Define fuzzy set theoretic model of association rules inspired by the GUHA method
- Compare the new model to other models
- Define logical calculus to represent fuzzy association rule

Fuzzy logical model – data matrix

- A novel theoretical model in fuzzy set theory
- The basic building structures are data matrices.

object	f_1	f_2	$\dots$	f_k
o_1	$f_1(o_1)$	$f_2(o_2)$	$\dots$	$f_k(o_k)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
o_m	$f_1(o_m)$	$f_2(o_m)$	$\dots$	$f_k(o_m)$

- Functions mapping objects into some sets (patients and their characteristics)
- The functions have arbitrary ranges

Fuzzy logical model - categorization

- Data are crisp, mapping concepts of natural language to exact mathematical domains should be fuzzy
- Categories are fuzzy sets defined on ranges of f_i's
- Results – *attribute with fuzzy categories*

Example:

object is a patient, f_i is age and categories are fuzzy sets defined on range of f_i (set of ages)

Fuzzy logical model – fuzzy attribute

- *Fuzzy item* – one category of an attribute with fuzzy categories
- *Basic fuzzy attribute* – several categories of an attribute with fuzzy categories connected by a t-conorm
- *Fuzzy attribute* – fuzzy item and basic fuzzy attributes are fuzzy attributes, moreover a t-norm, t-conorm of two fuzzy attributes and negator of a fuzzy attribute is again a fuzzy attribute

Fuzzy logical model – association rule

- Association rule is of form $\alpha \approx \beta$,

  where α and β are fuzzy attributes and ≈ is a *4ft-quantifier* computed on the basis of fuzzy four-fold contingency table (rational values)

$$a = \sum_{o \in \mu} T(\sigma(o, \alpha), \sigma(o, \beta))$$

$$b = \sum_{o \in \mu} T(\sigma(o, \alpha), N(\sigma(o, \beta)))$$

$$c = \sum_{o \in \mu} T(N(\sigma(o, \alpha)), \sigma(o, \beta))$$

$$d = \sum_{o \in \mu} T(N(\sigma(o, \alpha), N(\sigma(o, \beta)))$$

M	β	$\neg\beta$
α	a	b
$\neg\alpha$	c	d

Admissible operator problem

- For given object, a+b+c+d of the table must be equal to 1
- Using standard negator N(x) =1-x: solution is product t-norm T(x,y) = xy
- Using other negators – open problem
- Disjunction – algebraic product S(x,y) = x + y – xy, because of De Morgan laws

Comparison of models

- Association rule of each theoretical model except of fuzzy-transaction based can be transformed to fuzzy logical model
- The fuzzy logical model enables the broadest expressivity of the antecedent and consequent
- The fuzzy logical model lacks drawbacks of other models
- Evaluation of the rule – contingency table opposed to predefined measures – no fuzzy measures needed

LCFAR

- A collection of logical calculi for the fuzzy association rules named *logical calculi of fuzzy association rules* (*LCFAR*)defined
- Fuzzy counterpart of *logical calculi of association rules*
- Proven that association rules of fuzzy logical model can be transformed to LCFAR
- The existence of deduction rules in LCFAR examined in depth

Bit string approach – crisp version

Characteristics of examined objects are encoded as bit strings, this enables

- Fast computation – 32 or 64 operations in one processor instruction
- Coefficients – a complex way of tuning the association rule task (not present in mainstream implementations)

Fuzzy bit strings

- Which structures to use for best performance of fuzzy bit strings
- Which algoritms to use …
- Is there any hardware support?

Limitations:

Ferda + .NET Framework (+ alternatives)

Possible data types

Data Type	Number of bits	Possible values
Byte	8	0 to 255
UInt16	16	0 to 65535
UInt32	32	0 to 4294967295
UInt64	64	0 to 0 18446744073709551615
Float	32	-3.402823E38 to 3.402823E38
Double	64	-1.7976931348623E308 to 1.7976931348623E308

UInt16 vs. Float: Float

- No overflow checking, multiplication of two UInt16 numbers: 6 bitwise shifts, one (integer) multiplication and 5 copy operations
- Conversion from and to float

SIMD

- Single instruction, multiple data operations
- Performing one arithmetic operation on a 128 bit register (4 floats)
- SSE instruction set of x86 and x64 architectures
- Not supported in the .NET architecture, only in Mono
- Bright future – SSE4 instruction set (Core i7 Nehalem)

Experiments

1. What is the best algorithm to use for implementation of fuzzy bit string connectives?
2. Does Mono framework with support of SIMD instructions outperform the prevalent .NET framework?
3. How much slower are the operations on fuzzy bit string compared to operations on crisp bit strings?

Algorithms

Tested algorithms groups:

- Crisp, fuzzy, crisp – fuzzy conjunction
- Crisp, fuzzy, crisp – fuzzy disjunction
- Crisp, fuzzy negation
- Crisp, fuzzy sum

Altogether 55 algorithms and their modifications (safe/unsafe, with/without static variables or dynamic allocation)

Acknowledgement to Michal Kováč for valuable ideas and help

Example – precomputed crisp sum ulong a.k.a “Tschernosterova finta”

```
byte[] bitcounts = new byte[65536];
unsafe uint BoolPrecomputed(ulong[] r)
{
	uint result = 0;
	fixed (ulong* arrayPtr = r)
	{
		fixed (Byte* lookup = bitcounts)
		{
			ulong* currentPtr = arrayPtr;
			ulong* stopPtr = arrayPtr + r.Length;
			while (currentPtr < stopPtr)
			{
				ulong current = *currentPtr++;
				result += *(lookup + (uint)(current & 65535));
				result += *(lookup + (uint)((current >> 16) & 65535));
				result += *(lookup + (uint)((current >> 32) & 65535));
				result += *(lookup + (uint)(current >> 48));
			}
		}
	}
	return result;
}
```

Example 2 – Hamming weight algorithm Boolquick with Vector2ul

```
static unsafe uint QuickVectorSum(Vector2ul[] r)
    {
      Vector2ul M1 = new Vector2ul(0x5555555555555555, 0x5555555555555555);
      Vector2ul M2 = new Vector2ul(0x3333333333333333, 0x3333333333333333);
      Vector2ul M4 = new Vector2ul(0x0f0f0f0f0f0f0f0f, 0x0f0f0f0f0f0f0f0f);
      Vector4ui H01 = new Vector4ui(0x01010101, 0x01010101, 0x01010101, 0x01010101);
      Vector4ui result = new Vector4ui(0, 0, 0, 0);
      fixed (Vector2ul* ur = r)
      {
        Vector2ul* a = ur, kon = ur + r.Length;
        while (a < kon)
        {
          Vector2ul x = *a++;
          x -= (x >> 1) & M1;            //put count of each 2 bits into those 2 bits
          x = (x & M2) + ((x >> 2) & M2); //put count of each 4 bits into those 4 bits
          x = (x + (x >> 4)) & M4;       //put count of each 8 bits into those 8 bits
          result += ((((Vector4ui)x) * H01) >> 24);      //returns left 8 bits of x + (x<<8) + (x<<16) + (x<<24) + ... */
        }
      }
      return result.X + result.Y + result.W + result.Z;
    }
```

Computers

- 6 Windows and 1 Linux computers
- Performance ranging from 3GHz Pentium dual-core processor with 64 bit system to 1,2 GHz Pentium III
- Unfortunately no AMD processor
- Various SSE versions supported

Experiments setup

- Simple benchmarking framework by John Skeet used
- Each operation carried out 10000 times
- Operations on bit strings containing 6400000 bits (crisp or fuzzy)
- Each test was run twice
- On Windows machines, test was run both for .NET Framework and Mono

- Total time 553 hours, 11 minutes, 59 seconds

Experiments - results

- In each algorithm group (fuzzy conjunction …) algorithms were ordered according to their times - ranked
- The fastest algorithm for each group and each framework was the algorithm with highest average rank on all computers
- The times for highest ranking algorithms of .NET Framework and were compared …

Practical .NET/Mono performance

Algorithm group	I	II	III	IV	V	VI	Average
Fuzzy conjunction	0.97	1.52	1	1	0.96	1.30	**1.05**
Crisp – fuzzy conjunction	0.97	1.52	0.67	0.84	0.85	0.68	**0.92**
Fuzzy negation	0.90	1.01	1	0.95	0.97	1.17	**1**
Fuzzy disjunction	4.28	2.52	5.8	4.80	4.77	7.87	**5.01**
Crisp – fuzzy disjunction	2.93	2.88	2.28	2.84	2.91	3.15	**2.83**
Fuzzy sum	1.25	1.22	1.08	1.04	1.03	0.65	**1.05**

Table 8.37: Ratio of the highest ranked algorithms overall .NET/Mono

Performance ratio .NET/Mono, comparable algorithms

Algorithm group	$\sum$ .NET	$\sum$ Mono	Ratio
Crisp conjunction	0:01:15	0:01:22	**0.92**
Fuzzy conjunction	2:04:48	0:56:46	**2.20**
Crisp – fuzzy conjunction	4:12:34	5:45:43	**0.74**
Crisp negation	0:00:41	0:00:41	**1**
Fuzzy negation	1:25:38	0:43:57	**1.95**
Crisp disjunction	0:01:15	0:01:21	**0.92**
Fuzzy disjunction	4:49:06	0:58:13	**4.97**
Crisp – fuzzy disjunction	44:38:26	38:01:06	**1.17**
Crisp sum	0:03:29	0:05:51	**0.59**
Fuzzy sum	1:05:12	0:49:55	**1.31**

Table 8.38: Sum of running times of comparable algorithms .NET/Mono

Crisp – fuzzy slowdown

Connective	I	II	III	IV	V	VI	VII	Average
Conjunction	52.1	115.2	141	48.8	29.8	34.6	46	**66.2**
Negation	130	140	105	102	136	31.1	33	**96.7**
Disjunction	60.2	277	842	222.3	150	222.7	282	**293**
Sum	13.1	14.5	14.8	12.9	11.9	11.9	49	**18.3**

Table 8.39: Fuzzy crisp slowdown of the highest ranked algorithms overall .NET

Connective	I	II	III	IV	V	VI	Average
Conjunction	46.1	68.6	47.2	44.8	31.2	30.7	**44.8**
Negation	143.5	138.7	105	107.5	139.5	31.4	**111**
Disjunction	47.4	68.5	48.3	46.3	31.4	31.8	**45.6**
Sum	5.9	7.5	6.2	6.2	6.2	10.4	**7**

Table 8.40: Fuzzy crisp slowdown of the highest ranked algorithms overall Mono

Other results

- Problematic float disjunction on all computers
- Very fast Mono on Linux
- Possible improvement on Windows 7
- Waiting for SSE 4

Issues

- The slowdown examined is only slowdown of the bit string computations
- Agenda of the data mining software (creation and caching of bit strings, computation of quantifiers) need to be considered
- A set of tests in the Ferda software should be carried out to get more realistic results
- Expecting less slowdown

- Questions?
- Thank you for your attention