Fuzzy Logic and SAS® Software - Do They Work Together?

Markku Suni, Sampo Insurance Company, Turku, Finland

Abstract

Reports typically contain observations according to some criteria. An insurance company might want to list customers paying over x dollars or at most y dollars for an insurance policy. It might be that marketing department needs a report showing the customers with expensive car and boat - which can be deduced by the insurance amount - and insufficient life coverage. The problem is that no matter how we set the amounts x and y we may always miss some interesting (x+1) or (y-1) dollar cases. This is unfortunate but cannot be helped as programming logic is rather rigid. We are not hopeless, however: There is fuzzy logic that offers a lot of flexibility and there is data step that offers us all the exiting possibilities of fuzzy logic. We can combine SQL with fuzzy data step to see that marketing department receives the report, the whole report, and nothing but the report it needs. In this paper we review fuzzy logic with SAS programming and fulfill the wishes of the marketing manager in a way he or she did not think possible.

What is Fuzzy Logic Anyway?

The decade of 60's was in many ways a happy one. Pollution and overproduction were not yet invented, people were happy, happier, hippies. Music was ever so good with Elvis, the Beatles and so on. And there was a mathematician Lotfi Zadeh thinking about ways of programming the electronic equipment - radars and weapons - of a warship in such a way that the equipment could safely identify the oncoming airplane even though the airplane might use some electronic countermeasures to distract identification [1]. In such a case the information received is not precise but rather fuzzy. This called for logic with not only TRUE or FALSE but something that was almost TRUE but somewhat FALSE. Or the other way around. Thus Zadeh developed fuzzy logic to deal with imprecise information as an extension to traditional logic. As a matter of fact we use imprecise information in everyday life without much thought about it. We speak about new cars, young women, tall men and so on. It is not unusual for a craftsman to guide someone with very imprecise wording: *put the pie in hot oven and roast until done and golden brown.*

Fuzzy logic provides for ways to model human reasoning with a computer program. In fuzzy logic, exact reasoning is viewed as a limiting case of approximate reasoning, everything is a matter of degree (degree of truth for “red car”, “tall man”), and the reasoning is done by calculating those truth values. As it follows human reasoning, fuzzy logic is suitable for uncertain or approximate reasoning, especially for the systems with a mathematical model that is difficult to derive, and it allows decision making with estimated values under incomplete or uncertain information.

Fuzzy logic has a strong mathematical theoretical background, but this paper will just scratch the surface. To begin a logical deduction in a program we think about a variable that has a value. In conventional logic and programming this value is precise: *Fahrenheit 451*, *Pennsylvania 65000*, *21 miles from Tulsa* and so on. If we do not know the exact value, we can not compute. In fuzzy logic we think about fuzzy sets. A variable has a value that belongs to the fuzzy set (say, "old men") with a degree of membership. Degree 0 implicates no membership, degree 1 implicates full membership (sure cases). Everything in between implicates uncertainty. This fuzzy set can be formulated in many ways, depending on the problem at hand. An example might be:

$$\mathbf{OLD(v)} = \begin{cases} \mathbf{0,} & \mathbf{if\ v < 20} \\ \mathbf{(v - 20)/40,} & \mathbf{if\ 20 <= v <= 60} \\ \mathbf{1,} & \mathbf{if\ v > 60} \end{cases}$$

In this example we say that anyone under 20 can not be regarded old. On the other hand, anyone over 60 is surely old. For those in between we can calculate the degree of "oldness" using the formula above. For instance a person of 45 has degree of membership of 0.625. In everyday life he or she could be considered old by those under 20 but young by those over 60. This formula is monotonically increasing for values between 20 and 60 and this sounds reasonable. Depending on the decision maker this formula can be more or less complicated. The main thing is that it gives us a unique truth value that is between zero and one and indicates the degree of truth. As mentioned, it is the decision maker who defines the fuzzy sets. In this way, we can say that in fuzzy logic the boss is always right.

In traditional reasoning we have a number of rules. For instance:

IF the customer's age is over 20 THEN customer can rent a car.

Or:

IF the customer's age is over 20 and customer's driver's license is more that one year old, THEN customer can rent a car.
In conventional expert system there is usually just one rule that applies (or "fires"). In fuzzy rule base all rules fire to some degree (though possibly the degree may be zero). Thus we have to calculate the truth value as result for each rule, then combine the outcomes of each individual rule to form overall system output. To form the outcome of a rule we need a mechanism for implication: IF a THEN b. There are several known and used implication methods, for instance:

Larsen	$x \rightarrow y = xy$
Lukasiewicz	$x \rightarrow y = \min(1, 1 - x + y)$
Mamdani	$x \rightarrow y = \min(x, y)$

We read these as: the truth value of statement "x implies y" (or "IF x THEN y") equals the minimum of truth values of x and y (Mamdani case). "Truth value" and "degree of membership" can be used meaning the same. Mamdani implication is very often used because of its simplicity. Here we note one big difference between fuzzy logic and conventional logic. In conventional rule IF the premise (the IF part) is true THEN we do exactly as the rule says (the THEN part). In fuzzy system we calculate the degree of applicability of the rule as a whole (both parts). This is because there is some degree of membership in the IF part and there is some degree of membership in the THEN part of a rule. This applies to each and every rule. After calculating the truth values of the individual rules we combine these somehow to achieve the final outcome. We could take the minimum of the individual truth values as this is equal to the degree of satisfying all rules. We may want to have the result of satisfying just one rule, in which case we take the maximum of the individual truth values. Or we may feel that it is more reasonable to have some sort of weighted average. For instance, we may have noticed that our four rules give us truth values of (0.62, 0.20, 0.15, and 0.03). We may want that the first rule very much - but not totally - decides the outcome. The other rules have some effect.

Fuzzy programming

We have learned to write programs based on conventional logic. We can learn to write programs based on fuzzy logic. It gives more work because we need to do calculations to decide the degree of membership, and somebody has to define the membership functions for each possible linguistic value (old, young, middle-aged and so on) and to decide the "correct" form. We also have to do some calculations for the THEN part of each action as well; this can be difficult to understand and do in the beginning. On the other hand, we receive advantages for applying fuzzy logic: we can formulate the problem more closely to spoken language and follow the human reasoning more closely, our program works smoother and is not so prone to certain exact limits. Our parameters may not be exactly "right" anyway; fuzzy logic is more forgiving.

Fuzzy Logic in SAS

SAS is just another programming language, just as APL, BASIC, COBOL, DIBASIC, EIFFEL, FORTRAN, or some other. It has no particular features to support fuzzy logic but then again, very few - if any - programming languages have such features. Dealing with fuzzy logic calls for some extra programming and this calls for data step. There is no particular difficulty in writing fuzzy data step. A fuzzy proc step is almost impossible with present SAS. Almost but not quite as PROC REPORT allows data step statements within COMPUTE blocks. Those statements can be just as fuzzy as need be. Other than that, however, it is hard to imagine adding fuzzy logic to a SAS procedure by means of current PROC statements.

Having the theory, the problem is to implement it practically, which calls for some thinking. Companies often store their information in some relational data base, say DB2® of IBM® (trademarks). Our task is to write a report selecting from DB2 tables those observations that fulfill fuzzy constraints. We can read DB2 tables with SQL using PROC SQL, or we can use data step just as we were reading SAS tables. If we use SQL, we cannot readily use fuzzy logic. Thus it might be advantageous to use data step. If we do not want to create new tables every so often, we could use data step view. On the other hand, if we have to read many tables combining rows, it may well be that SQL is a better alternative. In that case we have to select all those cases that might be possible (having truth value more than zero) and in a later fuzzy data step drop out those that are not interesting. We can somewhat help our programming with a set of macros to which we describe the fuzzy sets and which generate the SQL statements.

What did the marketing department want and get?

Take an example. We work in an insurance company the marketing manager of which wants to increase the sales of life insurance. The manager, J. Random Boss, formulates a simple task: list the customers who have an expensive car and boat, and at the same time have insufficient life coverage. That is enough for her. We have to discuss with her about what is "expensive" and what is "insufficient"? This seems to be calling for fuzzy approach. She comes to the conclusion that the expensiveness of a car or boat can be seen on the price or based on the annual payment. It could be more complicated, but we have to start somewhere. As for life insurance it is clear that "insufficient" depends on the overall situation of the client. One rule of thumb says that life insurance should cover the person's debts, about which we have no information. Another rule of thumb says that life insurance amount should reach or exceed one year's worth of gross income. About that we do not store any information, either. Could we estimate a person's income by the value of car? J. Random Boss notes that it is not a trivial problem. Finally she decides that for the time being

- a car is surely expensive if it costs more than 30,000 dollars,
- a car is not expensive if it costs less that 15,000 dollars,
- a boat is expensive if it costs more than 40,000 dollars,
- a boat is not expensive if it costs less than 10,000 dollars,
- we take it that the customer's annual income equals twice the price of the car,
- life insurance with cover less than annual income is insufficient,
- life insurance with cover over two year's annual income is sufficient
- anything in between those limits is partly possible.

Having this we have to decide how to combine these premises. The general practice is to replace AND with a minimum (fulfills all rules) and OR with a maximum (fulfills at least one rule). In other words, take the smallest or largest truth value and take that as the final truth value. We do so to start with. But now we have gotten J Random Boss really excited. All these rules may not be of equal importance:

- a car is a necessity, a boat is a hobby; so let us give the car rule a weight of 3 and the boat rule a relative weight of 1
- insufficient life cover is the main thing; let us give it a weight of 6.

It might be a good idea to begin by trying either way, compare the results, and then decide. After this thinking we can formulate the fuzzy sets needed:

```
expensive car(c) =
        1,                  if c >= 30.000
        ( c - 15.000 ) / 15.000,
                            if  15.000 <= c <= 30.000
        0,                  if c < 15.000

expensive boat(b) =
        1,                  if b >= 40.000
        ( b - 10.000 ) / 30.000,
                            if  10.000 <= b <= 40.000
        0,                  if b < 10.000

sufficient life cover(l) =
        1,                  if l >= 2 * c
        ( l - c ) / c,      if c <= l <= 2 * c
        0,                  if l < c
```

Thus we have the reasoning rules:

```
        IF      the car is expensive
        AND     the boat is expensive
        AND     the life cover is insufficient
        THEN the customer is interesting.
```

For the report we could select the customers with large enough truth value of interest. After studying the resulting SAS program and the resulting report, J. Random Boss cried out loud: *"For crying out loud! Can a SAS program do something like this? And so easily! I thought this could only be done using advanced programming languages."*

Fuzzy SAS Program

We can write the program having a data step to read DB2-tables via views or use PROC SQL to select from DB2-tables those observations with truth value more that zero and a let a data step carry on from that point. In our data base we have five tables: car table, boat table, and life table all having policy number and information about the policy, a joint table having policy number and customer number, and finally customer table having customer number and information about the customer.

```
PROC SQL;
      create table crufty as
      select car.polnr,
             car.cprice,
             boat.bprice,
             life.lcover
      from   car, boat, life
      where  car.polnr = boat.polnr
      and    car.polnr = life.polnr
      and    cprice >= 15000
      and    bprice >= 10000
      ;
```

```
DATA cuspy
 (drop=weighc weighb weighl);
      retain weighc    3
             weighb    1
             weighl    6 ;
      set crufty;
      /* have a case, check */
      /* calculate each rule*/
      link rule1;
      link rule2;
      link rule3;
      /* combine the individual */
      /* rule outcomes          */
      link total;

      if l > 0; /* life cover ?? */
*      if alfa1 > 0.4; /* select */
*      if alfa2 > 0.4; /* some   */
      output;
      return;

rule1:
      if cprice >= 30000 then c = 1;
      else c = (cprice-15000)/15000;
      if c < 0 then c = 0;
      return;
rule2:
      if bprice >= 40000 then b = 1;
      else b = (bprice-10000 )/30000;
      if b < 0 then b = 0;
      return;
rule3:
      if lcover >= 2*cprice then l=0;
      else if lcover <= 0.5 * cprice
            then l = 1;
      else l=(cprice-lcover)/cprice;
      if l < 0 then l = 0;
      return;

total:
  /* Here we have two ways to combine
      rules */
  /* Way 1: AND - minimum */
      alfa1 = min( c, b, l );

  /* Way 2: weighed average  */
     alfa2 =
    (c*weighc + b*weighb + l*weighl)/
       ( weighc + weighb + weighl );
    return;

proc sort data = cuspy;
        by alfa1;

proc print data = cuspy noobs;
     title "The customers selected";
run;
```

```
                      The customers selected

POLNR      CPRICE      BPRICE      LCOVER      C      B      L   ALFA1   ALFA2

 100      $21,100     $12,510     $20,742   0.41   0.08   0.02   0.02    0.14
 171      $18,093     $16,087     $17,739   0.21   0.20   0.02   0.02    0.09
 149      $15,609     $11,139      $7,998   0.04   0.04   0.49   0.04    0.31
 105      $27,721     $11,304     $16,513   0.85   0.04   0.40   0.04    0.50
 133      $24,008     $25,979     $22,101   0.60   0.53   0.08   0.08    0.28
 101      $24,870     $12,744     $20,667   0.66   0.09   0.17   0.09    0.31
 121      $26,029     $12,810     $10,587   0.74   0.09   1.00   0.09    0.83
 124      $19,565     $12,905      $9,908   0.30   0.10   0.49   0.10    0.40
 169      $25,633     $14,733     $22,485   0.71   0.16   0.12   0.12    0.30
 109      $19,297     $14,031     $14,380   0.29   0.13   0.25   0.13    0.25
 134      $33,779     $23,248     $27,964   1.00   0.44   0.17   0.17    0.45
 132      $38,234     $16,291     $11,693   1.00   0.21   1.00   0.21    0.92
 155      $25,349     $17,041      $1,663   0.69   0.23   1.00   0.23    0.83
 177      $22,365     $18,305     $13,996   0.49   0.28   0.37   0.28    0.40
 112      $19,810     $19,326      $8,132   0.32   0.31   1.00   0.31    0.73
 150      $26,575     $19,558     $13,352   0.77   0.32   0.50   0.32    0.56
 137      $43,864     $24,160      $9,730   1.00   0.47   1.00   0.47    0.95
```

Fuzzy Conclusions

We note that we can formulate our programs more closely according to the decision maker's wishes. On the other hand, we now have to bother him or her to think about the matter more carefully, which may be something he or she does not want to do. But as we do not need exact and final limits of x dollars or so, we have a lot of flexibility. All in all we can say that we shall reach better and in a way more exact results.

In this example case the price or the boat seemed to be the problem point. In terms of our rules, the boats here seemed to be of very reasonable price, which is why our "expensive boat" had rather modest truth values. Maybe J. Random Boss should think again about that question: what can be regarded as "expensive boat" among the customers of this company. Or should she be interested in boats at all? Looking at our report we note that the weighted average (ALFA2) seems to give a result that feels intuitively right. On the list, observations with ALFA2 > 0.5 are changed to boldface.

In this example case the rules were very simple. The code for fuzzy logic did not add a lot to the program, which is easy to write and understand once the basic principles are grasped. In real life we may have much more complicated rules and a lot more of them. Yet we notice that the added programming effort for fuzzy reasoning is very reasonable.

Fuzzy logic is a way of reasoning. Having solid mathematical background it can be applied to all sorts of logical problems. It has been shown that any logical problem can be fuzzified. A fuzzy reasoning mechanism can be implemented in a SAS program with modest amount of effort. The eventual problem lies in formulating the fuzzy sets and rules of the problem, not so much in programming per se. SAS as such is no better or no worse for fuzzy reasoning than some other programming language. Definitely better than COBOL, anyway.

SAS programs are currently widely used in different reporting-related tasks, where we select observations from data files, combine and manipulate those and finally produce a report. Selecting the observations is a field where fuzzy logic can offer number of advantages compared to conventional logic. Unfortunately we cannot add fuzzy logic to SAS procedures, PROC SQL especially included. Thus in order to apply fuzzy logic we have to add data step programming to the task - even in the case where PROC SQL and PROC TABULATE would otherwise suffice, which means some extra labor and costs. In many cases the reward can exceed the costs included. The reward is

- greater conformity with the original verbal formulations of the problem
- more flexibility to the programs
- we can follow the human cognitive processes more closely
- our program works smoother and is not so prone to certain exact limits
- we do not need to know exatc limits
- we allow certain amount of softness in the reasoning
- our parameters may not be exactly right anyway; fuzzy logic is more forgiving.
- we can work with imprecise, approximated, or even missing, information.

All of this means that we shall probably hear more and more SAS programmers crying not "XUZZY" but "FUZZY", and see more and more SAS programs applying fuzzy logic.

References

[1] Robert Fullér: Neural Fuzzy Systems, Lecture Notes, Institute for Advanced Management Systems Research, Ser A:443, Åbo Akademi, 1995.

There are zillion books about fuzzy logic. To start with basic information one could think of the book *Fuzzy Thinking* by Bart Kosko.

For further information, comment and discussion, please contact:

Markku Suni
Consultant
The Sampo Group

P.O.B. 216
FI-20025 SAMPO
Finland

tel.: +358-10-514-2095
fax.: +358-10-514-2126

e-mail: markku.suni @sampo.fi