Paper 23-26

How to Build Human-like, Fuzzy Application with SAS/AF

Markku Suni, Sampo Plc, Turku, Finland

ABSTRACT

In this paper there are three points to direct our attention to: people, computers, and people and computers together. People do not usually think in exact terms. Instead of having a fixed amount in mind, they tend to ponder, whether a car, an insurance policy, a house, or something, feels expensive. The customer might be willing to go somewhat higher in terms of price if the offer is otherwise interesting. On the other hand, a not so interesting offer may feel expensive even at a lot lower rate. This is well known by all salespersons. On the other hand, computers are most exact in their thinking - or should we say, reasoning. When building applications of the people, by the people, and for the people, we should keep in mind those differences in thinking. As the raw processing power has become relatively cheap, we can afford to concentrate on making applications to imitate the human way of thinking.

Enter fuzzy logic, which is a way to handle fuzzy terms, like "expensive", "cheap", and so on. It is especially good for cases with uncertain, approximate reasoning, or incomplete information. Based on it we can create applications which more closely follow spoken language, are much more flexible, and not dependent on correct limits. The problem is, SAS is not a fuzzy language as such. This paper discusses the possibilities and methods to add fuzzy logic in a SAS application.

MOTIVATION OR "WHY BOTHER"

Our traditional way of building applications has been first to study the users' requirements, then to define very carefully and thoroughly the system, and finally to build it. We have heard of "waterfall" model and many others. The basic idea has been that we should a priori know who will use the system, in which way, and what sort of actions will be involved when using the application. Then we can precisely design the screens and functions of the application to conform the requirements. Of course, there has been a case or two where the application builders have had to make amendments and change the application a bit while still building it, but basically it has been this way: define first, then build.

The times they are a-changing, however. There are two very modern inventions that can have large impact in new applications. One of them is **Data Warehouse** and the other is **Internet**. Data Warehouse is an organized collection of data, organized and optimized for reporting - all sorts of reporting. Basically a Data Warehouse can be used by everyone in the organization to find out how well we have been doing in a certain area. In order to be useful, the Data Warehouse needs a user interface that makes it possible to make queries in the most flexible way. The problem is that there is no way of knowing *a priori*, what sort of queries there will be. In general we know that, but as time goes by, more and more users will begin using the Data Warehouse and posing even more and more weird queries. In case of Internet (or Intranet, which technically is exactly the same) we can place some data available to everyone via the net – even the data warehouse. The user interface is the browser and we can include all sorts of applications behind the same user interface. An insurance company might enhance its homepage with advisory application that gives advice to users as on how to select an appropriate insurance policy. A financial services provider might add an advisory application about how best to invest or save money. In these cases, instead of having certain fixed numeric limits, there is a need to use describing words like "expensive", "cheap", and so on. This sounds like fuzzy logic.

WHAT IS FUZZY LOGIC?

Back in sixties a mathematician Lotfi Zadeh was working for US Navy. He studied ways to program radar and other equipment of a warship to safely identify an oncoming airplane even though it tries to distract identification. The information received is inexact, missing, obscure, even wrong; in short: fuzzy. He developed fuzzy logic to deal with such information. For traditional logic there are two truth values: **TRUE** and **FALSE**. In fuzzy logic we speak about degrees of truth that range from **0** (surely false) to **1** (surely true) with all the values in between. For instance a car could be expensive with truth value **0.7**. For combining truth values we can use different methods, very common being the minimum ("all rules satisfied") or weighted average with the weights being selected in this or that way. To calculate the truth values we need a function that is defined by the decision-maker, the boss.

"BUILD THE SYSTEM", SAID THE BOSS.

Let us say we are working in an insurance company that uses SAS a lot. We are about to build an advisory system to recommend an appropriate insurance policy for a customer, perhaps someone who has visited our homepage. The user looks at our web pages, fills up a form on a page, and receives a recommendation for selecting a policy. Trans-

parently to the user, the browser collects the information on the page (the form) and starts an application giving it the information from the user. The application runs, puts the results on a page, which the browser finally shows to the user. The **SAS/AF**-based system together with the **www** pages will be very nice with all the bells and whistles in the user interface, but how about the logic. The system needs to get answers to questions like *"do you travel a lot?*", or *"have you got expensive photographic gear?",* where the exact amount may not be important but whether it is a lot or a little, like *"large mortgage?".*

OUR ALTERNATIVES

The first alternative is to take the Q&D approach and to leave the fuzziness totally to the user using questions like *"Do you travel a lot?",* or *"Do you prefer expensive, moderate, or cheap xxxx?"* and let the user answer **Yes, No**, or **1, 2, 3,** or **E, M, C** depending on the choice list. This way the application internally has simple exact values:

```
EARNLOT = 'Y';
TASTE   = 'E';
   ............
IF EARNLOT = 'Y' AND
      TASTE = 'E' AND
        .... THEN ...
```

Now the SCL code behind the screens can be filled with conventional IF-statements. Eventually the system is conventional, although it may seem to be fuzzy. The problem with this easy and logical approach is that it is very prone to user interpretation. What is expensive to one may be moderate to someone else. This method can only be recommended in cases, where there really is no danger of different interpretations.

The second possible alternative - still in the Q&D category - is to do as before but give the user some idea about what we mean by *"expensive", "moderate",* or *"cheap"* and so on adding explanative text on the input screens:

```
DO YOU LIKE
   EXPENSIVE    (ABOVE 30,000),
   MODERATE     (10,000 -  30,000),
   OR
   CHEAP        (BELOW 10,000)  CAR?
 ANSWER E, M OR C:
```

This approach helps in standardizing the answers, yet the internal SCL logic can be conventional. This is often seen in practice. As a matter of fact, we see it often in cases where nobody has been thinking about fuzzy logic.

If we want the system to work the way a human works, why not add real fuzzy logic. It is not overly difficult. We begin with asking the boss to define the fuzzy sets, like *"a car is surely expensive if it costs more than 30,000; it is surely cheap if it costs less than 10,000".* For the prices in between we calculate truth value, like

```
expensive car  =   ( price - 10000 ) / 20000
```

Should the need arise we can easily change the limits 10,000, or 30,000. They are not critical for program code and results. We could also add another rule:

"Truth value for car being moderately priced is
```
     ABS(1 - ( price - 15000 ) / 15000 ));"
```

Now the car can be somewhat expensive and somewhat moderately priced at the same time, we just combine these values in one way or another to get the final result.

SCL as a language contains a hungus amount of functions, but not a single function to deal with fuzzy sets. There is hope, however. As SCL accepts almost all the statements in SAS data step, there is no problem in implementing these fuzzy types of calculations. The calculations described above can be formulated easily in terms of data step statements. Exact values, like the price of the car, or the number of dependents can be fuzzified (converted to fuzzy numbers), then used as fuzzy information. This way the application has fuzzy information about whether something is *"expensive"*, something else is *"moderate"* and so on.

For instance, Hugo, our boss, might want to have the truth value for "travel a lot" to be as

```
"travel a lot" =  1, if more than 24 times a year
                 ( 1 -  ( 1 /  ( 2 * travels ))),
                    if 24 times a year, or less
                  0, if no travels
```

For the final outcome we need to combine the outcomes of the individual rules. A useful and simple method of combining the outcomes is to use the weighted average of individual rule outcomes, which gives us the added bonus of being able to weigh the factors according to the situation at hand and the taste of the boss.

Now, let's say we have whether a customer

travels a lot	(truth value lot)
regularly	(truth value reg)
carries expensive luggage	(truth value exp)
gets seasick easily	(truth value sic)

We can now have something of the form:

```
Answer =
(0.25*lot + 0.40*reg + 0.30*exp + 0.05* sic)
  / ( 0.25 + 0.40 + 0.30 + 0.05 );
```

In this formula, the regularity is the key with largest weight with expensive luggage being a good second. And finally we have a truth value, which we defuzzify and get a final result based on which we can the give the user our recommendation, something like:

```
"As you seem to
    travel a lot,
    at irregular intervals,
    carry expensive luggage,
    and get seasick easily,

 it seems that you should

 have a Travel-Sampo policy     and

 include this in your Sampo Agreement".
```

Of these the first five lines repeat the user's input. The sixth and seventh form the result of the reasoning. Finally the eight line is something always added as a recommendation.

CONCLUSION

Programming languages SAS and SCL are not fuzzy languages per se, yet I have shown that fuzzy logic can be added also to SCL code making the applications more flexible, more human-like and less prone to exact limits without adding unduly amounts of complexity. This way we can build applications that more closely resemble the human way of thinking, work more smoothly, and are not so prone to exact limits. As a matter of fact, we may not know the correct limit values, yet the application can be most useful. Fuzzy logic also gives us the added bonus of greater conformity to the original verbal formulation of the problem, and certain amount of softness in the reasoning.

There is no real reason not to apply fuzzy logic in SAS/AF applications every time the task calls for it.

SAMPO PLC

Sampo Insurance Company Plc is part of Sampo, the leading provider of financial services in Finland. I work there as the local SAS Consultant. We have had SAS since 1989.

REFERENCES - SORT OF

There are zillions of books about fuzzy logic. For basic information the book *Fuzzy Thinking* by Bart Kosko is excellent to start with.

SAS, SCL, and SAS/AF are trademarks of SAS Institute.

CONTACT INFORMATION

For further information, comment and discussion, please contact:

Markku Suni, MSc, VTS
Consultant
Sampo
P.O.B. 2063
FIN-20075 SAMPO
Finland

Tel: +358-10-514-2095
Fax: +358-10-514-2124

e-mail: markku.suni @sampo.fi