Implementation of a Fuzzy Rule-Based Classifier using Genetic Algorithms

Rashed,David,Joseph

December 8, 2017

1 Introduction

Usually, fuzzy systems are designed by expert. We rely on expert to find the membership values and the rules to design the Knowledge Base. However, there might be some cases where the expert is not trustworthy, or there is not expert available or else that a human may not be able to be an expert in our domain. Thus, this essay is trying to explore the use of a computer heuristic to design one part of a fuzzy rule-based classifier, namely Genetic Algorithm (GA).

GA is proven to be a good on optimization problems. They can search through complex spaces efficiently. Moreover, the fitness function let us implement some biases or preferences.

The goal of this project is to implement a rule based classification system using both genetic algorithm and fuzzy logic. We expect our algorithm to find rule set that satisfies two objectives : accuracy & simplicity. We want a rule set that classifies correctly our dataset, but also have simple and human-interpretable rules.

This work is inspired by the classifier presented by Xiong, N., and Lothar Litz. in *Generating Linguistic Fuzzy Rules for Pattern Classification with Genetic Algorithms.* (1999) based on the Iris data. This dataset is a well known classification data with four attributes and 3 classes. The attributes are Sepal Length, Sepal Width, Petal Length and Petal Width,and classes are Iris Setosa, Iris Verticolor and Iris Virginica.

(Do we have to do section X is for Y,....)

2 Fuzzy Logic

2.1 Bla

2.2 Fuzzification

2.3 Inference system

2.4 Defuzzification

A fuzzy rulebased system has two components: the Knowledge Base and the Inference System.

Knowledge Base : storing the available knowledge in the form of fuzzy rules Inference system : applying a fuzzy reasoning methodon the inputs and the KB rules to give a system output.

Knowledge base is usually obtained from expert knowledge, but also by machine learning since the field emerged in the 80s.

¿Data base: Granularity (nb of ling terms/labels)/ var Membership functions associated ot the labels ¿Rule Base: fuzzy rule composition

KB is composed of RB and DB RB: Rule set, DB: Membership funct

input¿fuzzification interface¿inference mechanism (applying RB & DB)¿Defuzzification Interface¿ output

Inference System is set up by choosing the fuzzy operator for each component (conjunction, implication, defuzzifier, etc...)

2.5 Classification model for fuzzy rules

For each of these attributes $x_i (i = 1 \cdots n)$ there is three fuzzy sets that are denoted A(i,S), A(i,M), A(i,L), respectively for Small, Medium and Large. Let's define **p** as the mapping function from $\{1 \cdots s(s \leq 4)\}$ to $\{1 \cdots 4\}$, with $\forall x \neq y, p(x) \neq p(y)$. We use that function to express that not all parameters are needed in a rule. Thus we can generalise rules as follow :

$$if \left([x_{p(1)} = \bigcup_{j \in D(1)} A(p(1), j)] \bigcap \cdots \bigcap [x_{p(s)} = \bigcup_{j \in D(s)} A(p(s), j)] \right) then \quad \mathbf{B}$$

With $D(i) \subseteq \{Small, Medium, Large\}$
$\bigcup$ corresponds to **or** and $\bigcap$ to **and**.
So for exemple, if:

$$D(1) = \{Small\}, D(2) = \{Medium, Big\}, D(3) = \{\}, D(4) = \{Small, Medium, Big\}$$

Then our rule would read as:

If Sepal Length is Small AND Sepal Width is Medium OR Big AND Petal Width is Small OR Medium OR Big, then C_i.

There is 12 different membership functions that can be present or not in a rule. If all the functions are absent from the rule, we have an empty fuzzy set for the condition part, which is an invalid rule.
Thus there is $2^{12} - 1 = 4095$ possible rules. The number of rules is exponentially increasing with the number of parameters and linguistic variables. We could have think to encode the resulting class in our rule, but it would have increase the search space.

Xiong & Litz proposed a way to compute the classes using the antecedant and the data set:

A rule can be summarised as *If A then B, or* $A \Rightarrow B$.
$B \in \{Class_1, Class_2, Class_3, Class_4\}$

$$A = [x_{p(1)} = \bigcup_{j \in D(1)} A(p(1), j)] \bigcap \cdots \bigcap [x_{p(s)} = \bigcup_{j \in D(s)} A(p(s), j)]$$

$\mu_A(u_i)$ is the membership value of an item $u_i (i = 1 \cdots m)$ from the training set $U_t = \{u_1, \cdots, u_m\}$ to A. The consequent B is a crisp subset which defines as follow:

$$\mu_B(u_i) = \begin{cases} 1 & \text{if } class(u_i) = B \\ 0 & \text{otherwise} \end{cases}$$

(For the class prediction later, B will be this time a fuzzy subset)

$A \Rightarrow B$ is equivalent to $A \subseteq B$, so the subsethood of A in B will be used to compute the truth value of a rule for each class:

$$truth(A \Rightarrow B) = \frac{M(A \cap B)}{M(A)} = \frac{\sum_{u_i \in U_T} (\mu_A(u_i) \wedge \mu_B(u_i))}{\sum_{u_i \in U_T} \mu_A(u_i)} = \frac{\sum_{class(u_i)=B} \mu_A(u_i)}{\sum_{u_i \in U_T} \mu_A(u_i)}$$

3 Rule Based Genetic Algorithm

3.1 Presentation

There is two type of GA rule based classifier : Pittsburgh approach and the Michigan approach one. Pittsburgh : Each chromosome is a rule set.

Michigan : each chromosome is a rule, the system evolve as a whole. We decided to use the Pittsburgh approach as it was more intuitive for us to build.

Genetic Algorithm can be used to various part of a fuzzy system:

Tuning the membership function Rule learning Rule selection Learning of Knowledge Base Inference parameters Etc... (see `http://sci2s.ugr.es/sites/default/files/files/TematicWebSites/GeneticFuzzySystems/FUZZ-IEEE09_Tutorial_EMOFRBSs-Ishibuchi-Alcala.pdf`)

3.2 Modeling the algorithm

3.2.1 Individual

We define an individual as a list of rules. Rules are represented with a list of 12 bits, 3 bits for each parameters. These 3 bits express the presence or not of each linguistic variable for D(i), 1 if present, 0 otherwise. For exemple the rule presented in section 2 would be represented as [1,0,0,0,1,1,0,0,0,1,1,1]. We decided to set 10 rules for each individual.

3.2.2 Selection and Crossover of individuals

In genetic algorithms, it's important to avoid a **premature convergence**, that is when a population converge toward a local extremum rather than the global optimum.
The Tournament Selection is a way to chose a user-selected number of random individuals in a population, and then take the fittest.
It is possible to give a chance to weaker individuals by reducing the number of participing candidates, or in the opposite to prevents weaker ones to breed by increasing the number. Two selections gives two individuals that can mate with a crossover. The idea is to mix the chromosomes of the individuals to produce a child inheriting the genes of his parents.
Many techniques exist to cross two chromosomes. We chose to pick the uniform crossover.
Given two rules, for each bit there is a 0.5 probability to inherit the gene of one of the parent.
We repeat the process for each pair of rules.

3.2.3 Mutation

Mutation is a key point in genetic algorithm to explore the problem space by maintaining a problem diversity. Similar to biological mutation, this operator modifies a chromosome in a unpredictable way and can give a better solution.

In our algorithm, there is a mutation parameter that will give the chance of each gene to flip the bit. Each bit of each rule of an individual will have a probability p_{mut} to be flipped.
If p_{mut} is too high the information given by the chromosome might be lost because it changed too much, and thus the algorithm may never converge. Else if p_{mut} is too low, the exploration space may be restricted and thus the algorithm may converge to a local optimum.

3.3 The fitness function

According to the law of the fittest, the best individuals have more chance breeding and transmiting their genes. But what does it mean for an individual to be fit?
For this work, we define the fitness function as the accuracy of prediction of an individual given a test data.

4 Implementation

Optimization : cache individual rule fitnesses
Set of rules : may be rules that are incorrect

5 Results and Conclusion

6 Compute the fitness function

The calculation of the fitness function is the core of our implementation.
We will be trying to explain in depth the structure of that function.

6.1 Vector of best class confidence

Given a rule, we measure the truth degree of that rule using the formula previously described on our training data for each class. The function *getConf* gives the vector :

$$\begin{bmatrix} C_1 : \mu_b^1(r) \\ C_2 : \mu_b^2(r) \\ C_3 : \mu_b^3(r) \end{bmatrix}$$

with C_i the class and $\mu_b^i(r)$ the truth degree of that rule r on class i.

An Individual is composed by a set of rules, so the function *getConfVect* gives the vector :

$$\vec{Conf} = \begin{bmatrix} C_i^1 : \mu_b^i(r_1) \\ C_i^2 : \mu_b^i(r_2) \\ \vdots \\ C_i^m : \mu_b^i(r_m) \end{bmatrix}$$

With m the number of rules. For each rule, we take the maximum μ_b from *getConf*. This vector represents for each rule fo an individual, what class it fits the better, and with which confidence.

6.2 Vector of μ_A, given a data instance and rules

The next step is to do fuzzy composition on each of the rule, given an instance of data, here $[x_1, x_2, x_3, x_4]$.

The function *getMuA* takes two parameters: a data instance and a rule.

Remembering this formula:

$$A = [x_{p(1)} = \bigcup_{j \in D(1)} A(p(1), j)] \bigcap \cdots \bigcap [x_{p(s)} = \bigcup_{j \in D(s)} A(p(s), j)]$$

We chose to measure μ_a like this:

$$\mu_A = min \begin{pmatrix} max(A(p(1), j), \forall j \in D(1)) \\ \vdots \\ max(A(p(s), j), \forall j \in D(s)) \end{pmatrix}$$

Example:

Let take our example rule:

If Sepal Length is Small AND Sepal Width is Medium OR Big AND Petal Width is Small OR Medium OR Big

And a data instance (normalized): $x_1 = 0.97, x_2 = 0.86, x_3 = 0.97, x_4 = 0.88$.

$$\begin{aligned} \mu_A &= min \begin{pmatrix} max(\mu_S(0.97)) \\ max(\mu_M(0.86), \mu_B(0.86)) \\ max(\mu_S(0.88), \mu_M(0.88), \mu_B(0.88)) \end{pmatrix} \\ &= min \begin{pmatrix} max(0) \\ max(0.28, 0.72) \\ max(0, 0.24, 0.76) \end{pmatrix} \\ &= min(0, 0.72, 0.76) = 0 \end{aligned}$$

In the same fashion as we did for our confidence vector, we use the function *getMuVect*:

$$\vec{\mu_A^k} = \begin{pmatrix} \mu_A^k(r_1) \\ \vdots \\ \mu_A^k(r_m) \end{pmatrix}$$

For the k-th piece of data from the test data.

6.3 Assembling the pieces: predicting a class

We want now to get the predicted confidence of classes for an instance.
$\vec{Conf}$ gives us the fittest class for each rule, and $\vec{\mu_A}$ its confidence given an instance of data.
There is several way to compute the class predicted using these vectors.

Simple method:
For each class, we take the average of μ_A for every rule that have that class as the highest truth degree:

$$\text{Predicted Classes Confidence: } \begin{pmatrix} avg(\mu_A^k(r_i)), \forall & i & | & Conf(r_i) = C_1 \\ avg(\mu_A^k(r_i)), \forall & i & | & Conf(r_i) = C_2 \\ avg(\mu_A^k(r_i)), \forall & i & | & Conf(r_i) = C_3 \end{pmatrix}$$

Here, the class predicted will simply be the maximum average.

6.4 Accuracy & Fitness function

The accuracy gives us the percentage of well predicted instances of our classification on a test data. We can use only this score as a fitness function.
It seems that using the training data to define the truth degree, then applying the simple method is already a good way to classify.
When taking the average of the average accuracy of 20 populations of 20 randomly generated individuals, we have a 0.77 accuracy.
And when taking the average accuracy of the fittest individual of 20 random populations, we get a 0.96 accuracy.
Decision Tree Classifier is a similar method with crisp rules. The accuracy of that method is 0.92 on the same dataset.
However, accuracy score is not our only concern : we would like to have the fewest rules as possible, as we want to produce a clear and simple rule system if one exists. So we have to compute a fitness function combining both the accuracy score and the complexity of the ruleset.

References