Genetic Tuning of Fuzzy Systems

Chapter 5
Chapter 5

Genetic Tuning of Fuzzy Systems

In this chapter we have tried to perform Rule generation of FIS using Genetic algorithm with integer encoding of population chromosomes and rule importance and support as the fitness function.

Evolutionary algorithms offers a number of techniques that inherits the principle of evolution and offers itself as an optimization tool. They use the stochastic approaches to solve the problem and are population based. Several different approaches exist, including the Genetic algorithm (GA), Genetic Programming, evolutionary strategies, Particle Swarm Optimizer (PSO) and Ant Colony Optimization (ACO).

5.1 Introduction and overview Genetic Algorithm

Genetic algorithm founded by John Holland in the 60s, hence after is widely studied with interest, experimentation and applied in many fields in Science and Engineering worlds. It simulates the Darwin’s theory of survival of the fittest, where it uses the principles of competition, inheritance and variation within a population. Survival of the fittest implies that the ‘fitter’ individuals are more likely to survive and have a greater chance of passing their ‘good’ features to the next generation and the weakest individuals perish.

GA is an evolutionary algorithm, which optimizes a fitness function to find the solution of a problem. In a population, each individual has a set of features that determines how it is appropriate to the environment. A population of chromosomes representing the solution to the problem, that grows over time through competition and controlled variation. Each chromosome in the population has an associated fitness for selecting new ones created using genetic operators in the process of competition. The population undergoes evolution in a form of natural selection. In every successive iteration, so-called generation a new population of chromosomes is formed on the basis of evaluations using...
selection mechanism and using genetic operators. The two most frequently used Genetic operators are as follows:-

- **Crossover (or Recombination)**, where parts from two (or more) individuals are combined together to generate new individuals [15]. The main objective of crossover is to explore new areas in the search space [16].
- **Mutation**, which modifies an individual by a small random change to generate a new individual [14]. This change can be done by inverting the value of a binary digit in the case of binary representations, or by adding (or subtracting) a small number to (or from) selected values in the case of floating point representations. Generally, mutation is applied using a low probability.

for each problem to be solved a fitness function must be devised. Although there are many possible variants of the basic GA, the principle underlying mechanism consists of three operations: Construction of initial population, formation of intermediate population and recombination through crossover and mutation [10, 14]. The canonical form of Genetic Algorithm is shown in the Figure 5.1.

Most of problem solutions are formulated as optimization problems that can be well-thought-as an instance of artificial intelligence, consisting of handling information and searching through possible alternatives for optimal solution.

The process involved in optimization problems through GA is based on the natural evolution and broadly the steps involved are as follows:
1. Random generation of an initial population of chromosomes to represent the potential solutions.

2. Formulation of fitness function according to problem characteristics.

3. For each solution evaluation of the fitness function.

4. Selection of two solutions with better fitness as parents for reproducing solutions.

5. Recombining the parent chromosomes (solutions) with crossover at a random point on the string to produce two new offspring solutions.

6. Mutation of offspring solutions based on a mutation probability.

7. Evaluation of the suitability or ‘fitness’ of each solution.

8. Insertion of filter offsprings to the population and omission of weaker solutions in current population.

9. Applying check to find whether solution has been optimized if not, go to step 3.


GAs are nothing more than educated guessing. The ‘education’ comes from knowing the suitability of previous candidate solutions and the ‘guessing’ comes from combining the fitter attempts in order to evolve an improved solution.
Chapter 5 - Genetic Tuning of Fuzzy Systems

Figure 5.2: GA process steps for evolving an optimized solution

1. Randomly generate initial population of problem solution as set of chromosomes.
2. Evaluation of fitness function for each solution in the population.
3. Selection of best fittest solution as parent chromosomes.
4. Apply crossover and mutation on parent chromosomes to generate offspring's.
5. Calculate the fitness value of offspring solution.
6. Check if offspring's is better than solutions in current population.
7. Add the offspring chromosomes to the population by replacing existing weak chromosomes.
8. If fitness function satisfies?
9. Optimized Solution
10. End
5.2 GA tuning of Membership Function of Fuzzy Inference System:

The basic Fuzzy Inference algorithm is an indiscriminate implementation of fuzzy logic used in many applications. The design elements of the Basic Fuzzy Inference algorithm are the design of membership functions for the fuzzy variables and inference rules. For each fuzzy variable, the Universe of Discourse (UoD) is determined. Next the shape of the membership functions is fixed. Finally, the number of membership functions and their placement on the universe of discourse are found. For simplicity of reasoning process the membership functions selected to be triangular in shape. Typically the fuzzy rules and membership functions are generated by Experts in the area using linguistic design. However, with increasing number of variables, the number of rules increases exponentially, which makes the design challenging for the Expert. If the partitioning into fuzzy sets is too large, number of patterns likely to be incorrectly spread over the UoD. On the contrary, if it is too small, the number of rules will increase enormously. Therefore, the choice of the membership functions is a subtle task.

A standard fuzzy logic based inference system comprises the following steps-

1. Identification of input and output variables
2. Fuzzification of input and output values
3. Inputting the crisp values of input
4. Formulation of Fuzzy Inference Scheme
5. Defuzzification for computing crisp value of output.

The MATLAB code is created for steps 3 to 5. The tuned population set of membership functions for input variables are investigated by executing the Genetic Tuning Algorithm. The FIS that generates the output variable in the acceptable range (fitness value) is selected to be used in the Fuzzy Logic Model for classification. Generation of fuzzy rules from numerical data consists of two phases: to partition the pattern space into fuzzy subspaces and then to define a
fuzzy rule for each fuzzy subspace. Partitioning the pattern space and rule generation are inter-reliant.

5.3 Genetic tuning of FIS rules

Fuzzy Inference have been applied to various problems. Classification problem apart of data mining task [11] is of one which has application in fields like Medicine, Education, Business etc. Classification process can be improved by using fuzzy sets having overlapping class definitions and ‘if-then’ rules for inference that provide better reasoning in ambiguous boundary conflicts. Fuzzy rules are usually constructed by Expert based on their domain knowledge, however normally it is a tedious and cumbersome exercise. Many Researchers are still trying for automatic generation of rules using neural network, association rule mining and genetic algorithm (GA) [1][9][6][5]. The reason why GA is preferred is its global-search nature, GAs work with a population of candidate solutions (individuals). In GAs a candidate solution is evaluated as a whole by the fitness function. In present work we intended to construct a classification system with fuzzy inference. The approach is towards automatic construction of fuzzy system. The automatic construction of fuzzy inference system (FIS) for classification is developed in two phases, as follows-.

1. Self-generated membership function for FIS
2. Automatic generation of Fuzzy Rules

The automatic construction of fuzzy classification system by self-generated membership function is reported in our paper[12]. A data driven approach for automatic generation of fuzzy rules was the focus of the experiment. Initial population of rules is built in the form of matrix and input membership functions are built with membership function parameters stored in array form. These parameter arrays are generated using clustering algorithm. The initial inference rules, input-output FIS variable parameters and data are passed as arguments to program for generating the rules using genetic algorithm approach.
5.3.1 Prior work in fuzzy system tuning and data mining task using GA

The authors in [5] proposed a algorithm for association rule mining named Quant-Miner using genetic algorithm. The algorithm dynamically chooses good intervals for each rule with categorical attributes. The approach used by them is evolutionary optimizing support and confidence with fitness function based on gain measure and the algorithm looking for positive gain. SoumadipGhosh et.al [6] proposed an algorithm for positive and negative association mining using GA to find frequent item sets.

Authors in [7] proposed the algorithm to measure the quality of categorical data in association rule mining, they have transformed the categorical data to asymmetric binary form by introducing new items as per the distinct attribute pair. Apriori association algorithm is applied to find the frequent itemset with minimum support. Quality of data in the frequent item set obtained is measured in terms of confidence factor, completeness, interestingness, comprehensibility. The rules with optimum values for all measures are searched using genetic algorithm. The author in [8] tried to explore the different methods of using genetic algorithm with K- Nearest Neighbor Algorithm to improve the classification accuracy and minimize the training error. Soraj et.al in [9] proposed a genetic-fuzzy approach for the discovery of fuzzy decision rules from datasets containing categorical as well as continuous attributes. They have fuzzified the continuous attributes using the triangular fuzzy membership function and also designed a similarity check while calculating the fitness of an individual in Genetic algorithm population. The flow chart of the genetic algorithm developed to generate fuzzy rules for classification is depicted in the Figure 5.3.
Data Mining: A Soft Computing Approach

Chapter 5 - Genetic Tuning of Fuzzy Systems

Figure 5.3: Flow chart of Genetic algorithm

1. Start
2. Run = 0
3. Gen = 0
4. Create initial rule population
5. Generate FIS with rule population created
6. Termination criterion satisfied?
   - Yes: Designate the result of run
   - No: i = 0
7. Evaluate fitness of each rule chromosome
8. i = i + 1
9. i = N?
   - Yes: Insert the offspring’s by replacing the worst individuals in new population
   - No: Select two individuals as parent p1 and p2: p1-Best fitness value, p2 - any other
10. Select two individuals with worst fitness
11. Select the crossover points and perform crossover
12. Generate the offspring’s
13. Both offsprings generated are exact duplicates of any rule in existing population?
   - No: Offspring’s fitness better than worst individuals?
     - Yes: Designate the result of run
     - No: End
   - Yes: Termination criterion satisfied?
     - Yes: Designate the result of run
     - No: Insert the offspring’s by replacing the worst individuals in new population
14. End

Run = Run
5.3.2 Procedure for Fuzzy Rule Base Generation and Tuning

The proposed algorithm for generation of Fuzzy rules comprises of four procedures.

1. The genetic algorithm for rule generation: \textit{Genetic \_ Rule\_Gen}
2. The fitness calculation function comprising of two procedures: \textit{support\_Rule\_class} and \textit{rule\_ratio\_importance}
3. Evaluation of FIS with rule population generated: \textit{GARule}

The steps for generation of fuzzy rules by using genetic algorithm are as follows:

Repeatedly for ‘n’ generations following steps are carried out-

1. Built the FIS with rule population and evaluate the FIS rules against the output class to return rule firing strength for given data.
2. Calculate and return the support for each rule and the classification class.
3. Calculate and return the importance of each rule which is the fitness value of each rule.
4. Select two parent chromosomes ‘p_1’ and ‘p_2’ from the rule population where ‘p_1’ is parent chromosome with best fitness value and ‘p_2’ is any parent chromosome from rule population, such that ‘p_1’ \neq ‘p_2’. Usually according to GA approach two best fitness parent chromosomes are chosen. In this experiment we have chosen one best fitness value for parent chromosome and other parent chromosome with any fitness value as this lead to more variations in children chromosome than in usual approach.
5. Select two rule chromosomes with least fitness value.
6. Crossover points are generated using two points crossover method.
7. Two offspring chromosomes are generated at the crossover points of parent chromosomes.
8. Each of the offspring generated is checked and if present in current population either partially or fully.
9. If offspring chromosomes are not fully or partially matched with current population rule chromosome then replace the offsprings with selected least fitness value chromosomes to build new rule population.
10. If both offsprings are present fully in current population do not replace the offsprings with the least value chromosomes, simply go through next generation.

11. If any one or both of the offspring chromosomes are in partial form matching with any of the rule chromosomes in current population, then evaluate the fitness function of the offspring chromosome and compare it with matched rule chromosomes.

12. If offspring chromosome is having better fitness value than matched rule chromosome then replace the offspring chromosomes with respective matched rule chromosome to build new rule population.

13. Pass the new rule population to the next generation.

There has been ample research on termination criteria of genetic algorithm. The simplest being the iteration based on specific number of times or time based where after specific time with ever best fitness value obtain is taken as final [2][3][9]. Bhandari et al [4] proposed variance as stopping criteria for genetic algorithm, where they have proposed that if \( a_i \) be the best fitness function value obtained at the end of \( i^{th} \) iteration of an GA, then \( a_1 \leq a_2 \leq a_3 \leq \ldots \leq a_n \leq \ldots \leq F_1 \), as \( F_1 \) is the global optimal value of the fitness function.

if \( \bar{a}_n = \frac{1}{n} \sum_{i=1}^{n} a_i \) be the average of the \( a_i \)'s and \( \bar{a}_n^2 = \frac{1}{n} \sum_{i=1}^{n} a_i^2 \) be the average of the \( a_i^2 \)'s up to the \( n^{th} \) iteration, then variance of the best fitness values obtained up to the \( n^{th} \) iteration, defined by \( b_n \), is

\[
b_n = \frac{1}{n} \sum_{i=1}^{n} (a_i - \bar{a}_n)^2 = \frac{1}{n} \sum_{i=1}^{n} a_i^2 - \bar{a}_n^2 \]

\( b_{no} \) can be used as a stopping criterion for a GA. It is stopped or terminated after \( 'N' \) iterations when \( b_N < \varepsilon \), where \( (\varepsilon > 0) \) is a user defined small quantity.
5.3.3 Algorithms of GA Procedures

The proposed algorithms of GA procedures are as Genetic _ Rule_Gen, support_Rule_class, rule_ratio_importance, GARule

5.3.3.1 Algorithm of Procedure Genetic _ Rule_Gen

Algorithm of Procedure ‘Genetic _ Rule_Gen’ is as follows-

Step 1: Input the initial rule population as matrix P[] \_{nxm} (where n is the number of rules and ‘m’ is the number of rule parameter in form \([a_1,a_2,…a_n,o,w,c]\)

‘a’ is input variables of a rule with ‘i’ input variables; i =1…n,

‘o’ is the output variable,

‘w’ is the rule weight and

‘c’ is connection AND/OR.

membership function generated for fuzzy input variables as matrix \text{input}(i)[]_{1\times mf} (where ‘i’ is input number and \text{mf}=1,2,… j), input data,

‘M’ is the specified number of epochs, ‘E’ the error tolerance.

Step 2: Declare the two parents chromosome structures \(p_1,p_2\) and two offspring chromosome structures \(off_1,off_2\).

Step 3: Initialize the Run count to 1

Step 4: Repeat step 5 to step 16untilRun< M

Step 5: Initialize the generation count to 1

Step 6: Call GaRule procedure to automatically generate the FIS and evaluate the FIS rules against the output class to return rule firing strength for given data.

Step 7: Call fuzzy classification procedure specified in Appendix program fuzzyclass.mto classify the input data, calculate average error and return it.
Step 8: if average_error ≤ε

Step 8.1: Store the new population POP of Run in Run_output [] array.
Step 8.2: Increment Run count by 1.
else
Step 8.1: Calculate the fitness of each rule from rule population.
Step 8.2: Call support_Rule_class procedure to calculate and return the support for each rule and for each classification class.
Step 8.3: Call rule_ratio_importance procedure to calculate and return the importance of each rule which is the fitness value of each rule.

Step 9: Select two parent chromosomes from the population where

\[ p_1[ a_{ij1}, a_{ij2}, \ldots, a_{ijn}, o_i, w_i, c_i ] \]

is parent chromosome with best fitness value and

\[ p_2 \]

is any other parent chromosome from rule population P[], \( p_1 \neq p_2 \).

Step 10: Compute the crossover points \( crp_1, crp_2 \) as follows

\[ crp_i = \text{floor}(1 + (\text{rand}(1) \times n)) \quad i=1,2 \]

Step 11: Generate offsprings at the crossover points

\[ crp_1, crp_2, \text{off1}[X_1, X_2, \ldots, X_n, w, c], \text{off2}[Y_1, Y_2, \ldots, Y_n, w, c] \]

Step 12: Select the least fitness value chromosomes from the P[]

\[ d1= \text{rule chromosome number with ‘min’ rule importance} \]

\[ d2= \text{rule chromosome number with second last ‘min’ rule importance} \]

Step 13: Check if offspring off1 and off2 chromosome pattern generated are fully or partially in P[].

Step 13.1: if (\( \exists (\text{off1} \land \text{off2} ) \in P \)) then set flag_both = true

Step 13.2: if (\( \exists T[x_1, x_2 \ldots, x_n - 1] \in P == \text{off1}[x_1, x_2 \ldots, x_n - 1] \)) then

Step 13.2.1: Call GaRule, support_Rule_class,
Chapter 5 - Genetic Tuning of Fuzzy Systems

**rule_ratio_importance** procedures for off1

Step 13.2.2: if rule_ratio_importance(off1) >

rule_ratio_importance(T) then d1 = T;

Step 13.3: if \( \exists T[x1, x2, ..., xn - 1] \in P \implies \text{off2}[x1, x2, ..., xn - 1] \)

then

Step 13.3.1: Call GaRule, support_Rule_class,

**rule_ratio_importance** procedure for off2

Step 13.3.2: if rule_ratio_importance(off2) >

rule_ratio_importance(T) then d2 = T;

Step 14: if(flag_both == true) then

Step 14.1: no replacement of off1 and off2 with d1 and d2 in P.

else

Step 14.1: P[ ] = P[ ] - d1 and P[ ] = P[ ] - d2

Step 14.2: P[ ] = P[ ] \cup \text{off1} \cup \text{off2}

Step 15: increment generation count.

Step 16: return P // new rule population

Step 17: end

Pseudo code implementation of the algorithm described above are given in pseudo code 5.1.

**Pseudo code for procedure Genetic_Rule_Gen (POP, m, n, data, \( \epsilon \))**

**Input:**

POP: array representing population of rule chromosomes.

m: number of runs to be executed.

n: number of rules.

data: input records for classification

\( \epsilon \): % error tolerance for classification
Output: Run_output: rule populations generated

1. Start
2. \( P[i] \leftarrow \text{POP}[i] \)
3. \( P1 \leftarrow \text{declare structure to store parent1 chromosome} \)
4. \( P2 \leftarrow \text{declare structure to store parent2 chromosome} \)
5. \( \text{Off1} \leftarrow \text{declare structure to store offspring1 chromosome} \)
6. \( \text{Off2} \leftarrow \text{declare structure to store offspring2 chromosome} \)
7. \( \text{Run} \leftarrow \text{declare the variable to count the number of epochs} \)
8. \( \text{generation_count} \leftarrow \text{declare the variable to count the number of generations in each run and initialize to zero} \)
9. For Run = 1 to \( m \)
11. [Output,IRR,ORR,ARR,FIS] \leftarrow \text{call GA_Rule() procedure to generate FIS Andits output evaluated for the rule population} \)
12. \text{Call fuzzy_classification() procedure to classify the data with FIS Generated} \)
13. if avg_error return by fuzzy_classification \( \leq \epsilon \) then
14. { \)
15. store the new population generated in Run_output array
16. Increment Run by 1
17. } \)
18. Else
19. { \)
20. Calculate the fitness of each rule from rule population.
21. \[\text{support_rule, total_support_rule, rule_class, support_class}\] \leftarrow \text{Call support_Rule_class() procedure to calculate and return the support for each rule and for each classification class} \)
22. \text{Rule_ratio_importance[]\leftarrow Call rule_ratio_importance() procedure to calculate and return the importance of each rule which is the fitness value of each rule.} \)
23. \( P1 \leftarrow \text{Select parent1 chromosome with best fitness value} \)
24. \( P2 \leftarrow \text{select parent2 chromosome randomly from population} \)
25. \( \text{Crp1, Crp2} \leftarrow \text{Compute crossover points} \)
26. \( \text{Off1,Off2} \leftarrow \text{perform crossover operations on p1 and p2 at the crossover points crp1, crp2 to return offsprings} \)
Chapter 5 - Genetic Tuning of Fuzzy Systems

27. \( d_1, d_2 \leftarrow \text{select rule chromosome number with minimum and next to minimum rule importance} \)

28. if offspring Off1 and Off2 chromosome pattern generated are fully in \( p[] \) then

29. \( \text{flag\_both} \leftarrow \text{Set flag to true} \)

30. if offspring off1 is partially matching with some rule chromosome \( T \) in \( P[] \) then

31. \{ 
32. \hspace{1cm} \text{Call GaRule(), support\_Rule\_class(), rule\_ratio\_importance() procedures for off1} 
33. \hspace{1cm} \text{If rule\_ratio\_importance(off1) > rule\_ratio\_importance(T) then} 
34. \hspace{1.5cm} D1 \leftarrow T 
35. \}

36. if offspring off1 is partially matching with some rule chromosome \( T \) in \( P[] \) then

37. \{ 
38. \hspace{1cm} \text{Call GaRule(), support\_Rule\_class(), rule\_ratio\_importance() procedures for off2} 
39. \hspace{1cm} \text{If rule\_ratio\_importance(off2) > rule\_ratio\_importance(T) then} 
40. \hspace{1.5cm} D2 \leftarrow T 
41. \}

42. If flag\_both == true then
43. no replacement of off1 and off2 with \( d_1 \) and \( d_2 \) in \( P[] \).
44. Else
45. \{ 
46. \hspace{1cm} \text{Delete rule chromosome number \( d_1 \) and \( d_2 \) from \( P[] \)} 
47. \hspace{1cm} \text{Insert off1 and off2 in \( P[] \)} 
48. \}
49. \text{Increment generation count by 1}
50. \text{End For}
51. \text{Return \( P[] \)}
52. \text{End}

Pseudo code 5.1: Genetic\_Rule\_Gen (\( POP, m, n, data, \varepsilon \))

5.2.2.3 Algorithm of procedure GARule: Procedure builds up a FIS automatically based on the input data by constructing the input and output membership functions and also the rule generated. Returns FIS evaluation parameters (output, IRR, ORR, ARR)
Following are the steps of the GARule procedure

Step 1: Input the membership function array created using clustering technique for input and output variables, input data arrays and rule population matrix.

Step 2: Calculate the input membership function degree of input membership functions using ‘evalmf’ matlab function and store it in ‘eval_input’ matrix.

Step 3: Create an FIS clustclassFIS using ‘newfis’ Matlab function
   Step 3.1: Add input/s and output variables to the FIS using ‘addvar’ matlab function.
   Step 3.2: Add input/s and output membership functions to the FIS using ‘addmf’ matlab function with input membership array data generated.
   Step 3.3: Add rule population to the FIS using ‘addrule’ matlab function.

Step 4: Evaluate the FIS created with the input data using ‘evalfis’ MATLAB function which returns following data:
   Output: Is a matrix of size M-by-L, where ‘M’ represents the number of input values specified previously, and ‘L’ is the number of output variables for the FIS[17]
   IRR: The result of evaluating the input values through the membership functions. This matrix is of the size numRules-by-N, where ‘numRules’ is the number of rules, and ‘N’ is the number of input variables [18].
   ORR: The result of evaluating the output values through the membership functions. This matrix is of the size numPts-by-numRules*L, where ‘numRules’ is the number of rules, and ‘L’ is the number of outputs. The first ‘numRules’ columns of this matrix correspond to the first output, the next ‘numRules’ columns of this matrix correspond to the second output, and so forth [18].
ARR: the numPts-by-L matrix of the aggregate values sampled at numPts along the output range for each output[18].

Step 5: End

Pseudo code implementation of the algorithms describe above are given in pseudo code 5.2.

**Pseudo code for procedure GARule(InptMem, data, POP)**

<table>
<thead>
<tr>
<th>Input:</th>
<th>InptMem - array consisting parameters of input and output membership functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>POP</td>
<td>rule population matrix</td>
</tr>
<tr>
<td>data</td>
<td>input records for classification</td>
</tr>
<tr>
<td>Output:</td>
<td>ORR[], ARR[], IRR[], output[], FIS</td>
</tr>
</tbody>
</table>

1. Start
2. clustclassFIS ← declare and initialize FIS variable using newfis() matlab function
3. in1, in2, in3, opt ← declare and add input/s and output variables to the FIS using addvar() matlab function
4. For each input variable in1, in2, in3 add its membership functions with addmf() matlab function using InptMem array.
5. Add the membership function with addmf() matlab function to 'opt' output variable of FIS using InptMem array.
6. Add the rule matrix POP to the FIS using addrulematlab function
7. [output, ORR, IRR, ARR] ← evaluate the FIS created for the input ‘data’ using evalfis() matlab function
8. Return [output, ORR, IRR, ARR] and clustclassFIS
9. End /* end of pseudo code*/

**Pseudo code 5.2: GARule(InptMem, data, POP)**

5.3.3.3 Algorithm of procedure: support_Rule_class(OR, POP, n)

OR : Array consisting the result of evaluating the output values through the membership functions

POP : Rule population

n : Number of rules
Step 1: Declare and initialize array for support_rule(Number of data points supporting individual rule), rule_class(class support for each rule), total_support_rule(total inference values by support_rule), V( is a temporary vector to store all OR values), support_class(Number of data points supporting rule class).

Step 2: Loop through all rows ‘s’ and columns ‘t’ of OR matrix

Step 2.1: if OR[s,t] > 0 then

\[ V[t] = OR[s,t] \]

else

\[ V[t] = -1 \]

Step 2.2: Find the rule no ‘I’ with maximum OR value in vector V

Step 2.3: support_rule[I] = support_rule[I] + 1;

Step 2.4: Total_support_rule[I] = total_support_rule[I] + V[I];

Step 3: Calculate the support_class for each rule \( \forall i \in \{1, 2, ..., n\} \) by using formula

Step 3.1: if support_rule[i] > 0 then

\[ \text{support\_class}[\text{POP}[i,3]] = \text{support\_class}[\text{POP}[i,3]] + \text{support\_rule}[i] \]

Step 4: Calculate the rule_class for each rule by using formula

\[ \text{rule\_class}[i] = \text{support\_class}[\text{POP}[i,3]] \forall i \in \{1, 2, ..., n\} \]

Step 5: End

Pseudo code implementation of the algorithms describe above are given in pseudo code 5.3.

**Pseudo code for procedure Support_Rule_Class (OR, POP, n)**

**Input:**

- OR - array consisting result of evaluating the output values through the membership functions
- POP - rule population
- n - number of rules

**Output:**

- support_rule[], total_support_rule[], rule_class[], support_class[]
1. Start
2. support_rule[] ← declare and initialize an array to store count of data records
3. rule_class[] ← declare and initialize array to store the class supported by each
4. total_support_rule[] ← declare and initialize array to store total support value
5. V ← declare an temporary vector to store all OR values
6. support_class[] ← count of data records supporting rule class.
7. m ← dimension of OR[] row wise.
8. For s=0 to m
9.     For t=0 to n
10.        if (OR[s,t] > 0) then
11.            V[t] ← OR[s,t]
12.        else
13.            V[t] ← -1
14.     End For
15.     I ← rule no with maximum value in V
16.     support_rule[I] ← increment the value of support_rule [I] by 1
17.     total_support_rule[] ← Total_support_rule[] + V[I]
18. End For
19. /* calculate the support_class and rule_class of each rule by using formula*/
20. j ← dimension of POP[] column wise – 2
21. For i = 0 to n
22.     If support_rule[i] > 0 then
23.        support_class(POP[i,j]) ← support_class(POP[i,j]) + support_rule[i]
24.     End For
25. For i=0 to n
26.     rule_class[i] ← support_class(POP[i,j])
27. End For
28. End /* end of pseudo code*/

Pseudo code 5.3: Support_Rule_Class( OR, POP, n)
5.3.3.4 Algorithm of procedure: ratio_importance (SR, RC, TSR, n)

SR: rule support
RC: rule class
TSR: total support for the rule in each class
n: number of rules

Step 1: Declare ratio and importance array for n rules

Step 2: Calculate ratio and importance ∀ i ∈ {1, 2, ..., n})

Step 2.1: if (RC[i] == 0) then
    ratio[i] = SR[i] // if rule class is not defined assign rule support as the ratio for the rule
else
    ratio[i] = SR[i]/RC[i] // ratio will give the number of data points supporting rule with the rule class

Step 2.2: importance[i] = ratio[i] * TSR[i] // if ratio is bigger more the importance of the rule

Step 3: Return ratio and importance

Step 4: End

Pseudo code implementation of the algorithms describe above are given in pseudo code 5.4.

**Pseudo code for procedure Rule_ratio_importance (SR, RC, TSR, n)**

*Input:*
SR: array consisting of count of data records supporting individual rule.
RC: array with the class supported by each rule.
TSR: array consisting of total support value of each rule
n: number of rules

*Output: ratio[] and importance[]*

1. Start
2. ratio[] ← declare and initialize array to store support ratio obtained for n rules
3. importance[] ← declare and initialize array to store importance of each
4. For i = 1 to n
5. If RC[i] == 0 then
6. ratio[i] ← SR[i]
7. Else
8. ratio\[i\] ← SR\[i\]/RC\[i\]
9. End if
10. Importance\[i\] ← ratio\[i\]* TSR\[i\]
11. End
12. Return ratio and importance
13. End /* end of pseudo code*/

**Pseudo code 5.4: Rule\_ratio\_importance(SR,RC,TSR, n)**

### 5.4 Result and Conclusion of Genetic Tuning of FIS rules:
The GA tuning of FIS rule base was tested with student data set on fuzzy tool box of MATLAB and Genetic Algorithm is implemented using MATLAB programming. The GA Population comprises of the fuzzy rule list with candidate rules coded in the form of ‘if-then’ rules as given in Table 5.1

<table>
<thead>
<tr>
<th>Rule</th>
<th>Input 1</th>
<th>Input 2</th>
<th>Input 3</th>
<th>Class</th>
<th>Rule 1</th>
<th>Rule 2</th>
<th>Rule 3</th>
<th>Rule 4</th>
<th>Implication</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Parent chromosome 1:

<table>
<thead>
<tr>
<th>23</th>
<th>4</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Parent chromosome 2:

| 3  | 4 | 4 | 5 | 1 | 1 |

Cross over points: two point crossover operation at position 2 – 3

Offspring chromosome 1:

| 2  | 4 | 4 | 3 | 1 | 1 |

Offspring chromosome 2:

| 3  | 3 | 3 | 5 | 1 | 1 |

The rules generated using GA is shown Table 5.2 with each rule antecedent, consequent and rule importance also the plot of results obtained against the target is shown in figure 5.4 with the average error. The developed GA algorithm developed with rule importance as fitness criteria and with termination criteria as combination of specified generations and accepted classification error and has shown consistent results. The system was tested for 10 epochs. It is been observed that with the initial population of 64 rules, only 17 rules have been fired with good rule significance in classifying the items.
Table 5.2: Rules generated using GA with its rule importance

<table>
<thead>
<tr>
<th>Input1</th>
<th>Input 2</th>
<th>Input 3</th>
<th>Class</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0.010752688</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0.267904481</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1.525621976</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0.267904481</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0.033335792</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0.004909984</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0.127880184</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2.194776828</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1.865060865</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>12.22235971</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>9.4723093</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>2.194776828</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0.951977401</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>7.11017871</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>19.54803689</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5.918618294</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>13.63920911</td>
</tr>
</tbody>
</table>

Figure 5.4: Fuzzy classification plot using GA
5.5 The Genetic Tuning Of Fuzzy Membership Functions

The genetic tuning algorithm aims to optimize FIS fuzzy model for classification by creating the population of range values of membership function. The range values are encoded in the chromosome as real numbers representing the range values of membership function. Each chromosome represents spread of fuzzy membership function. Figure 5.5 schematically depicts the overall functional flow of genetic MF tuning process of FIS desired for building FIS for classification problem.

The output membership functions are tuned manually before FIS is built and evaluated in the run process of Genetic Tuning Algorithm. Tables 5.3, 5.4 and 5.5 shows how a chromosome appears in the parameter population for input membership functions.

Figure 5.5 Functional flow of process for genetic tuning of FIS MF

<table>
<thead>
<tr>
<th>Table 5.3 Chromosome of Input Membership Function for ‘In’</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Internal marks (In)</strong></td>
</tr>
<tr>
<td><strong>NA</strong></td>
</tr>
<tr>
<td>Extreme left</td>
</tr>
<tr>
<td>-5</td>
</tr>
</tbody>
</table>
Table 5.4 Chromosome of Input Membership Function for ‘Term’

<table>
<thead>
<tr>
<th>Internal marks (In)</th>
<th>NA</th>
<th>Low</th>
<th>Med</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extreme left</td>
<td>Top left</td>
<td>Top right</td>
<td>Extreme right</td>
<td>Left vertex</td>
</tr>
<tr>
<td>-5</td>
<td>-5</td>
<td>-3</td>
<td>0</td>
<td>a2</td>
</tr>
</tbody>
</table>

Table 5.5 Chromosome of Input Membership Function for ‘Prac’

<table>
<thead>
<tr>
<th>Internal marks (In)</th>
<th>NA</th>
<th>Low</th>
<th>Med</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extreme left</td>
<td>Top left</td>
<td>Top right</td>
<td>Extreme right</td>
<td>Left vertex</td>
</tr>
<tr>
<td>-5</td>
<td>-5</td>
<td>-3</td>
<td>0</td>
<td>a3</td>
</tr>
</tbody>
</table>

For each input variable the chromosome specifies triangular fuzzy sets with amendable left vertex \((a_m, c_m)\) and right vertex \((b_m, d_m)\) values along with center values \((c_{m1}, c_{m2})\). That of trapezoidal fuzzy set, the extreme left \((e_m)\) and top left \((c_{m3})\) values. The subscript \(m=1, 2, 3\) depicts input variable number shown in Figure 5.6.

![Figure 5.6: Input variables Membership Functions parameter](image)

The fuzzy partitions of output variable ‘programming_skill’ is specified with \(n=1, 2, 3, 4, 5\) with two extreme end trapezoidal fuzzy sets and three inner triangular fuzzy sets having four and three parameters respectively. The output variable MFs are depicted in Figure 5.7 and the set of rules mapping the inputs over to output are as given in Table 5.6.
The genetic tuning of FIS MFs to improve the performance of classification task is done in two phases as follows:

1. Create Initial chromosome population
2. Genetic algorithm to tune input MFs
The first phase to create initial input MFs chromosome population is been accomplished by ‘CreateMFPop procedure’ implemented through MATLAB programming.

5.5.1 Algorithm of CreateMFPop procedure: The ‘CreateMFPop( )’ creates population input variables of the FIS, each of size 10. The pseudo code of ‘CreateMFPop( )’ procedure algorithm are enumerated as follows in pseudo code 5.5.

Pseudo code : CreateMFPop( )

Output: pop1,pop2 and pop3 chromosome population of three input variables

1. Start
2. /* pop1 MF chromosome generation
3. pop1, pop2,pop3 ← Declare and initial the structures to store population of ‘in’, ‘term’ and ‘prac’ input variables population, respectively.
4. count ← initial the population count to 2
5. loop while ( count <= 10)
6. {
7. /*first triangular fuzzy MF parameters a1,c11 and b1
8. a1 ← choose a random integer value between -5 and 0
9. b1 ← choose a random integer value between 5 and 15
10 /* Check if minimum distance between a1 and b1 >= 5 */
11 While (distance between a1 and b1 < 5 )
12 {
13 b1 ← choose another random integer value between 5 and 15
14 }
15 c11 ← calculate the mean value between a1 and b2 to form the center peak of triangular MF
/*second triangular fuzzy MF parameters c1,c12 and d1

\[c1\leftarrow \text{choose a random integer value between values of } c11 \text{ and } b1\]

/*check if adjacent MFs are overlapping */

While (distance between b1 and c1 <3)

{ \[b1\leftarrow \text{increment the } b1 \text{ value by 3}\] }

/* recalculate the center c11and c1values

\[c11\leftarrow \text{calculate the mean value between a1 and b1 to form the center peak of triangular MF}\]

\[c1\leftarrow \text{choose a random integer value between values of } c11 \text{ and } b1\]

}  

\[d1\leftarrow \text{choose a random integer value between values of } b1 \text{ and } 22\]

/* Check if minimum distance between c1 and d1 is greater or equal to 5 */

While (distance between c1 and d1 <5)

{ \[d1\leftarrow \text{choose another random integer value between } b1 \text{ and } 22\] }

\[c12\leftarrow \text{calculate the mean value between d1 and c1 to form the center peak of triangular MF}\]

/* right trapezoidal fuzzy MF parameters e1 and c13

\[e1\leftarrow \text{choose a random integer value between values of } c12 \text{ and } d1\]

/*check if adjacent MFs are overlapping */

While (distance between d1 and e1 <3)

{ \[d1\leftarrow \text{increment the } d1 \text{ value by 3}\] }

/* recalculate the center c12 and e1 values
40  \( \text{c12} \leftarrow \text{calculate the mean value between} \ d1 \ \text{and} \ \text{c1} \ \text{to form the center peak of triangular MF} \)
41  \( \text{e1} \leftarrow \text{choose a random integer value between values of} \ \text{c12} \ \text{and} \ d1 \)
42  \}
43  \( \text{c13} \leftarrow \text{calculate the mean value between} \ d1 \ \text{and theoretical maximum value ‘in’ i.e 30 to form the top left of trapezoidal MF} \)
44  \ Insert the generated chromosome string to the population pop1 as
45  \ [-5 -5- 30; a1 \ c11 \ b1; c1 \ c12 \ d1; e1 \ c13 \ 30 \ 30]\)
46  \( \text{count} \leftarrow \text{increment count by 1} \)
47  \}
48  /* pop2 MF chromosome generation */
49  \( \text{count} \leftarrow \text{initial the population count to 2} \)
50  \ loop while ( count \ <= \ 10) \}
51  /*first triangular fuzzy MF parameters a2,c21 and b2 */
52  \( \text{a2} \leftarrow \text{choose a random integer value between} \ -5 \ \text{and} \ 0 \)
53  \( \text{b2} \leftarrow \text{choose a random integer value between} \ 5 \ \text{and} \ 35 \)
54  /* Check if minimum distance between a2 and b2 is greater or equal to 5 */
55  \ While (distance between a2 and b2 \ < 5) \}
56  \}
57  \( \text{b2} \leftarrow \text{choose another random integer value between} \ 5 \ \text{and} \ 35 \)
58  \}
59  \( \text{c21} \leftarrow \text{calculate the mean value between} \ a2 \ \text{and} \ b2 \ \text{to form the center peak of triangular MF} \)
60  /*second triangular fuzzy MF parameters c2,c22 and d2 */
61  \( \text{c2} \leftarrow \text{choose a random integer value between values of} \ \text{c21} \ \text{and} \ b2 \)
62  /*check if adjacent MFs are overlapping */
While (distance between \( b_2 \) and \( c_{12} < 3 \))

\[
\begin{align*}
\text{\( b_2 \)} & \leftarrow \text{increment the \( b_2 \) value by 3} \\
/* \text{recalculate the center} \ c_{21} \text{and} \ c_2 \text{values} */ \\
\text{\( c_{21} \)} & \leftarrow \text{calculate the mean value between} \ a_2 \text{and} \ b_2 \text{to form the center peak of triangular MF} \\
\text{\( c_2 \)} & \leftarrow \text{choose a random integer value between values of} \ c_{21} \text{and} \ b_2 \\
\end{align*}
\]

\[
\begin{align*}
\text{\( d_2 \)} & \leftarrow \text{choose a random integer value between values of} \ b_2 \text{and 55} \\
/* \text{Check if minimum distance between} \ c_2 \text{and} \ d_2 \text{is greater or equal to 5} */ \\
\text{While} \ (\text{distance between} \ c_2 \text{and} \ d_2 < 5) \\
\begin{align*}
\text{\( d_2 \)} & \leftarrow \text{choose another random integer value between} \ b_2 \text{and 55} \\
\end{align*}
\]

\[
\begin{align*}
\text{\( c_{22} \)} & \leftarrow \text{calculate the mean value between} \ d_2 \text{and} \ c_2 \text{to form the center peak of triangular MF} \\
/* \text{right trapezoidal fuzzy MF parameters} \ e_2 \text{and} \ c_{23} */ \\
\text{\( e_2 \)} & \leftarrow \text{choose a random integer value between values of} \ c_{22} \text{and} \ d_2 \\
/* \text{check if adjacent MFs are overlapping} */ \\
\text{While} \ (\text{distance between} \ d_2 \text{and} \ e_2 < 3) \\
\begin{align*}
\text{\( d_2 \)} & \leftarrow \text{increment the} \ d_2 \text{value by 3} \\
/* \text{recalculate the center} \ c_{22} \text{and} \ e_2 \text{values} */ \\
\text{\( c_{22} \)} & \leftarrow \text{calculate the mean value between} \ d_2 \text{and} \ c_2 \text{to form the center peak of triangular MF} \\
\text{\( e_2 \)} & \leftarrow \text{choose a random integer value between values of} \ c_{22} \text{and} \ d_2 \\
\end{align*}
\]

Data Mining: A Soft Computing Approach

205
c23 ← calculate the mean value between d1 and theoretical maximum value 'term' i.e 70 to form the top left of trapezoidal MF
Insert the generated chromosome string to the population pop2 as [-5 -5 -30; a2 c21 b2;c2 c22 d2; e2 c23 70 70]
count ← increment count by 1
} /* pop3 MF chromosome generation *
count ← initial the population count to 2
loop while ( count <= 10)
{
/*first triangular fuzzy MF parameters a3,c31 and b3
a3← choose a random integer value between -5 and 0
b3← choose a random integer value between 5 and 25
*/ Check if minimum distance between a3 and b3 is >= 5 */
While (distance between a3 and b3 < 5)
{
b3← choose another random integer value between 5 and 25
}
c31 ← calculate the mean value between a3 and b3 to form the center peak of triangular MF
/*second triangular fuzzy MF parameters c3,c32 and d3
c3← choose a random integer value between values of c31and b3
*/check if adjacent MFs are overlapping */
While (distance between b3 and c32<3)
{
b3← increment the b3 value by 3
/* recalculate the center \( c_3 \) and \( c_3 \) values

\[
c_31 \leftarrow \text{calculate the mean value between } a_3 \text{ and } b_3 \text{ to form the center peak of triangular MF}
\]

\[
c_3 \leftarrow \text{choose a random integer value between values of } c_31 \text{ and } b_3
\]

\[
d_3 \leftarrow \text{choose a random integer value between values of } b_3 \text{ and } 38
\]

/* Check if minimum distance between \( c_3 \) and \( d_3 \) is greater or equal to 5 */

\[
\text{While (distance between } c_3 \text{ and } d_3 \text{ < 5)}
\]

\[
d_3 \leftarrow \text{choose another random integer value between } b_3 \text{ and } 38
\]

\[
c_32 \leftarrow \text{calculate the mean value between } d_3 \text{ and } c_3 \text{ to form the center peak of triangular MF}
\]

/* right trapezoidal fuzzy MF parameters \( e_3 \) and \( c_33 \)

\[
e_3 \leftarrow \text{choose a random integer value between values of } c_32 \text{ and } d_3
\]

/* check if adjacent MFs are overlapping */

\[
\text{While (distance between } d_3 \text{ and } e_3 \text{ < 3)}
\]

\[
d_3 \leftarrow \text{increment the } d_3 \text{ value by 3}
\]

/* recalculate the center \( c_32 \) and \( e_3 \) values

\[
c_32 \leftarrow \text{calculate the mean value between } d_3 \text{ and } c_3 \text{ to form the center peak of triangular MF}
\]

\[
e_3 \leftarrow \text{choose a random integer value between values of } c_32 \text{ and } d_3
\]

\[
\}
\]

\[
c_33 \leftarrow \text{calculate the mean value between } d_1 \text{ and theoretical maximum value 'prac' i.e 50 to form the top left of trapezoidal MF}
\]
On completion of first phase the initial population of input variables are formed and is passed to second phase of genetic tuning. This phase comprises of calls to many procedures listed as follows. 

i. GA_Fuzzy_Mem_tune_algo(): the procedure to implement GA algorithm 

ii. Ga_Fuzzy_Mem_Tune(): the procedure to build FIS

iii. GA_Fuzzy_Mem_tune_fitness(): the procedure evaluate the classification error of FIS 

iv. ga_mf_operation(): the procedure to apply GA crossover operation to generate new offspring chromosomes

v. fuzzclass_MFtune(): to check the classification error of generated FIS.

All these procedures are implemented as ‘.m script’ files through MATLAB programming.

5.5.2 Algorithm of GA_Fuzzy_Mem_tune_algo procedure: The pseudo code of ‘GA_Fuzzy_Mem_tune_algo’ procedure is enumerated as in pseudo code 5.6.
Pseudo code for procedure **GA_Fuzzy_Mem_tune_algo**

Input:

- Pop1 - input ‘in’ population set of MFs chromosomes
- Pop2 - input ‘term’ population set of MFs chromosomes
- Pop2 - input ‘prac’ population set of MFs chromosomes
- o_mem – output ‘programming_skill’ MFs parameter set
- data – input dataset

Output:

- gen_f - an array with best tuned FIS each generation
- gen_o - an array with min average classification error of best tuned FIS

1. Start
2. gen_f ← declare structure to store generation wise FIS
3. Gen_o ← declare structure to store generation wise minimum average
4. Gen ← initial generation count to 1
5. Loop while (gen <= m)
6. |
7. /* extract MF parameters from population sets of each input variable and perform all combinations */
8. for (i = 1 to 10 )
9.  |
10.  for (j = 1 to 10 )
11.  |
12.  |
13.  Input1_mem[] ← extract and store MF chromosome parameters from pop1(i)
14.  Input2_mem[] ← extract and store MF chromosome parameters from pop2(j)
15.  Input3_mem[] ← extract and store MF chromosome parameters from pop3(k)
16.  |
17. /* pass the extracted parameters to Ga_Fuzzy_Mem_Tune to build FIS */
18. genfis ← call the Ga_Fuzzy_Mem_Tune (input1_mem,input2_mem,input3_mem, o_mem, rule) procedure to
18. \( \text{fis}(i,j,k) \leftarrow \text{store the generated FIS genfis} \)
19. \}
20. \}
21. \/* \text{check the fuzzy classification error of FIS generated} */
22. \( \text{min_avg_error} \leftarrow \text{call the GA_Fuzzy_Mem_tune_fitness( fis, data, target) procedure to return the minimum average classification error} \)
23. \( \text{gen_o}[\text{gen}] \leftarrow \text{store the min_avg_error for that generation} \)
24. \( \text{gen_f}[\text{gen}] \leftarrow \text{store the tuned FIS with min_avg_error for that} \)
25. \/* \text{check if termination condition satisfied} */
26. \( \text{if}(\text{gen_o}[\text{gen}] < \text{Err}) \)
27. \( \text{break;} \)
28. \( \text{Else} \)
29. \}
30. \/* \text{call ga_mf_operation to generate new offspring chromosomes using} \)
31. \( \text{Pop1} \leftarrow \text{call ga_mf_operation(pop1, data, 1) to return new population} \)
32. \( \text{Pop2} \leftarrow \text{call ga_mf_operation(pop2, data, 2) to return new population} \)
33. \( \text{Pop3} \leftarrow \text{call ga_mf_operation(pop3, data, 2) to return new population} \)
34. \}
35. \( \text{gen} \leftarrow \text{increment gen count by 1} \)
36. \} /* \text{end the outer loop} */
37. \End

**Pseudo Code 5.6: GA_Fuzzy_Mem_tune_algo**

**5.5.3 Algorithm of GA_Fuzzy_Mem_Tune procedure:** The ‘Ga_Fuzzy_Mem_Tune( )’ procedure is implemented to build the FIS from the input variables MFs parameter along with output variable parameters and rule base. The pseudo code of the procedure is listed in pseudo code 5.7.

The pseudo code of procedure

**Ga_Fuzzy_Mem_Tune(inpt1_mem, inpt2_mem, inpt3_mem, o_mem, rule)**

**Input:**
- **inpt1_mem** – array with input 1 variable MFs parameters
- **inpt2_mem** – array with input 2 variable MFs parameters
- **inpt3_mem** – array with input 3 variable MFs parameters
- **o_mem** – array with output variable MFs parameters
- **rule** – array with rule base parameters
Inpt3_mem – array with input 3 variable MFs parameters
o_mem - array with output variable MFs parameters
rule – rule base with FIS inference rule

Output : a – structure of FIS built

1. Start
2. \( a \leftarrow \) create a FIS with mamdani inference using newfis('GA_MF_TUNE_FIS', 'mamdani', 'min', 'max', 'min', 'max', 'centroid') function.
3. Add input variables ‘in’, ‘term’ and ‘prac’ to the FIS ‘a’ using addvar() function
4. Add Inpt1_mem[], Inpt2_mem[] and Inpt3_mem MFs parameters to the corresponding input variables using addmf().
5. Add output variable ‘programming_skill’ to the FIS ‘a’ using addvar() function
6. Add o_mem MFs parameter to the output variable using addmf().
7. Add rulebase rule to the FIS ‘a’ using addrule() function.
8. Return built FIS ‘a’
9. End

Pseudo Code 5.7 : Ga_Fuzzy_Mem_Tune

5.5.4 Algorithm of GA_Fuzzy_Mem_tune_fitness procedure:
Each FIS built is checked and if it satisfying the termination criteria using ‘GA_Fuzzy_Mem_tune_fitness()’ procedure which in turn calls ‘Fuzzclass_MFtune( )’ to classify the input data against the target data provided. ‘Fuzzclass_MFtune( )’ returns the minimum average classification error. The pseudocode enumerating the steps of ‘GA_Fuzzy_Mem_tune_fitness( )’ is listed in Pseudo code 5.8.

Pseudo code for procedure GA_Fuzzy_Mem_tune_fitness(fis, input, target)

Input:
fis- array with FIS built through all 10 x10 x 10 input combination
input – input data
target – target data

Output:
Posi, posj,posk – index position of three input variables MF population comprising the FIS structure
min_avg_error – minimum average classification error calculated through for the given FIS

1. Start
2. for ( i= 1 to 10)
3. {   for (j=1 to 10)
4.     {    for(k=1 to 10)
5.       {
6.         [ avg_error, correct, incorrect] ← call fuzzclass_MFtune( ) with extracted FIS at position i,j,k of fis array, along with input and target data.
7.         fit_avg_error[i,j,k] ← store the avg_error calculated.
8.     } } }
9. [min_avg_error, I] ← find the minimum average error among the average error of different FIS and the index position of the FIS with Minimum average error.
10. End

Pseudo Code  5.8 : GA_Fuzzy_Mem_tune_fitness

The code of ‘Fuzzclass_MFtune( )’ used to classify data set against target data is listed in Appendix .

5.5.5 Algorithm of ga_mf_operation procedure:
The ‘ga_mf_operation( )’ procedure incorporates the process of generating new population. Offsprings are generated by performing crossover operation on selected parent chromosomes. The parent chromosomes are selected, based on the fitness criteria. The fitness criteria used in this study is membership function evaluation value which exhibits how good the MFs maps the input data . To
evaluate the input variable MFs; ‘evalmf()’ MATLAB FUZZY tool function is used. The offspring produced by crossover operation on parent chromosomes are checked for validity. Offspring are checked if already existing in current population, if so then offsprings are ignored otherwise are inserted in population by replacing the worst fit parent chromosomes. The pseudo code of ga_mf_operation() procedure is listed in Pseudo Code 5.9.

**Pseudo code for procedure ga_mf_operation(pop1,data,col)**

**Input:**
- Pop1- input MF population
- Data- the input data set
- Col- index position of column within data with input variable data

**Output:**
- Pop1 – new population

1. Start
2. Eval_input[ ] ←Extract each chromosome from population pop1 with MFs parameter using evalmf() function for each ‘X’ data corresponding the
3. For each MF chromosomes from POP1
4. {
5.   inpt1_avg[]←Calculate the average of all evalmf() values return for each
6. }  
7. /*selecting the good membership function chromosomes as parent chromosomes*/
8. Parent1← maximum value chromosome in inpt1_avg[]
9. Parent2← next Ma maximum value chromosome in inpt1_avg[]
10. /* choose the crossover points, set the 3 fixed crossover points */
11. cr1=4; % second row fourth column
12. cr2=2; % third row second column
13. cr3=1 % fourth row first column
14. [Offspring1, offspring2]← Exchange the values of parent1 and parent2 at cr1, cr2, cr3 position
15. /* check if valid mf offsprings */
16. If (parameters of offspring MF are according to MF construction rules) then return true
17. If (offsprings valid )
18. /* check if offsprings present in current population*/
19. If (offspring present in Pop1 )
20. { ignore offspring(s), do not add to new population pop1
21. }
22. Else
23. {
24. Pop1[ ]←Insert offspring(s) in new population pop1 by replacing the worst chromosome in population
25. }
26. Return pop1
27. End

Pseudo Code 5.9 :ga_mf_operation

5.6 Result and Conclusion Genetic Tuning of MF’s: The implemented algorithm is tested for programming_skill classification as discussed in previous chapters. The initial MF chromosome population for input variables ‘in’, ‘term’ and ‘prac’ created by CreateMFPop procedure are shown in Table 5.6, 5.7 and 5.8 respectively. Each chromosome is 4x4 matrix containing membership range values.

Table 5.6: MF chromosome population of ‘in’ input variable of FIS

<table>
<thead>
<tr>
<th>Sr. No</th>
<th>Input Variable MF chromosome of ‘in’</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NA</td>
</tr>
<tr>
<td>1</td>
<td>[-5 -5 -3 0]</td>
</tr>
<tr>
<td>2</td>
<td>[-5 -5 -3 0]</td>
</tr>
<tr>
<td>3</td>
<td>[-5 -5 -3 0]</td>
</tr>
<tr>
<td>4</td>
<td>[-5 -5 -3 0]</td>
</tr>
<tr>
<td>5</td>
<td>[-5 -5 -3 0]</td>
</tr>
</tbody>
</table>
Chapter 5 - Genetic Tuning of Fuzzy Systems

Table 5.7: MF chromosome population of ‘term’ input variable of FIS

<table>
<thead>
<tr>
<th>Sr. No</th>
<th>Input Variable MF chromosome of ‘term’</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NA</td>
</tr>
<tr>
<td>1</td>
<td>[-5 -5 -3 0]</td>
</tr>
<tr>
<td>2</td>
<td>[-5 -5 -3 0]</td>
</tr>
<tr>
<td>3</td>
<td>[-5 -5 -3 0]</td>
</tr>
<tr>
<td>4</td>
<td>[-5 -5 -3 0]</td>
</tr>
<tr>
<td>5</td>
<td>[-5 -5 -3 0]</td>
</tr>
<tr>
<td>6</td>
<td>[-5 -5 -3 0]</td>
</tr>
<tr>
<td>7</td>
<td>[-5 -5 -3 0]</td>
</tr>
<tr>
<td>8</td>
<td>[-5 -5 -3 0]</td>
</tr>
<tr>
<td>9</td>
<td>[-5 -5 -3 0]</td>
</tr>
<tr>
<td>10</td>
<td>[-5 -5 -3 0]</td>
</tr>
</tbody>
</table>

Table 5.8: MF chromosome population of ‘prac’ input variable of FIS

<table>
<thead>
<tr>
<th>Sr. No</th>
<th>Input Variable MF chromosome of ‘prac’</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NA</td>
</tr>
<tr>
<td>1</td>
<td>[-5 -5 -3 0]</td>
</tr>
<tr>
<td>2</td>
<td>[-5 -5 -3 0]</td>
</tr>
</tbody>
</table>
The output variable ‘programming_skill’ MF parameters given to the algorithm is shown in Table 5.9

<table>
<thead>
<tr>
<th>3</th>
<th>[-5 -5 -3 0]</th>
<th>[-1 8 17]</th>
<th>[9 16 22]</th>
<th>[17 36 50 50]</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>[-5 -5 -3 0]</td>
<td>[-5 9 23]</td>
<td>[17 24 31]</td>
<td>[25 40.5 50 50]</td>
</tr>
<tr>
<td>5</td>
<td>[-5 -5 -3 0]</td>
<td>[-3 5 12]</td>
<td>[8 17 25]</td>
<td>[20 37.5 50 50]</td>
</tr>
<tr>
<td>6</td>
<td>[-5 -5 -3 0]</td>
<td>[-1 16 12]</td>
<td>[6 20 33]</td>
<td>[25 41.5 50 50]</td>
</tr>
<tr>
<td>7</td>
<td>[-5 -5 -3 0]</td>
<td>[-4 5 13]</td>
<td>[5 14 22]</td>
<td>[19 36 50 50]</td>
</tr>
<tr>
<td>8</td>
<td>[-5 -5 -3 0]</td>
<td>[-5 2 9]</td>
<td>[4 19 33]</td>
<td>[19 41.5 50 50]</td>
</tr>
<tr>
<td>9</td>
<td>[-5 -5 -3 0]</td>
<td>[-2 9 18]</td>
<td>[12 20 29]</td>
<td>[21 39 50 50]</td>
</tr>
<tr>
<td>10</td>
<td>[-5 -5 -3 0]</td>
<td>[-5 3 10]</td>
<td>[3 9 15]</td>
<td>[10 32.5 50 50]</td>
</tr>
</tbody>
</table>

The algorithm is executed for 20 generation, each generation has 10x10x10 combinations of input MF variables, a total of 1000 combination each generation. Among these combination, the best tuned FIS MF chromosome is selected for the generation. The results of some of the best tuned FIS generated for classification in terms of average error using are shown in Table 5.10.
Table 5.10: Classification results of FIS

<table>
<thead>
<tr>
<th>‘in’ MF parameters</th>
<th>‘term’ MF parameters</th>
<th>‘prac’ MF parameters</th>
<th>Average Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>-5 -5 -3 0</td>
<td>-5 -5 -3 0</td>
<td>-5 -5 -3 0</td>
<td>0.0591</td>
</tr>
<tr>
<td>-3 6 12</td>
<td>-3 15 30</td>
<td>-3 8 15</td>
<td></td>
</tr>
<tr>
<td>8 16 22</td>
<td>22 35 50</td>
<td>12 25 35</td>
<td></td>
</tr>
<tr>
<td>17 24 30 30</td>
<td>43 56 70 70</td>
<td>31 38 50 50</td>
<td></td>
</tr>
<tr>
<td>-5 -5 -3 0</td>
<td>-5 -5 -3 0</td>
<td>-5 -5 -3 0</td>
<td>0.0746</td>
</tr>
<tr>
<td>-3 6 11</td>
<td>-2 16 33</td>
<td>-2 9 18</td>
<td></td>
</tr>
<tr>
<td>7 13 20</td>
<td>21 36 50</td>
<td>12 24 31</td>
<td></td>
</tr>
<tr>
<td>18 24 30 30</td>
<td>47 60 70 70</td>
<td>23 40 50 50</td>
<td></td>
</tr>
<tr>
<td>-5 -5 -3 0</td>
<td>-5 -5 -3 0</td>
<td>-5 -5 -3 0</td>
<td>0.0771</td>
</tr>
<tr>
<td>-3 6 14</td>
<td>-2 16 30</td>
<td>-2 9 23</td>
<td></td>
</tr>
<tr>
<td>9 13 20</td>
<td>22 36 50</td>
<td>17 20 29</td>
<td></td>
</tr>
<tr>
<td>17 24 30 30</td>
<td>43 60 70 70</td>
<td>25 39 50 50</td>
<td></td>
</tr>
<tr>
<td>-5 -5 -3 0</td>
<td>-5 -5 -3 0</td>
<td>-5 -5 -3 0</td>
<td>0.0664</td>
</tr>
<tr>
<td>-2 7 11</td>
<td>-1 16 33</td>
<td>-2 9 14</td>
<td></td>
</tr>
<tr>
<td>7 16 23</td>
<td>18 32 46</td>
<td>7 20 29</td>
<td></td>
</tr>
<tr>
<td>18 26.5 30 30</td>
<td>39 58 70 70</td>
<td>17 39 50 50</td>
<td></td>
</tr>
<tr>
<td>-5 -5 -3 0</td>
<td>-5 -5 -3 0</td>
<td>-5 -5 -3 0</td>
<td>0.0692</td>
</tr>
<tr>
<td>-1 8 19</td>
<td>-3 15 33</td>
<td>-1 7 18</td>
<td></td>
</tr>
<tr>
<td>15 16 21</td>
<td>21 35 50</td>
<td>12 15 23</td>
<td></td>
</tr>
<tr>
<td>20 25.5 30 30</td>
<td>47 56 70 70</td>
<td>21 36.5 50 50</td>
<td></td>
</tr>
<tr>
<td>-5 -5 -3 0</td>
<td>-5 -5 -3 0</td>
<td>-5 -5 -3 0</td>
<td>0.0496</td>
</tr>
<tr>
<td>-1 6 12</td>
<td>-2 9 33</td>
<td>-1 7 14</td>
<td></td>
</tr>
<tr>
<td>6 15 24</td>
<td>25 36 49</td>
<td>12 23 30</td>
<td></td>
</tr>
<tr>
<td>16 27 30 30</td>
<td>47 59.5 70 70</td>
<td>28 36.5 50 50</td>
<td></td>
</tr>
</tbody>
</table>
The GA tuned input MF’s are shown in Figure 5.8 and the classification plot is shown in Figure 5.9.

![Figure 5.8: GA tuned input MF’s](image)

<table>
<thead>
<tr>
<th>Input MF</th>
<th>Input term MF</th>
<th>Output programming kill MF</th>
</tr>
</thead>
<tbody>
<tr>
<td>-5 -5 -3 0</td>
<td>-5 -5 -3 0</td>
<td>-5 -5 -3 0</td>
</tr>
<tr>
<td>-1 8 19</td>
<td>-1 16 30</td>
<td>-3 8 18</td>
</tr>
<tr>
<td>15 16 21</td>
<td>22 32 46</td>
<td>12 25 35</td>
</tr>
<tr>
<td>20 30.5 30</td>
<td>43 58 70 70</td>
<td>23 38 50 50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input prac MF</th>
<th>Output programming.kill MF</th>
</tr>
</thead>
<tbody>
<tr>
<td>-5 -5 -3 0</td>
<td>-5 -5 -3 0</td>
</tr>
<tr>
<td>-1 9 15</td>
<td>-2 16 33</td>
</tr>
<tr>
<td>11 19 23</td>
<td>18 36 50</td>
</tr>
<tr>
<td>16 26.5 30</td>
<td>39 60 70 70</td>
</tr>
<tr>
<td>31 40.5 50 50</td>
<td>0.0559</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input MF</th>
<th>Input term MF</th>
<th>Output programming kill MF</th>
</tr>
</thead>
<tbody>
<tr>
<td>-5 -5 -3 0</td>
<td>-5 -5 -3 0</td>
<td>-5 -5 -3 0</td>
</tr>
<tr>
<td>-1 9 15</td>
<td>-2 16 33</td>
<td>-5 9 15</td>
</tr>
<tr>
<td>11 19 23</td>
<td>18 36 50</td>
<td>12 24 31</td>
</tr>
<tr>
<td>16 26.5 30</td>
<td>39 60 70 70</td>
<td>31 40.5 50 50</td>
</tr>
<tr>
<td>0.0579</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 5.9: Classification Plot with GA tuned input MF

The best tuned MF that has been achieved is the FIS with average error 0.0496.
References:


