Parallel Distributed Multi-objective Fuzzy Genetics-based Machine Learning Final Report

Bowen Zheng, Shijie Chen, Shuxin Wang
Department of Computer Science and Engineering
Southern University of Science and Technology
Shenzhen, Guangdong, China

***Abstract*—In this project, we finished an asynchronous parallel implementation of a 3-objective fuzzy GBML classifier. It's composed of a fuzzy-rule based classifier, a genetics based optimization framework and an asynchronous parallel scheme. Compared to non-parallel implementation, our model could obtain reasonable results with significantly reduced computation time. Our model is also about 10% faster than prior synchronized model of similar performance.**

I. Introduction

In this project, we aim to build a parallel distributed implementation of a multi-objective genetics based machine learning(GBML) algorithm. We choose a specific problem of a three-objective fuzzy rule-based classifier and fit it into a hybrid GBML framework. Then we develop a parallel mechanism to accelerate computation.

Code of the three parts have been completed. We will integrate them together, fix bugs and run some test problems in the next stage.

II. Fuzzy Rule-based Classifiers

The design and implementation of fuzzy classifier is based on [1].

A. Fuzzy Rules

Variable	Name
n	dimension of patterns
M	number of classes of patterns
S	a fuzzy classifier
I	number of membership functions
x_p	a pattern vector
x_{pi}	attribute of x_p value on i-th dimension
A_{qi}	antecedent fuzzy set
$\mu_{A_{qi}}(x)$	membership function of A_{qi}
$\mu_{A_q}(x_p)$	compatibility grade of x_p with A_q
A_q	antecedent part of q-th rule
R_q	q-th rule
C_q	consequent class for q-th rule
CF_q	rule weight for q-th rule

TABLE I: Notation

We use the following "if-then" rules:

$$Rule\ R_q:\ if\ x_{pi}\ is\ A_{qi},\ then\ Class\ C_q\ with\ CF_q \tag{1}$$

Input vectors are normalized to a hypercube $[0,1]^n$ using the following equation:

$$x_{pi} = \frac{x_{pi} - min(x_i)}{max(x_i) - min(x_i)} \tag{2}$$

The antecedent fuzzy set contains a membership function of the form:

$$\mu_{A_{qi}}(x_{pi}) = max\{1 - \frac{|a - x_{pi}|}{b}, 0\} \tag{3}$$

$$a = \frac{k-1}{K-1} \tag{4}$$

$$b = \frac{1}{K-1} \tag{5}$$

where K is the number of intervals of fuzzy set A_{qi} and k is the order of the interval. As shown in Fig.1 and Fig.2. The $don't\ care$ condition is a constant function with value 1. As we will discuss in classification process, feature x_{pi} with A_{qi} being $don't\ care$ is ignored by rule R_q.

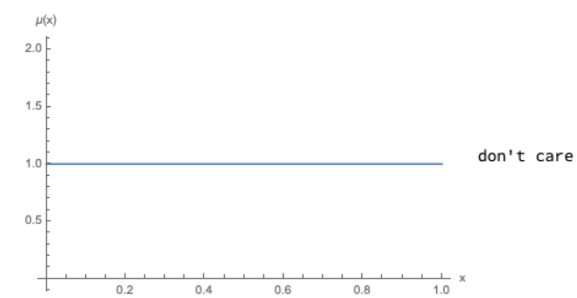

Fig. 1: The $don'tcare$ function

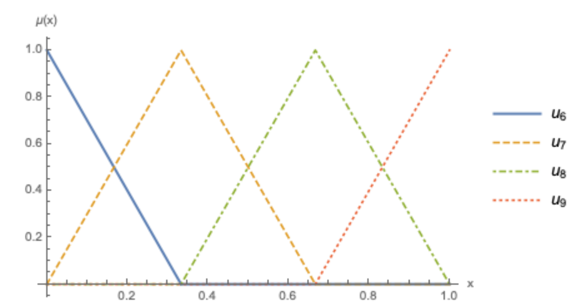

Fig. 2: Membership Functions with 4 intervals

The antecedent part of R_q is a set of antecedent fuzzy sets:

$$A_q = \{A_{qi} | i = 1, 2, ..., n\} \tag{6}$$

We define the compatibility grade of a pattern x_p with rule R_q as

$$\mu_{A_q}(x_p) = \prod_{i=1}^{n} \mu_{A_{qi}}(x_{pi}) \tag{7}$$

The consequent class C_q and rule weight CF_q are decided by the training data used to generate the rule. Assume we have m training patterns $x_p \in \{x_{p1}, x_{p2}, ..., x_{pm}\}$ from the M classes. We determine C_q and CF_q as follows:

First, we compute the confidence of fuzzy rule R_q to each class $c(A_q \Rightarrow Class\ h), h = 1, 2, ..., M$

$$c(A_q \Rightarrow Class\ h) = \frac{\sum\limits_{x_p \in Class\ h} \mu_{A_q}(x_p)}{\sum\limits_{p=1}^{m} \mu_{A_q}(x_p)} \tag{8}$$

Then the consequent class C_q as the class with which R_q has the largest confidence.

$$c(A_q \Rightarrow Class\ C_q) = max\{c(A_q \Rightarrow Class\ h) | h = 1, 2, ..., M\} \tag{9}$$

Rule weight CF_q of R_q is given by the difference between its consequent class and other classes.

$$CF_q = c(A_q \Rightarrow Class\ C_q) - \sum_{h=1, h \neq C_q}^{M} c(A_q \Rightarrow Class\ h) \tag{10}$$

Rule weight CF_q shows the quality of a classification result given by a fuzzy rule. Fuzzy rules with negative rule weight are abandoned.

B. Fuzzy Classifier

A fuzzy classifier S is a set of fuzzy rules. Given an input pattern x_p, the classification result C_w is produced by a winning rule R_w, chosen as follows:

$$\mu_{A_w}(x_p) \cdot CF_w = max\{\mu_{A_q}(x_p) \cdot CF_q | R_q \in S\} \tag{11}$$

C. Generate Fuzzy Rule From Training Patterns

At the initiation stage of our GBML algorithm, the population, set of fuzzy classifiers, is created from training data. We can use certain training patterns to create a fuzzy classifier and repeat the process to create a set of classifiers.

Each rule R_t in each classifier is generated from a training pattern x_t as follows:

We choose an antecedent fuzzy set according to each x_{ti} and form the antecedent part A_t of rule R_t so that x_t has the largest compatibility:

$$\mathsf{A}_{ti} = \arg\max_{\mu_j}\{\mu_j(x_{ti})\}, j = 1, 2, ..., I \tag{12}$$

Then, randomly change the antecedent fuzzy set to $dont't\ care$ according to a pre-specified probability p_{dc}. This step prevents overfitting and improves generalizing ability of fuzzy rules.

III. Hybrid Genetics-based Machine Learning Framework

After designing the fuzzy classifier, the next step is to approach our three objectives: maximizing number of correctly classified training patterns, minimizing number of fuzzy rules and minimizing total number of antecedent conditions in the fuzzy classifier S. Our current implementation only consider first two objectives.

We use a hybrid GBML algorithm to find the non-dominated rule sets. The genetic algorithm we used is based on [1]. This hybrid GBML is implemented in the framework of non-dominated sorting genetic algorithm II(NSGA-II) and it's a Pittsburgh-style algorithm. Besides,it use Michigan-style GBML to change a rule as the mutation operation.

In our genetic algorithm, the population is a set of fuzzy classifiers, and each individual is a single fuzzy classifier. The basic steps of our algorithm are:

1) Initialize the population with size N_{pop} using training data.
2) Generate N_{pop} offsprings by crossover operation. Then apply mutation on the offsprings.
3) Combine the original population and offsprings and keep the best N_{pop} individuals according to Pareto ranking and crowding measure.
4) Repeat 2) and 3) until the stopping criterion is met.

A. NSGA-II

NSGA-II, which is a common algorithm for multi-objective optimizing problems, is introduced in detail in [2]. Compared to ordinary genetic algorithms, it uses elite-preserving, Pareto ranking, and crowding measure to control the population. Elite-preserving is a strategy that we keep the individuals with the best performance in each iteration. Pareto ranking and crowding measure are used for non-dominated sorting in the three-objective situation. Individuals are ranked by Pareto-ranking first. If two individuals have the same Pareto-ranking, on with lower crowding measure is preferred.

1) Elite-preserving: Elite-preserving is a replacing strategy. The idea is that we replace individuals with bad performance in the original population by the outstanding offsprings, and keep preeminent parents to reserve their high performance gene.

2) Pareto ranking: Pareto-optimal front is the solution set that we want to find in the multi-objective optimizing problem, and its definition can been seen at [1]. Solutions are divided into non-dominated solution sets where there is no solution dominated by another solution in the set. Pareto ranking are assigned as the number of non-nominated solution sets that dominates it. In particular, the Pareto-optimal front is not dominated and is seen as the set of best solutions.

3) Crowding measure: Crowding measure is used to find a more evenly distributed Pareto set and maintain the diversity of the population. This will enhance the algorithm's ability to escape local optimal. An individual's crowding measure is computed as the sum of its distances to its neighboring vectors on both positive and negative directions for each objective. Priority is given to individuals with larger crowding measure.

B. Michigan-style GBML

Different from the Pittsburgh-style algorithm, Michigan-style GBML sees a single fuzzy rule as the population and sees the membership function as an individual. We use Michigan-style GBML as a mutation operator acting on a single rule to add diversity to the population.

IV. Asynchronous Parallel Distributed System Design

We propose an asynchronous parallel distributed system to accelerate our GBML algorithm.

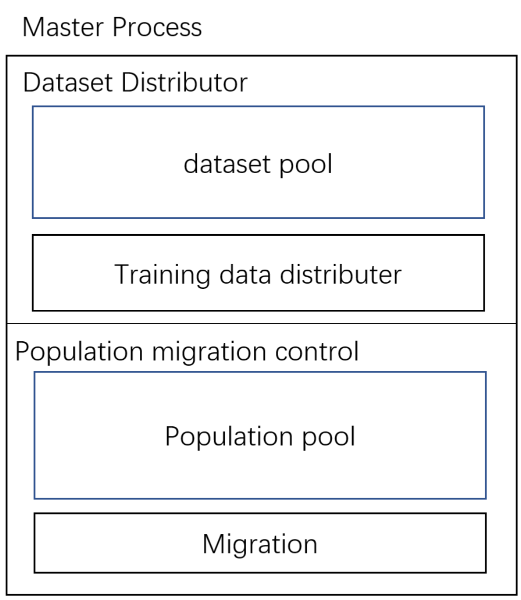

Fig. 3: The master process

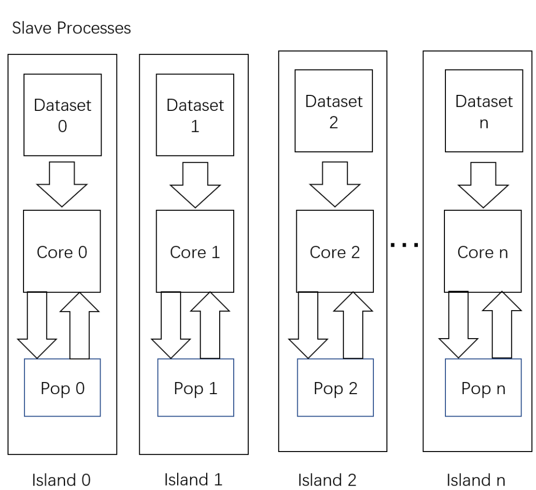

Fig. 4: The slave process

As shown in Fig.3 and Fig.4, the model includes 3 parts: dataset distributor, local island worker and population migration control. (For convenience, slave process are referred to as island in this report)

A. Master and Slave process

1) Master Process:

1) Read dataset and initialize the dataset pool
2) Distribute N_{land} samples to each island through dataset distributor layer.
3) Start P slave processes. Each works on a cpu core.
4) Wait for all islands finish their "init" part and return their population. Use these populations, build an initialized population pool.
5) Start each islands main part.
6) Listening each island, exchange(migrate) population on request.
7) Terminate when receiver a stopping message from all islands.

2) Slave Processes:

Init:

1) Receive training dataset from the master process.
2) Initialize population from the dataset.
3) Run the GBML for I_{init} iterations.
4) upload population to the master process.

Main:

1) Run the GBML for I_{update} iterations.
2) Upload the population to the master process and wait for the new population from migration control layer.
3) Terminates if stopping criterion is met and sent a stopping message to the master process.

B. Dataset Distributor:

The dataset distributor is responsible for the dataset distribution process, which delivers a portion of the training dataset to each island as local training data. During this process, the distributor randomly picks N_{land} data from the dataset pool and send it to a slave process. Repeat this procedure for each island.

C. Population Migration Control:

In the initialization stage, the population migration control layer send training data to each island and awaits returned population. When it receives returned population from all islands, it put together these populations and form a large population pool.

After the initialization, this layer delivers new population to each island on request and activates their "main" part. The new population is generated by randomly selecting N_{pop_i} individuals from the current population pool. Selected data will be deleted from the pool so that the size of the pool remains unchanged.

D. Asynchronous Population Migration

After initialization(synchronized), each island will run the GBML algorithm independently for I_{update} iterations and then updates its population through population migration control layer. In this way, population migration of each island is independent and that reduces the barrier time cost. We expect that our model could improve parallel efficiency especially when the update interval of the islands are different.

V. Computational Experiments

We tested our model in both 2 and 3 objectives and compared the results with non-parallel model and the synchronized parallel implementation [3]. Experiments are conducted on Gesture Phase Segmentation dataset [4] (32 attributes, 1743 attributes).

A. Pareto Front

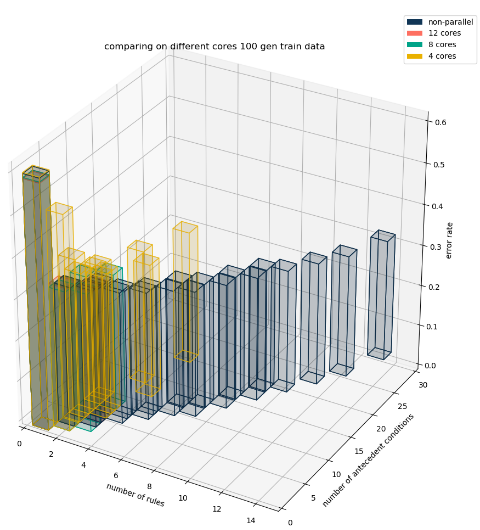

Fig. 5: Pareto Front on training data

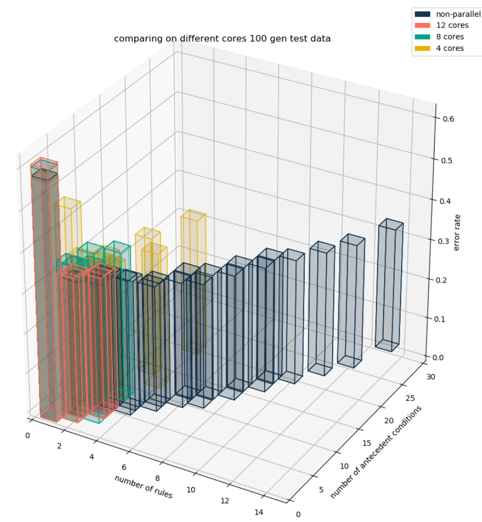

Fig. 6: Pareto Front on test data

As is shown by Fig.5 and Fig.6, our model is able to get results that is similar to non-parallel models. Note that with the increase of the number of cores, the ability of the model to obtain classifiers with more rules and antecedent fuzzy sets is decreasing, due to the decrease of sub-population. It's acceptable since our objective is to minimize the two objectives.

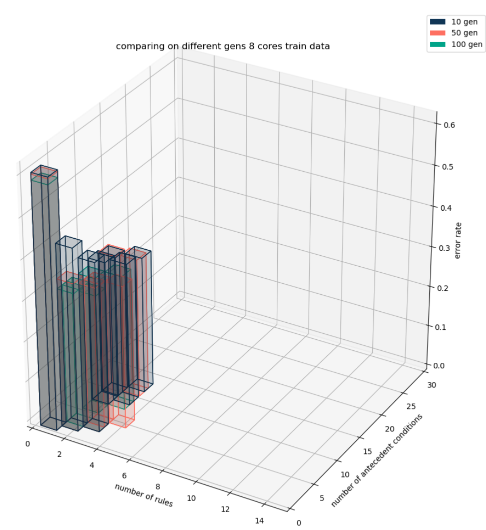

Fig. 7: Performance on test data using Gesture Phase

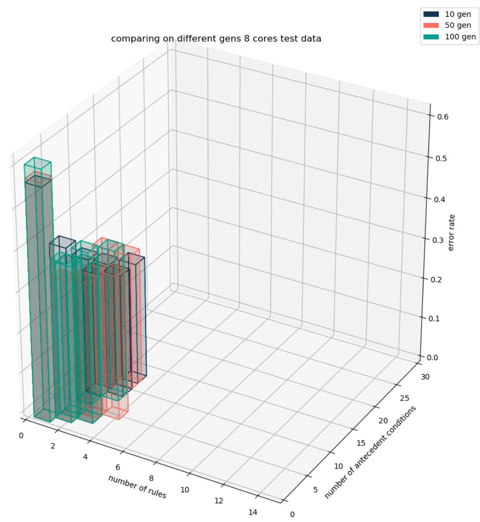

Fig. 8: Performance on test data using Gesture Phase

However, we observe significant performance drop on the Gesture Phase data set as is shown by Fig.8 and Fig.9. The dataset has 32 attributes and 1743 instances.

B. Computation Time

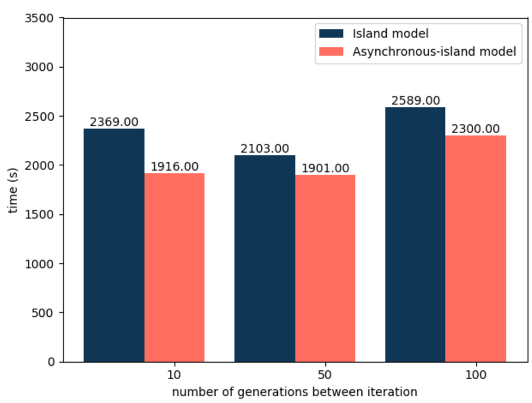

Fig. 9: Performance on test data using Gesture Phase

VI. Conclusion

Currently we have implemented a fuzzy rule-based classifier in a two-objective hybrid GBML framework. We will fix bugs (if any) of the fuzzy GBML algorithm, bring in another objective and integrate the algorithm into our proposed asynchronous parallel system in the final stage of this project.

VII. Contribution

- Bowen Zheng - Design & Implementation of parallel system
- Shijie Chen - Design & Implementation of fuzzy classifier, Design of parallel system
- Shuxin Wang - Design & Implementation of Hybrid GBML framework

References

[1] H. Ishibuchi and Y. Nojima, "Analysis of interpretability-accuracy tradeoff of fuzzy systems by multiobjective fuzzy genetics-based machine learning," *International Journal of Approximate Reasoning*, vol. 44, no. 1, pp. 4–31, 2007.

[2] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multi-objective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182–197, April 2002.

[3] Y. Nojima, Y. Takahashi, and H. Ishibuchi, "Application of parallel distributed implementation to multiobjective fuzzy genetics-based machine learning," in *Asian Conference on Intelligent Information and Database Systems*, pp. 462–471, Springer, 2015.

[4] R. C. B. Madeo, P. K. Wagner, and S. M. Peres, "Gesture phase segmentation data set," 2014.